Reliability Lessons Learned From GPU Experience With The Titan Supercomputer at Oak Ridge Leadership Computing Facility

Devesh Tiwari*, Saurabh Gupta*, George Gallarno†, Jim Rogers*, and Don Maxwell*

†Christian Brothers University
*Oak Ridge Leadership Computing Facility, Oak Ridge National Laboratory

ABSTRACT

The high computational capability of graphics processing units (GPUs) is enabling and driving the scientific discovery process at large-scale. The world's second fastest supercomputer for open science, Titan, has more than 18,000 GPUs that computational scientists use to perform scientific simulations and data analysis. Understanding of GPU reliability characteristics, however, is still in its nascent stage since GPUs have only recently been deployed at large-scale. This paper presents a detailed study of GPU errors and their impact on system operations and applications, describing experiences with the 18,688 GPUs on the Titan supercomputer as well as lessons learned in the process of efficient operation of GPUs at scale. These experiences are helpful to HPC sites which already have large-scale GPU clusters or plan to deploy GPUs in the future.

1. INTRODUCTION

GPU architecture inherently provides more parallelism than traditional CPU architecture. Traditionally, graphics processing applications, such as video games and image processing applications, have utilized GPUs for faster processing and rendering. However, with the increased thrust towards general-purpose computing on graphics processing units (GPGPU), GPUs have started to influence high performance computing (HPC) systems as well. Researchers have demonstrated that GPU parallelism can be exploited for a variety of scientific applications. Besides performance advantages, GPUs are also highly energy-efficient (i.e., flops/watt). These inherent advantages make GPUs an attractive computing architecture for large-scale HPC centers that aim toward exaflop performance under a limited power budget. Consequently, HPC centers are beginning to adopt GPUs in their mainstream large-scale HPC systems. Titan, the world's second fastest supercomputer, consists of 18,688 NVIDIA Tesla K20X GPUs that computational scientists use routinely to perform scientific simulations and data analysis.

Applications from various domains, including combustion, condensed matter physics, nuclear sciences, and molecular dynamics, have been ported to scale on the Titan supercomputer and have efficiently exploited the GPU parallelism at scale [1]. There are numerous efforts documenting and sharing the application porting and optimization experience on GPU hardware [22]. However, there is still limited understanding about the reliability characteristics of GPUs and its impact on scientific applications at large-scale. It is very critical to understand the reliability characteristics of GPUs and its impact on the HPC workloads. HPC workloads are typically fairly long running simulations that often rely on checkpointing mechanism to continue making forward progress even in the case of failures. Therefore, understanding the characteristics of GPU related errors on large-scale systems are likely to benefit both system operators, designers, and end users.

In this paper, we share our experience with the Titan supercomputer's GPUs as these GPUs have been in production for more than two years. We share our experience based on different error logging methodologies with regards to GPUs. We look at the GPU system failures specifically to see how they impact the applications (e.g., execution interruption). We discuss the current state of practice in GPU error logging, and some of the current challenges and issues. We show the frequency of these errors and how they may impact the applications. We address questions such as, which GPU errors are more dominant than others, and what are some of the most common driver related errors? We study time inter-arrivals among different types of GPU errors, and how different type of errors are distributed at different computing granularity (e.g, at the cabinet and cage level). We investigate the correlation between resource utilization and different kind of errors to understand the impact and implication of these errors. This work also studies how application errors are distributed in space over the job's node allocation and how this changes over time. As we present our analysis and discuss our findings, we point out how our study could be useful for other current and future GPU-enabled HPC centers, system designers and programmers. We believe that insights derived from our field data analysis carry significant implications for current and future HPC computing facilities, and researchers focusing on resilience of GPU workloads.

SC '15, November 15-20, 2015, Austin, TX, USA

 ISBN 978-1-4503-3723-6/15/11…$15.00

DOI: http://dx.doi.org/10.1145/2807591.2807666

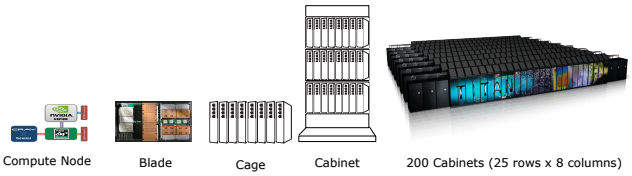

Figure 1: Physical organization of the Titan supercomputer.

Table 1: GPU hardware related errors.

GPU Error	XID
Single Bit Error (corrected by the SECDED ECC)	–
Double Bit Error (detected by the SECDED ECC, but not corrected)	48
Off the Bus	–
Display Engine error	56
Error programming video memory interface	57
Unstable video memory interface detected	58
ECC page retirement error	63,64
Video processor exception	65

2. BACKGROUND AND METHODOLOGY

This section provides a brief background on the architecture of the Titan supercomputer, GPU architecture details, built-in resilience support, and GPU related errors. Subsequently, we also describe our data collection and analysis methodology.

2.1 Titan Supercomputer Organization and GPU Architecture

First, we describe the physical organization of the Titan supercomputer (Fig. 1). Then, we briefly describe the details of K20X GPU architecture details.

The basic building block of the Titan supercomputer is a node consisting of a single AMD CPU and single NVIDIA GPU. Each CPU is a 16-core AMD Opteron 6274 processor with 32 GB of DDR3 memory. Each GPU is a NVIDIA K20X Kepler architecture-based GPU with 6 GB of GDDR5 memory. One custom, high-speed interconnect Gemini router is shared by two nodes as shown in Fig. 1. Four nodes comprise one blade/slot and each cage has eight such blades. Each cabinet has three cages. The Titan supercomputer consists of 200 such cabinets organized as a total of 25 rows and 8 columns (Fig. 1).

The Kepler K20X GPU architecture, GK110 processor used in the Titan supercomputer's GPUs, is composed of multiple streaming multiprocessors (SM). These SMs are the basic building blocks of the GPU. Each SM has dedicated shared memory and L1 cache regions. Each SM also has a dedicated register file that can be accessed only by the threads in the same SM. SMs have access to the shared resources as well, such as L2 cache and the device memory. The thread block scheduler dispatches one or more blocks of threads to a particular SM. Each SM has multiple Compute Unified Device Architecture (CUDA) cores. Each CUDA core executes only one thread at a time and has access to shared memory and L1 cache region. Specifically, the K20X GPU has 2688 CUDA cores (28nm process technology). There are a total of 14 SMs and 192 CUDA cores within each SM. A single GPU has 3.95 Tflops single precision peak performance and 1.31 Tflops double precision peak performance. The on-chip memory hierarchy on a GPU consists of each SM having 64K registers, 64KB of combined shared memory and L1 cache, and 48KB of read-only data cache. All SMs on the GPU share a 1536 KB L2 cache and 6GB GDDR5 memory.

Major structures of a GPU, such as device memory, L2 cache, instruction cache, register files, shared memory, and L1 cache region, are typically protected by a Single Error Correction Double Error Detection (SECDED) ECC. But, it may not be possible to protect all the regions. For example, logic, queues, the thread block scheduler, warp scheduler, instruction dispatch unit, and interconnect network are not ECC protected. We note that this opens up the possibility of a soft-error causing side-effects (crash or silent data corruption), but still not being caught by the ECC mechanism. However, the chip area covered by an unprotected structure is much smaller in comparison to the caches and other memory structures, hence, the probability of such failure events is fairly low. In K20X GPU architecture, the register files, shared-memory, L1 and L2 caches are SECDED ECC protected, while the read-only data cache is parity protected.

2.2 GPU Errors, Collection and Analysis Methodology

Our study covers more than 280 million node hours worth of console logs from the Titan supercomputer. We study Titan's system logs collected over the period of Jun'2013 to Feb'2015. The console logs from the Titan supercomputer are parsed using simple event correlators (SEC) on software management workstations (SMW) to log critical system events. This is a comprehensive log of critical system events that alerts the system operators of unexpected/undesired behavior. In this study, we focus specifically on GPU related events that may affect an application's execution. An application may encounter a GPU error for multiple reasons, for example, an application bug, driver bug, hardware or radiation-induced bit corruptions. Therefore, these errors can be classified in two categories: (1) GPU related system failures/errors that are caused by hardware or cosmic rays (Table 1), (2) GPU related errors that are primarily caused by the application error, driver issue, thermal issue, etc. (Table 2).

We note that some errors may appear in both tables since determining precise source of a particular error is not always possible. We also point out that analyzing GPU errors due to an application or a runtime system can be misleading as an excessive number of errors may be occurring due to the debug job runs, program bugs or bad programming practices. One can refer to a complete list of different GPU errors logged with different XID codes here [2]. Later, we also describe in detail the challenges and issues involved in recording these events. We note that in addition to soft-errors, there may be system integration related errors that may not be specific to the GPU micro-architecture. For example, "Off the bus" error is related to losing the connection to the host. It may be caused due to system integration issues, and is not specific to the GPU micro-architecture design.

Some error events may be followed by multiple system error events shortly after the initial errors occurrence. The-

Table 2: GPU software/firmware related errors.

GPU Error (possible cause)	XID
Graphics Engine Exception (Driver, User App, System Memory or FB Corruption, Bus Error, and Thermal Issue)	13
GPU memory page fault (Driver or User App error)	31
Invalid or corrupted push buffer stream (Driver, User App, Memory or FB Corruption, Bus Error, and Thermal)	32
Driver firmware error (Driver error)	38
Video processor exception (Driver error)	42
GPU stopped processing (Driver error)	43
Graphics Engine fault during context switch (Driver error)	44
Preemptive cleanup, due to previous errors (Driver error)	45
Error programming video memory interface (Driver error)	57
Unstable video memory interface detected (Driver error)	58
Internal micro-controller halt (old driver error)	59
Internal micro-controller halt (new driver error, thermal)	62

refore, there may be one real "parent" event and multiple "child" events. One can exclude these "child" error events by applying a filtering to avoid bias in failure characterization. We perform a filtering scheme similar to other works [15, 21, 30, 32], but also study the impact and effect of these "child" error events in order to better understand the GPU related errors. We note that such phenomena in the context of GPU errors have not been studied before.

In addition to console logs, the GPU errors were also collected by running *nvidia-smi* utility on all the GPU nodes. This is primarily because console logs do not capture the single bit error information. However, note that this utility is a snapshot information and doesn't timestamp all the single bit errors. In addition to reporting the single bit errors, nvidia-smi output also includes double bit and ECC page retirement related errors. Therefore, we use both these methods of data collection to quantify and analyze the characteristics of GPU errors. Furthermore, we have very recently developed a framework where we can take *nvidia-smi* snapshots before and after each batch job. This helps in identifying the single bit error counts, location and its correlation with different types of jobs. We have also utilized job logs and resource utilization logs to study these issues in detail.

3. QUANTIFICATION, IMPACT AND ANALYSIS OF GPU ERRORS

First, we quantify and characterize the GPU errors related to hardware. Next, we discuss software/firmware related GPU errors and their key observed characteristics. Then, we discuss the spatial and temporal characteristics of single bit errors. As we discuss our findings and analysis, we point to the significance of our findings for designers and architects of future exascale systems.

3.1 Understanding Hardware Related Errors

GPU Double Bit Errors (DBE) Double bit errors are soft errors that can be caused by a variety of reasons including cosmic ray strikes and voltage fluctuations. The SECDED mechanism in K20X GPU architecture can only detect, but not correct these errors. Note that a double bit

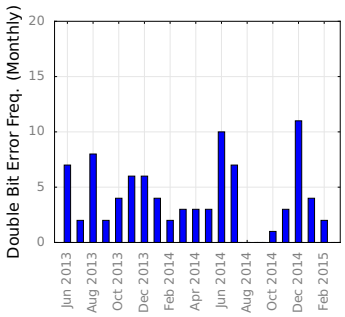

Figure 2: Monthly frequency of double bit errors on the Titan supercomputer (Jun'2013-Feb'2015).

error does not necessarily mean that it will produce incorrect program output or cause the execution to terminate prematurely. But, when a DBE is encountered, SECDED mechanism always crashes the program. This is because SECDED mechanism can not correct such errors and hence, can not guarantee correct execution beyond the point of error detection.

We analyze the temporal and spatial characteristics of GPU double bit errors (DBE) in this subsection. Fig. 2 shows the monthly occurrences of DBEs on the Titan supercomputer since the machine went into production with the GPUs. On average, one DBE occurs approximately every seven days (approx. 160 hours). We also found that an excessively high number double bit errors did not occur on particular day or set of days, which indicates that these errors are not bursty in nature. Overall, the estimated MTBF based on the vendor datasheet would be significantly lower for our system compared to what our field data indicates. We believe that this is in part due to (1) early rigorous, stress, acceptance tests that weed out bad GPUs, (2) GPU architecture has matured significantly, their soft-error resilience has improved [30], and vendors have evolved to develop better in-house stress tests.

We identify cards which incur double bit errors and put them out of the production use (such cards undergo further rigorous testing in a hot-spare cluster before being returned to the vendor after encountering a threshold number of DBEs). We have returned the GPUs to the vendor after they were stress tested in the hot-spare cluster and GPU system failures were encountered. Such errors would have likely occurred in production, but we avoided that by moving error-encountering cards to the hot-spare cluster. It is expected that swapping out error-prone cards will lead to improved system MTBF. However, we note that accurately quantifying the impact of such replacement is often very hard, since it is difficult to predict how many errors would have been avoided in future after replacement.

Fig. 3(a) shows the spatial distribution of double bit errors in the rows and columns of the Titan supercomputer's cabinets. Since DBEs are fairly rare event, an uneven distribution in space is not completely surprising. However, interestingly, we observed that DBE seem to occur more in the upper cages of the cabinet than the lower cages (Fig. 3(b)). We suspect that it indicates the DBE's sensitivity toward temperature. We note that due to the power/cooling set up in the Titan supercomputer higher cages are typically hot-

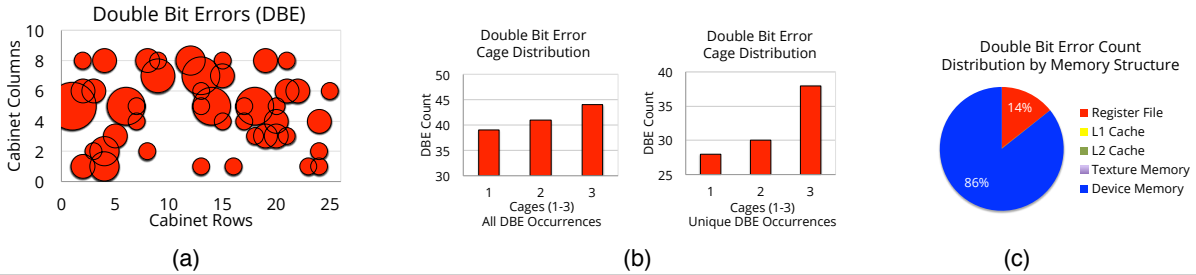

Figure 3: Spatial distribution, cage-wise distribution and breakdown by memory structure type of double bit errors.

ter than the lower cages in the same cabinet. For example, we found that the GPUs in the uppermost cage are on an average more than 10°F hotter than the GPUs in the lowermost cage, as per a snapshot taken by the nvidia-smi utility. However, it is fundamentally challenging to establish a correlation between temperature and DBEs because some GPU cards may inherently be more prone to DBEs even if they are situated in the lower cages. Fig. 3(b) also shows the number of distinct GPU cards experiencing DBEs categorized by cage location within a cabinet. Counting only one DBE error per card addresses the previously mentioned issues, and shows that the trend only gets stronger.

Observation 1. *MTBF of double bit errors on the Titan supercomputer is quite high (approx. 160 hours, i.e., approx. one DBE per week). We believe that improved resilience of GPU architecture, acceptance testing, and rigorous hot-spare testing contribute toward such a high MTBF.*

One may attempt to compare the reliability of GPU-based systems with current and past CPU-based systems. However, we note that it is challenging to perform a fair comparison across different systems due to significant differences in the underlying micro-architecture, design, process-technology, system-software stack, etc. In general, the reliability of OLCF systems have continued to improve even with the introduction of new heterogeneous architecture.

Our double bit error analysis so far is based on the console logs. Now, we attempt to confirm our findings against the output of the nvidia-smi utility (static snapshot). Unfortunately, the counts do not match exactly. Nvidia-smi output reports fewer number of DBEs than our console log filtering method. We suspected that this may be because of a driver issue where a double bit error causes the node to shut down before the DBE incident is logged in the NVML InfoROM that is queried by the nvidia-smi utility. Our interaction with the vendor confirmed this explanation. We observed some other inconsistencies in the error logging as well. Nvidia-smi reports a greater number of double bit errors than single bit errors for some cards during the same time-period. We can not verify this completely, since single bit errors are not recorded to the console logs. But, the theoretical probability of a double bit error happening is lower than the probability of single bit error event. Therefore, it can be attributed to inconsistency in logging. However, we would like to emphasize that the nvidia-smi utility is highly useful for operational purposes, and such utilities have come a long way from their inception, continuously adding new and useful features – easing the job of system operators.

Observation 2. *Nvidia-smi utility has evolved as a very useful tool for GPU error monitoring. However, there are some challenges in getting it to accurately account for all DBEs. Therefore, one should not entirely rely on nvidia-smi utility for analyzing double bit errors. The low-overhead of the nvidia-smi utility makes it particularly attractive for other monitoring purposes.*

Next, we study the breakdown of double bit error per structure type (i.e., register file, L1 cache, L2 cache, texture memory, and device memory). We did this by decoding the error log for DBE occurrences, instead of using the nvidia-smi output which readily provides this information (without any decoding). Previously, we explained why the console logs are more reliable for analyzing double bit errors. As shown in Fig. 3(c), we found that 86% of double bit errors happen in the device memory. This is not surprising as device memory is larger than other memory structures by orders of magnitude. Interestingly, the remaining 14% of the double bit errors happen in the register files only, one of the smaller memory structures on the GPU. We speculate that a less effective interleaving technique may be employed to reduce the chances of registers getting corrupted by double bit errors. However, such reasoning are speculative in nature since the vendor considers such information to be proprietary. More effective interleaving techniques may cause more area and time overhead – causing them to be less attractive in fabrication and from the access-latency standpoint.

Observation 3. *We observed that 86% of the DBEs happen in the device memory, while the remaining 14% happen in the register file – despite it being a much smaller structure. This indicates that vendors should continue to improve DBE resilience of the register file structure for future exascale systems.*

GPU Off The Bus and ECC Page Retirement Errors

Next, we analyze "Off the bus" and "ECC page retirement" errors. Off the bus errors are related to system integration issue and not architecture-specific. Off the bus errors cause the applications to crash as host looses connection to the GPU. Fig. 4 shows that Off the Bus errors only dominant the period before December 2013. A system integration issue with the GPU cards was identified, and subsequently resolved by soldering the cards. Consequently, these errors have almost become negligible. Notably, these errors were mostly clustered and that's when the criticality of the issue was identified. We also note that these errors are fairly dis-

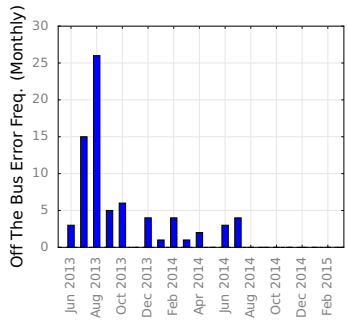

Figure 4: Monthly frequency of Off the bus error.

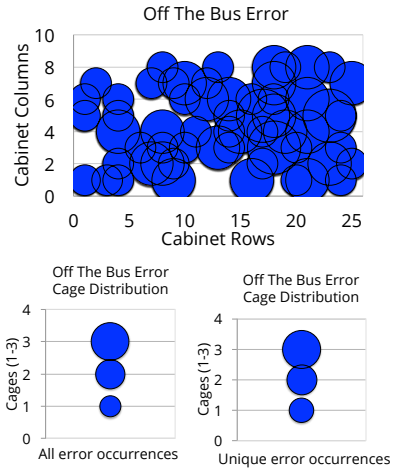

Figure 5: Spatial Distribution of Off the bus error.

tributed across the machine (Fig. 5). In particular, "Off the Bus" errors show strong sensitivity towards temperature as they tend to occur in upper cages more frequently (Fig. 5). As noted earlier, the GPUs in the uppermost cage are on an average more than 10°F hotter than the GPUs in the lowermost cage, as per a snapshot taken by the nvidia-smi utility. The difference between all occurrences and unique card occurrences is small because Off the Bus errors do not tend to reappear on the same card.

Observation 4. *"Off the bus" error used to be a frequent GPU failure event that was caused by system integration issues and not because of the inherent GPU micro-architecture design. Such issues were resolved with soldering. The upper cages in the cabinet experience more such errors than lower cages, indicating the possibility of temperature sensitivity. This observation was used for improved job scheduling for large GPU jobs at OLCF.*

ECC page retirement error is relatively new XID introduced by NVIDIA. As shown in Fig. 6, it has started appearing only since Jan'2014. ECC page retirement error is supposed to happen under two circumstances: (1) one double bit error or (2) two single bit errors in the same page. Page address is stored in the InfoROM and when the driver loads it can get to know these page addresses and framebuffer can ensure that these pages are not used by the application. This essentially improves the life of the card. The application crashes

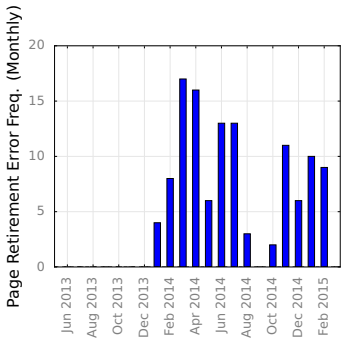

Figure 6: Monthly frequency of ECC page retirement error.

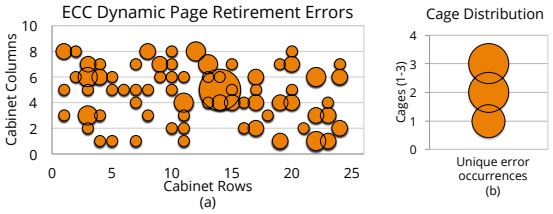

Figure 7: Spatial Distribution of ECC page retirement error.

in the first case, but not in the second case.

The spatial and temporal characteristics of ECC page retirement error are similar to previously discussed errors. The spatial distribution is not uniform, and cards in the upper cages are slightly more likely to experience errors (Fig. 7). Next, we attempt to understand how likely it is that a DBE is followed by an ECC page retirement error. Note that the the DBE occurrences happening only after the period Jan'2014 are accounted toward this analysis, and there may be cases where no ECC page retirement error occurs between two DBEs. Fig. 8 shows the distribution of ECC page retirement errors under different time intervals since the last DBE. Interestingly, 18 page retirement happens within 10 minutes of a DBE occurrence, while only 1 event happened between 10 minutes and 6 hours. Therefore, an ECC page retirement is likely to happen soon after the DBE occurrence, but if some time (10 minutes in this case) has passed, the node should be considered less likely to experience an ECC page retirement error. Cases where ECC page retirement occurs much later after the DBE occurrence (more than 10 minutes, 18 such cases in our scenario) are likely caused by two SBEs happening in the same page. We found that there were 17 instances when no ECC page retirement happened between two successive DBEs. It is not fully understood at this point if this is intentional or an issue with the error logging.

Observation 5. *"ECC page retirement" error is a recent introduction in the list of GPU errors. It effectively improves the quality of card by retiring degrading cells. Its inter-arrival time relationship with double bit errors leads to more information about the multiple single bit errors occurring on the same page. System operators have to keep updating their log parsing rules to account for such new introductions and understand the implications.*

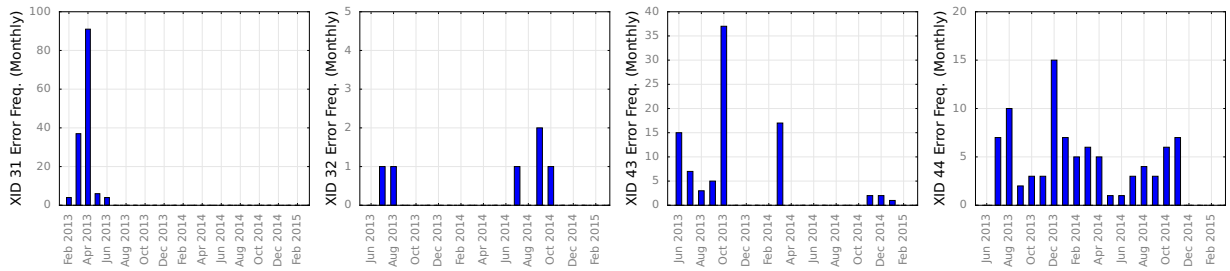

Figure 9: Error frequency of XID 31, 32, 43, and 44. (GPU memory page fault, Invalid or corrupted push buffer stream, GPU stopped processing, and Graphics Engine fault during context switch, respectively).

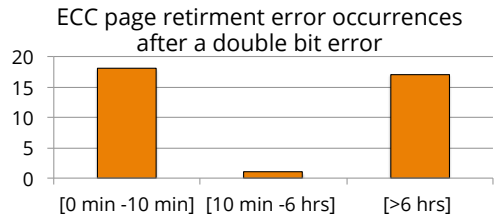

Figure 8: Occurrence of ECC page retirement error following a DBE. Cases where no ECC page retirement error happens between two DBEs are not shown here.

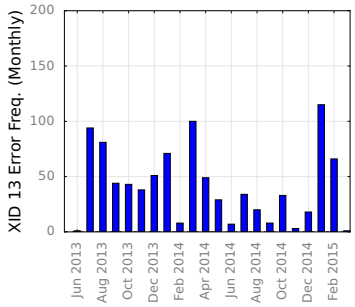

Figure 10: Error frequency of XID 13 (Graphics engine exception).

3.2 Understanding Software, Firmware, and Application Related GPU Errors

In this section, we characterize the impact of GPU errors that are not caused by the GPU hardware. We present a set of results showing the frequency of different XID errors. Each XID error has its own possible source of cause, as described in Table 2 and NVIDIA's XID documentation [2]. Fig 10 shows the frequency of XID 13 error that has user application listed as one the possible causes in the NVIDIA's XID documentation [2]. These errors often occur in bursts, i.e., multiple errors happening on the same day. We suspect this behavior may be caused by the debug and test runs by the end users. Sudden rise in such errors may also correlate with domain scientists' project or paper deadlines since we observed a significantly more number of failures in certain weeks. However, it is not necessary that all events occurs

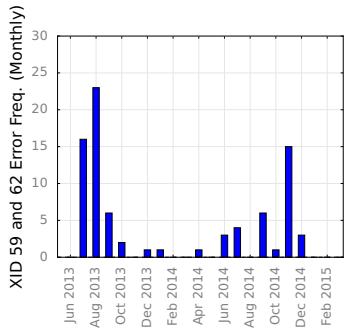

Figure 11: Error frequency of XID 59 and 62 (Internal micro-controller halt).

due to an application error.

Some XID errors, which are primarily caused by driver issues, are less frequently observed. For example, invalid or corrupted push buffer stream and driver firmware error have occurred less than ten times during the production run (Fig. 9). Some driver related errors do not occur at all (e.g., XID 42). On the other hand, certain driver related errors such as GPU stopped processing, graphics engine fault during context switch, and micro-controller halts occur more frequently (Fig. 9 and 11). Note that these errors are primarily driver related errors and may not show as much bursty behavior (Fig. 11) as the user application related XIDs do (e.g., XID 13, graphics engine exception).

Observation 6. *User application caused XID errors are bursty in nature and are frequent, while driver related XID errors are not bursty and occur relatively less frequently. We observed that only a few driver related XID errors occur more frequently than others, such as micro-controller halts, graphics engine fault during context, and GPU stopped processing.*

Next, we provide an understanding of how an application error propagates over time on different nodes. First, we observed that user application related errors are reported on all the nodes allocated to the job, instead of just one node where the problem may have occurred. We take XID 13 error as an example to illustrate the change in spatial distribution as we apply time filtering at different granularities. Fig. 12 (top) shows the spatial distribution of XID 13 for the case when no filtering is applied, i.e., XID 13 error appearance

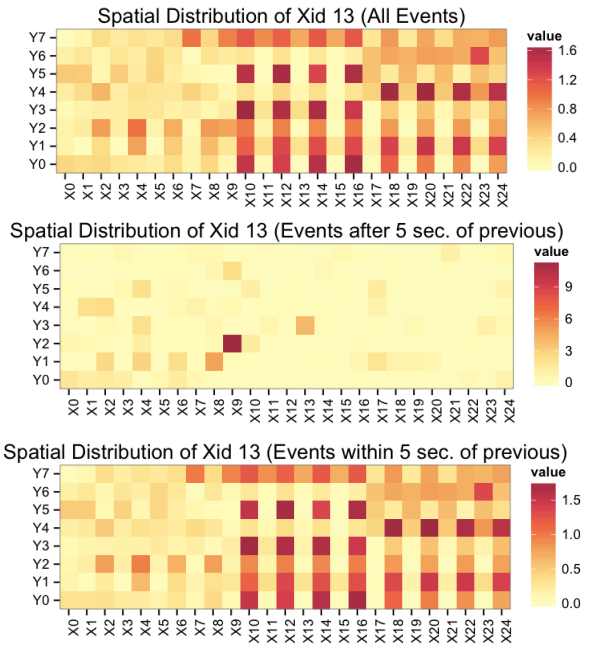

Figure 12: Spatial distribution of XID 13 error with different time threshold filtering.

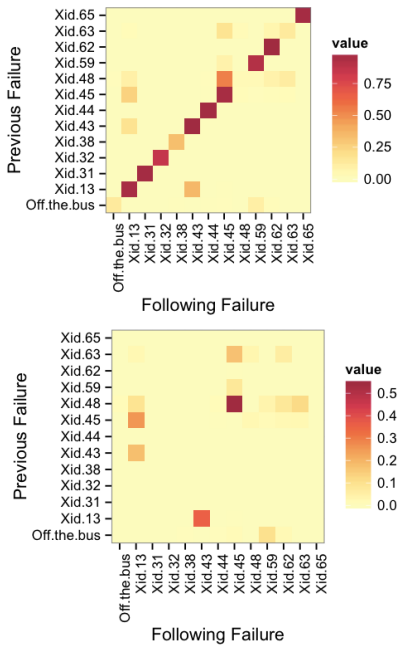

Figure 13: Temporal re-occurance relationship between different XID events (for a time window of 300 sec.)

on all the nodes are accounted for. Fig. 12 (middle) shows the spatial distribution of XID 13 for the case when a five-second filtering is applied, i.e., any XID 13 error appearing in the console log after a previously encountered XID 13 is ignored if the time difference is less than five seconds. Effectively, this counts only one XID 13 event per job because the job would crash after the error. Fig. 12 (bottom) shows the spatial distribution of XID 13 events that occurred within the five-second window.

First, we observe that the XID 13 errors exhibit uneven spatial distribution (Fig. 12 (middle)). This indicates that debug jobs may be unevenly distributed across the cabinets. A more surprising observation is that both Fig. 12 (top) and (bottom) show a distinct pattern where alternate cabinets have greater event density. This is due to folded-torus cabling used in Titan [8] to avoid uneven length of cables in the classic 3-D torus. This causes nodes within the same job to be allocated in this alternating manner in the 3-D torus Gemini interconnect resulting in such a pattern. We also found that five seconds was a reasonable interval within which all nodes in the same job reported the error.

Observation 7. *Spatial distribution of user application related XID errors can be understood using the physical organization and interconnect of the system. Typically, we observed that the errors appear on all the nodes allocated to the job within five seconds.*

While XID 13 is not supposed to be caused by hardware, we found an instance where a particular node was repeatedly encountering XID 13, irrespective of the application scheduled on it. Since XID 13 is not associated with hardware, we did not take the node down immediately after the error event. We later confirmed, after the node diagnostic testing, that it was indeed a problem related with the hardware.

Observation 8. *It is extremely challenging to pinpoint the source (hardware, software, firmware, thermal issues etc.) of any XID error. NVIDIA has performed rigorous tests to come up with the list of possible sources of these XID errors. But, we recently observed a case where XID 13 was due to hardware instead of other problems as previously assumed/known.*

We investigate the parent-child relationship among different Xid events in Fig. 13. The figure shows the fraction of Xid events shown on 'Previous Failure' axis that will observe an event shown on 'Following Failure' within a 300 sec window. We use a large time window here in order to allow more time for child events to show up after a parent event. The top heatmap includes all event pairs while the bottom heatmap excludes the pairs of same type of events. We can observe that a DBE (XID 48) is likely to be followed by XID 45 and XID 63, and XID 13 is likely to be followed by XID 43. We also observe that many XID errors often occur multiple times (or at multiple nodes in the same job) in the console logs after the original XID event. This can be observed by the entries along the diagonal which show high values for these XIDs. On the other hand, off the bus, XID 38, XID 48 (DBE), and XID 63 do not show multiple occurrences within a 300-second time window. This implies that these events are relatively more isolated in nature.

Observation 9. *Doing correlation analysis between different types of errors help us understand which errors are more likely to be followed by another type of error, which errors occur in isolation and may not have precursor events, etc.*

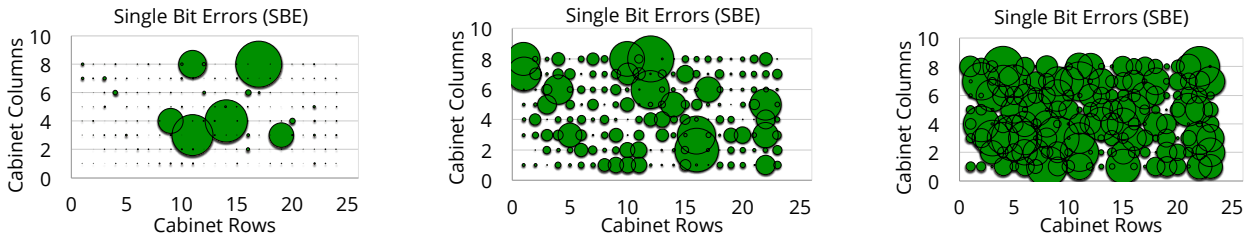

Figure 14: Spatial distribution of SBEs based on number of SBE offenders: All GPU cards that ever encountered any SBE (left), top 10 most offending GPU cards removed (middle), and top 50 most offending GPU cards removed (right).

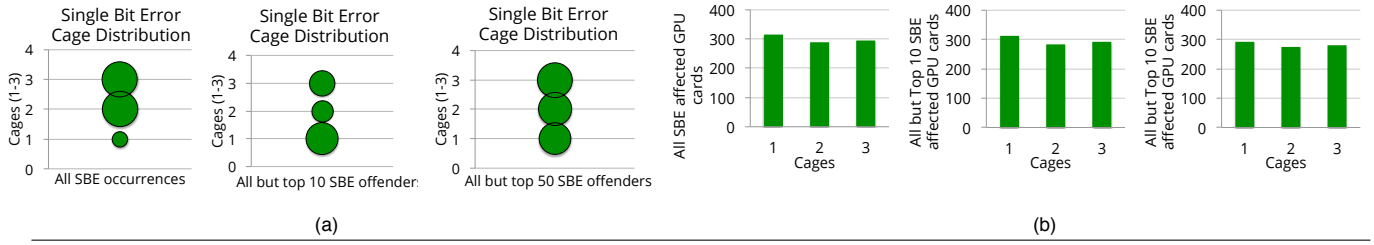

Figure 15: Cage distribution of single bit errors and SBE experiencing distinct GPU cards: Frequency of single bit errors (a), number of distinct GPU cards affected by SBE (b).

3.3 Understanding Single Bit GPU Errors

In this section, we study the characteristics of single bit errors. Recall that single bit errors are corrected by the SEC-DED ECC mechanism, so they do not affect the program's execution. However, they are an indicator of error resilience of the hardware and have implications toward the ECC page retirement error as described earlier. We verify this fact by looking at the nodes that experience high number of SBEs and find that some of those nodes have experienced ECC page retirement error in the console logs. But, it should be noted that nvidia-smi output aggregates the SBE count over a period of time, therefore, we cannot directly relate it to the console log events for causal relationship.

Fig. 14 shows the spatial distribution of single bit errors. The leftmost figure clearly shows a highly skewed distribution of SBEs. This is primarily because a small fraction of GPU cards are responsible for almost all of the SBEs. To address this issue, we excluded the 10 GPU cards which have experienced the most number of SBEs and plotted the spatial distribution (Fig. 14(middle)). We observe that the skewness is reduced, and it goes further down when we eliminate the top 50 GPU cards. This shows that some cards experience significantly more single bit errors than others. We found that removing the top 50 cards from the spatial distribution produces an almost homogeneous distribution. However, we also found that less than 1000 cards have ever experienced a single bit error (less than 5% of the whole system).

Next, we investigate if, similar to the double bit errors, single bit errors also tend to occur more frequently in the top cage in a cabinet. Clearly, the uneven distribution of SBEs among GPUs make this analysis challenging. If all GPU cards are considered, then single bit errors occur most in the the topmost cage (Fig. 15(a)). However, this trend quickly reverses when we exclude the top 10 SBE experiencing GPU cards. Interestingly, when we remove the top 50 SBE offenders, we obtain a fairly homogeneous distribution of single bit errors across the cages, like the spatial distribution across rows and columns. However, this analysis may still have some GPU cards experiencing multiple SBEs. Therefore, we plotted the distribution of "distinct" cards across the cages for all three cases (Fig. 15(b)). Interestingly, GPUs experiencing any single bit error are distributed equally across the cages for all three cases. This indicates that single bit errors may be less sensitive toward the location within a cabinet, instead some cards are inherently likely to experience more SBEs than others (as illustrated by Fig. 14).

Observation 10. *Single bit errors show a highly skewed distribution on the Titan supercomputer. However, when 50 top SBE offending nodes are removed, the distribution becomes relatively homogeneous in space. It remains challenging to illustrate that cards in higher cages are prone to more single bit errors. It appears that some cards are inherently more prone to SBEs rather than due to their location.*

4. UNDERSTANDING THE CORRELATION BETWEEN SINGLE BIT ERRORS AND GPU RESOURCE UTILIZATION

In this section, we want to understand the relationship between single bit error (SBE) and GPU resource utilization. In particular, we investigate the correlation between GPU core hours, number of nodes and memory consumption with SBEs. Recall that double bit errors happen relatively infrequently, consequently, it is quite challenging to draw statistically sound conclusions about the correlation between DBE and GPU resource utilization. On the other hands, we observe SBEs in the order of hundreds per day,

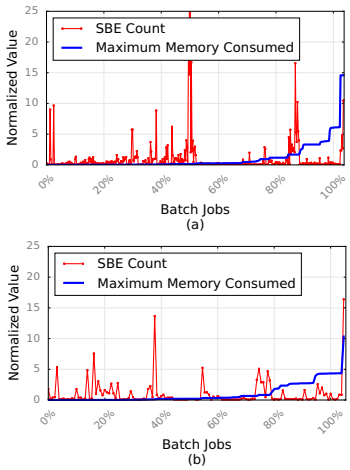

Figure 16: Maximum memory consumption and single bit errors: all jobs (a), and excluding jobs that used any of the 10 GPU cards experiencing the most single bit errors (b). Batch jobs are sorted based on the total memory consumption. Respective curves have been normalized to the average of total memory consumption and SBE count.

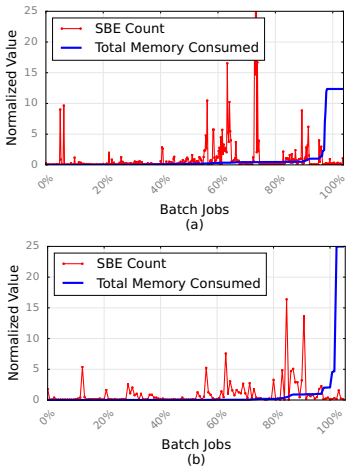

Figure 17: Total memory consumption and single bit errors: all jobs (a), and excluding jobs that used any of the 10 GPU cards experiencing the most SBEs (b).

making it statistically meaningful to investigate the correlation. For this analysis, we used recently deployed method of collecting SBE counts on a per batch job basis for the period of over a month, and the correlation was studied between SBEs and GPU resource utilization. We note that the SBE counts can not be collected on a per aprun basis instead it is collected on a job basis since the nvidia-smi output is run before and after the job script, irrespective of number of apruns within the job script.

Fig. 16, 17, 18, and 19 have been sorted by maximum memory consumption, total memory consumption, number of nodes, and the GPU core hours, respectively. We sorted batch jobs in this order to better visualize the correlation. We also note that the values have been normalized to average value of the respective metrics because the ranges are quite large, making it hard to visualize how does the SBE count change with the increase in a particular resource utilization. The first case in each plot considers all the GPU jobs. However, as we noted earlier, some GPU nodes may experience significantly more SBEs than other nodes, and hence, can potentially skew our analysis. Therefore, our second case excludes jobs that used any of of top 10 SBE offender nodes. We found that removing top 50 offending nodes showed similar results as removing top 10 SBE offenders (but results not shown due to space restrictions).

From Fig. 16 and 17, we observed that maximum and total memory consumption had very little correlation with the SBE count (both the Spearman and Pearson coefficient were less than 0.50 with p-value <0.05). We explain this observation by looking at the memory structures where these SBEs are happening. Most of the single bit errors happen in the L2 cache despite its much smaller size than the device memory. This could help us explain why SBE count may not always exhibit strong correlation with the memory utilization.

Observation 11. *The single bit errors show very weak correlation with the GPU memory utilization. This is caused by the SBEs occurring predominately in on-chip memory structures. However, current GPU activity accounting tools do not provide cache and register file level activity tracking to investigate this issue in more detail.*

SBE count shows good correlation with the number of nodes and GPU core hours for the case where all jobs are considered (the Spearman coefficient is 0.57 and 0.70, respectively). We note that the Pearson coefficient was still low because the correlation may not be linear in nature and hence, is better captured by the Spearman correlation analysis. SBE occurrences tend be more strongly correlated with GPU core activity than memory utilization. Interestingly, removing the jobs that used any of the top 10 offenders weakens the correlation for both number of nodes and GPU core hours (reducing it to below 0.50 for the Spearman coefficient).

Observation 12. *Based on our experience, statistically establishing a correlation between GPU resource utilization and SBE count is not straight forward. In fact, our results indicate that higher core utilization does not necessarily lead to increased SBE occurrences. More fine grained activity counters, such GPU SM level activity, will help us establish such correlation better and could be used for other purposes including code optimization.*

We also investigated if SBE occurrence has sensitivity toward specific user-code. Ideally, this can be investigated if we have access to full information about the binaries that are being run, how the code characteristics change across input types, different implementations of the same algorithm, etc. However, many applications that are run on Titan may be mission critical and application-level information may be sensitive, so it is not feasible to perform a rigorous analysis at the whole system level. But, we have used userID as a proxy for the kind of application they represent in order to perform a first-order analysis. Fig. 20(left) shows that typically users utilizing more GPU core hours tend to experience

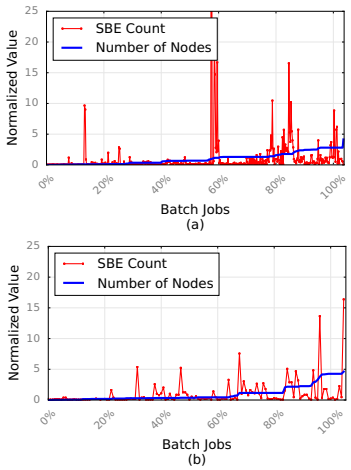

Figure 18: Number of nodes and single bit errors: all jobs (a), and excluding jobs that used any of the 10 GPU cards experiencing the most single bit errors (b).

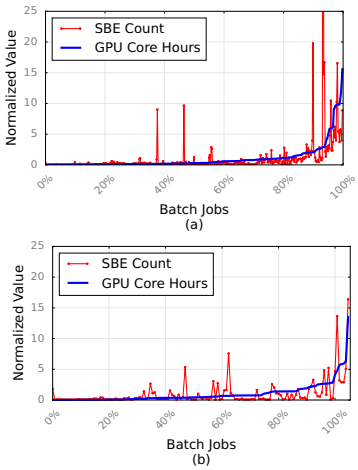

Figure 19: GPU core hours and single bit errors: all jobs (a), and excluding jobs that used any of the 10 GPU cards experiencing the most single bit errors (b).

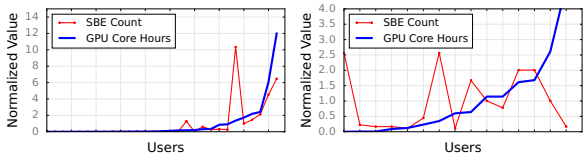

Figure 20: GPU core hours and single bit errors categorized by users.

higher SBE occurrences. Interestingly, the Spearman coefficient is 0.80, higher than the one previously discussed based on the batch job analysis. This indicates the userID may be a better indicator for SBE correlation than solely using GPU core hours. Our correlation coefficient actually improves as the top 10 SBE offender nodes are excluded from the discussion. However, there are notable exceptions as seen in the Fig. 20(right), which is a zoomed version of the first part of the left figure.

Observation 13. *UserID seems to a better proxy for identifying which users/codes may be getting affected by SBE occurrences. This information can be used to identify user codes for better soft-error resilience.*

Next, we discuss the characteristics of GPU workloads on the Titan supercomputer. Fig. 21 shows how the memory consumption, number of nodes vary with GPU core hours and wall clock time. In particular, we observed that it is not necessary that the longer running jobs consume more memory. In fact, we noticed that jobs with the highest maximum and total memory use less than the average GPU core hours (Fig. 21(a)). However, as expected, jobs with longer GPU core hours also tend to use higher number of nodes (Fig. 21(b)). Fig. 21(c) shows that some jobs with smaller node counts may actually be the longest running jobs (as per the wall clock time). Some jobs using larger node counts show mixed behavior: some jobs run for longer wall clock time, while others don't. Fig. 21(d) shows that jobs consuming the maximum amount of memory may actually be running on a relatively smaller node count.

Observation 14. *We observed that GPU jobs running at really large-scale or running for maximum wall clock time do not necessarily stress or consume the maximum amount of memory. Some relatively smaller scale workloads consume the memory resource most. Some smaller scale jobs may even run much longer than larger scale jobs. We also observed that the jobs consuming the maximum amount of memory may be running on a relatively smaller node count.*

5. RELATED WORK

Understanding, characterizing, and mitigating the side-effects of system failures has been an active research area. Several studies have looked into reliability characteristics of large-scale HPC systems [5, 7, 18, 19, 21, 23–25, 31]. These studies have primarily focused on system and application errors on CPU-based, large HPC systems. They have characterized the frequency and impact of different kind of errors. These studies have taken advantage of properties of the failures to improve the efficacy of fault-tolerance mechanisms such as checkpoint-restart [15, 20, 32]. Some of these studies also propose to exploit the correlation among failures to alert/trigger events for failure prediction [11–13, 18]. These techniques may employ machine learning and other techniques to utilize the RAS log information.

Some studies have specifically focused on different system components, such as DRAM [17, 27, 28] and hard-drive disks [26]. Some studies have also focused on evaluating and characterizing emerging architectures such as FPGAs and CMPs [3, 4]. However, large-scale GPU reliability characterization studies are relatively limited [5, 16, 30], primarily because GPU architecture is relatively newer technology in the HPC space.

Recently, there have been many efforts focusing on improving GPU reliability [14]. For example, fault injection experiments have been conducted to track error propagation and estimate the AVF of several GPU kernels [9, 10, 29]. Some other works have focused on evaluation of the robustness of kernels to soft errors [6]. Haque et al. [16] deployed

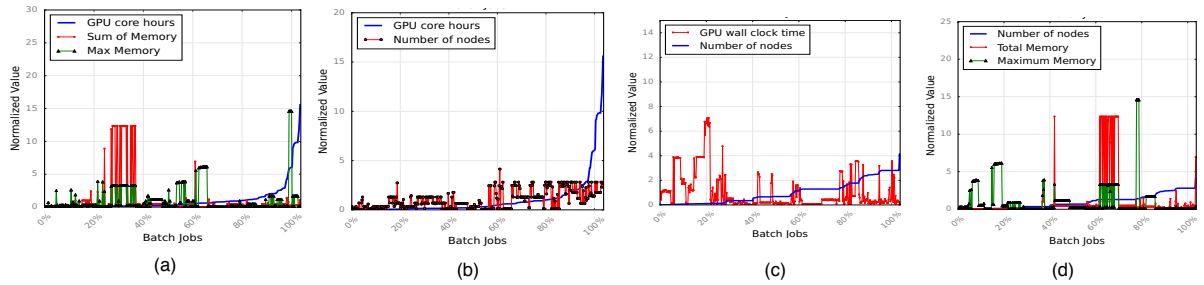

Figure 21: GPU resource consumption characteristics (sorted by GPU core hours (a), (b), and number of nodes (c), (d)).

a software-based GPU soft-error detector on Folding@home distributed platform. This tool and study focused on G80 and GT200 architectures. The study shows that newer generation of GPUs exhibit an order of magnitude lower soft error rate and GPUs also show sensitivity to memory faults in pattern dependent manner. Other recent studies [5, 30] report double bit error rate on the NCSA Blue Water and Titan supercomputer. These studies also show that newer generations of GPUs are more error resilient despite large structure sizes. These conclusions were confirmed with the neutron beam test results. Evidence was provided to show that GPU errors also show temporal locality.

However, none of these studies cover the application and firmware specific errors. Neither do they show the correlation among GPU system failures and the spatial distribution of different kind of errors. In contrast to previous studies, we also show how application specific errors appear over its node allocation and correlation among different types of GPU errors. We also studied the effect of GPU resource utilization and different kind of GPU errors. We provide insights about the GPU workloads and how they may be used toward future HPC system operations. We also discuss the challenges and issues with the current GPU error logging methods and its impact on system operation. As we approach toward exascale, GPUs are likely to be an important part of an exaflop HPC system. Therefore, we believe that early experience with the world's largest GPU-enabled system will help the whole community in improving the understanding the operations of GPUs, their reliability characteristics, and issues involved in error logging. We believe that lessons learned derived from our field data analysis may help in improving the operation efficiency of current and future HPC computing facilities.

6. CONCLUSION

Understanding the operations of GPUs and identifying critical GPU reliability challenges for exascale time-frame are going to become increasingly important. In this paper, we present an in-depth study of different types of GPU errors, their characteristics and impact on HPC workload on the Titan supercomputer. We discuss many important findings related to the spatial characteristics of GPU errors, their frequency and their correlation among themselves. We also discussed how GPU resource utilization may impact different types of errors and some general characteristics of GPU workloads on the Titan supercomputer. Our findings can potentially be helpful in improving the operational efficiency of other HPC centers and improving the accuracy of reliability modeling and simulation in future research studies. Overall, we believe that our experience and analysis is useful in understanding the challenges and issues with GPU error measurement and analysis at large-scale.

Acknowledgment

We thank the reviewers for their feedback that has significantly improved the paper. George Gallarno was supported by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the SULI Program. This work was supported by the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725).

7. REFERENCES

[1] "Gpus speed early science apps," http://www.hpcwire.com/2014/01/06/gpus-speed-early-science-apps/.

[2] "Understanding xid errors," http://docs.nvidia.com/deploy/xid-errors/index.html.

[3] S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga, "Using fpga devices to accelerate biomolecular simulations," *Computer*, vol. 40, no. 3, pp. 66–73, 2007.

[4] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter, "Characterization of scientific workloads on systems with multi-core processors," in *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 2006, pp. 225–236.

[5] C. Di Martino, F. Baccanico, W. Kramer, J. Fullop, Z. Kalbarczyk, and R. Iyer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," *44th international*.

[6] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen, "Matrix multiplication on gpus with on-line fault tolerance," in *Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium on*, May 2011, pp. 311–317.

[7] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how hpc systems fail, DSN," 2013.

[8] M. Ezell, "Understanding the impact of interconnect failures on system operation," in *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[9] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi, "Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications," 2014.

[10] B. Fang, J. Wei, K. Pattabiraman, and M. Ripeanu, "Poster: Evaluating error resiliency of gpgpu applications," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, Nov 2012, pp. 1504–1504.

[11] S. Fu and C. Xu, "Quantifying temporal and spatial correlation of failure events for proactive management," in *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*. IEEE, 2007, pp. 175–184.

[12] A. Gainaru, F. Cappello, J. Fullop, S. Trausan-Matu, and W. Kramer, "Adaptive event prediction strategy with dynamic time window for large-scale hpc systems," in *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. ACM, 2011, p. 4.

[13] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 77.

[14] L. A. B. Gomez, F. Cappello, L. Carro, N. DeBardeleben, B. Fang, S. Gurumurthi, S. Keckler, K. Pattabiraman, R. Rech, and M. S. Reorda, "Gpgpus: How to combine high computational power with high reliability," in *2014 Design Automation and Test in Europe Conference and Exhibition*, Dresden, Germany, 2014.

[15] S. Gupta, D. Tiwari, C. Jantzi, J. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale hpc systems," *International Conference on Dependable Systems and Networks (DSN)*, 2015.

[16] I. S. Haque and V. S. Pande, "Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 691–696.

[17] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 111–122, 2012.

[18] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*. IEEE, 2006, pp. 425–434.

[19] Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta, "Filtering failure logs for a bluegene/l prototype," in *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*. IEEE, 2005, pp. 476–485.

[20] Q. Liu, C. Jung, D. Lee, and D. Tiwari, "Clover: Compiler directed lightweight soft error resilience," in *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2015 CD-ROM*. ACM, 2015, p. 2.

[21] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE, 2007, pp. 575–584.

[22] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Computer graphics forum*, vol. 26, no. 1. Wiley Online Library, 2007, pp. 80–113.

[23] A. Pecchia, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "Improving log-based field failure data analysis of multi-node computing systems," in *Dependable Systems & Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*. IEEE, 2011, pp. 97–108.

[24] R. K. Sahoo, M. S. Squillante, A. Sivasubramaniam, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in *Dependable Systems and Networks, 2004 International Conference on*. IEEE, 2004, pp. 772–781.

[25] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, 2010.

[26] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, 2007, pp. 1–16.

[27] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: a large-scale field study," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1. ACM, 2009, pp. 193–204.

[28] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng shui of supercomputer memory: positional effects in dram and sram faults," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 22.

[29] J. Tan, N. Goswami, T. Li, and X. Fu, "Analyzing soft-error vulnerability on gpgpu microarchitecture," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, Nov 2011, pp. 226–235.

[30] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux *et al.*, "Understanding gpu errors on large-scale hpc systems and the implications for system design and operation," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 331–342.

[31] D. Tiwari, S. Gupta, J. Rogers, and D. Maxwell, "Experience with gpus on the titan supercomputer from a reliability, performance and power perspective," in *Proceedings of Cray User Group Conference (CUG 2015)*, 2015.

[32] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," *International Conference on Dependable Systems and Networks (DSN)*, 2014.