A Methodology for Comparing the Reliability of GPU-Based and CPU-Based HPCs

NEVIN CINI and GULAY YALCIN, Abdullah Gul University, Turkey

Today, GPUs are widely used as coprocessors/accelerators in High-Performance Heterogeneous Computing due to their many advantages. However, many researches emphasize that GPUs are not as reliable as desired yet. Despite the fact that GPUs are more vulnerable to hardware errors than CPUs, the use of GPUs in HPCs is increasing more and more. Moreover, due to native reliability problems of GPUs, combining a great number of GPUs with CPUs can significantly increase HPCs' failure rates. For this reason, analyzing the reliability characteristics of GPU-based HPCs has become a very important issue. Therefore, in this study we evaluate the reliability of GPU-based HPCs. For this purpose, we first examined field data analysis studies for GPU-based and CPU-based HPCs and identified factors that could increase systems failure/error rates. We then compared GPU-based HPCs with CPU-based HPCs in terms of reliability with the help of these factors in order to point out reliability challenges of GPU-based HPCs. Our primary goal is to present a study that can guide the researchers in this field by indicating the current state of GPU-based heterogeneous HPCs and requirements for the future, in terms of reliability. Our second goal is to offer a methodology to compare the reliability of GPU-based HPCs and CPU-based HPCs. To the best of our knowledge, this is the first survey study to compare the reliability of GPU-based and CPU-based HPCs in a systematic manner.

CCS Concepts: • **Computer systems organization → Reliability**;

Additional Key Words and Phrases: System failure, log file analysis, checkpoint/recovery

ACM Reference format:
Nevin Cini and Gulay Yalcin. 2020. A Methodology for Comparing the Reliability of GPU-Based and CPU-Based HPCs. *ACM Comput. Surv.* 53, 1, Article 22 (February 2020), 33 pages.
https://doi.org/10.1145/3372790

1 INTRODUCTION

For today's technology, High-Performance Computing (HPC) is an indispensable instrument. HPCs are supporting research and innovation in many scientific fields, from nuclear sciences to health, by increasing scientific exploration speed and effectiveness significantly. HPCs open the door of countless scientific discoveries, including condensed matter physics, molecular dynamics, and human genome projects. Nevertheless, the demand for a higher-performance supercomputer keeps increasing; thus, the number of cores and nodes of supercomputers grows day by day.

Besides performance, energy consumption is also a main issue in designing HPCs, especially when HPCs are heading to extreme-scale computing. This is because the number of system components is increased due to the performance concern. In order to provide high performance in

Authors' addresses: N. Cini (corresponding author) and G. Yalcin, Abdullah Gul University, Kocasinan, Kayseri, 38080, Turkey; emails: {nevin.cini, gulay.yalcin}@agu.edu.tr.

0360-0300/2020/02-ART22 $15.00
https://doi.org/10.1145/3372790

an energy-efficient manner, *heterogeneous HPCs* were designed and implemented, and several of those heterogeneous systems were among the first 10 supercomputers in the TOP500 list, such as Titan [2] and Piz Daint [1]. Particularly, high parallelism provided by the GPU architecture offers higher performance and finer granularity compared to previously implemented CPU-based HPCs (or *homogeneous HPCs*) [67]. Also, GPUs provide better performance per watt compared to CPUs (see Section 2). Given the power bottleneck in exascale systems and thanks to the advantages of GPU architecture, it is expected that future HPCs will consist of largely GPU nodes [22]. However, as the number of components grows in supercomputers, this trend may cause new problems in terms of reliability [7, 33, 86].

The HPC community has reached a consensus that reliability is a serious issue for large-scale systems [24]. While the number of system components is increased, software and hardware will become more complicated and are likely to experience more errors [99]. The main observation from existing large-scale systems is that exascale systems will be exposed to a variety of faults many times per day [10]. While ensuring the reliability of today's petascale systems is costly, many studies show that obtainment of the reliability of future exascale systems will be much more costly with existing fault tolerance techniques. It is even predicted that this cost will reach the level at which the benefits of the new system can be neutralized. Moreover, the common concern is that checkpoint-restart, which is the key fault tolerance technique, can be useless since the time between two failures will be too short so that the system will experience a new failure before the checkpoint-restart period is complete. This means that large applications running on these systems for several hours cannot be completed successfully [10, 93].

GPUs were initially developed for applications in which reliability was not the main concern, such as multimedia applications in which faults could be tolerated easily [28, 62]. However, the general trend is to use GPUs in critical computations in supercomputers because of the high degree of parallelism they provide [26]. Therefore, they pose a significant risk for reliability. Moreover, HPCs are particularly sensitive to soft errors because of their high degree of complexity and size, and adding a huge number of GPUs may increase the likelihood of a system failure dramatically[1] [11, 34].

We are confident that GPUs improve performance on HPCs; however, we do not know how they affect the total reliability of the system, which is an important deficiency in this area. There have been several studies that investigate the fault tolerance of individual CPUs [44, 83, 87] and GPUs [78, 80, 117]. However, there is not enough study about how the reliability of the system changes when we use two types together and massively. Also, although there have been several studies investigating the reliability of HPCs, only a few of them concentrate on the heterogeneous systems [22, 41, 67, 112–114]. These studies were done by examining the log files of Titan and Blue Waters supercomputers. Titan log records have been examined in five of these studies [41, 67, 112–114] and one of these studies has analyzed the Blue Waters log files [22].

However, these studies mostly focus on the reliability of memory components such as DRAMs and caches, and there is not enough study about the reliability of the instruction execution and control logic of those systems.

In the literature, as the number of studies that evaluate the reliability of heterogeneous systems is insufficient, the number of studies comparing the reliability of homogeneous and heterogeneous systems is quite poor as well. To the best of our knowledge, there is only one study that analyzes homogeneous and heterogeneous HPCs together [32]. In this study, log data were collected from four homogeneous HPCs (Jaguar XT4, Jaguar XT5, Jaguar XT6, Eos XC 30) and one heterogeneous

[1]The terminology used in reliability studies for heterogeneous systems is not standardized, and many refer to failure as the occurrence of incorrect bit values.

HPC (Titan) over a period of about 8 years. This study is, although not sufficient, still important since it is the first and single example of the comparative analysis in which we tried to draw attention to the deficiency in the literature.

Therefore, in this study, our goal is to provide a comparative analysis of homogeneous (CPU-based) and GPU-based heterogeneous HPCs in order to draw a comprehensive road map for the researchers who study to mitigate the reliability issues of heterogeneous HPCs. To this end, we collect the studies in this domain in terms of two main aspects. First, we collect studies that analyze the error characteristics of heterogeneous HPCs and compare them with relevant studies conducted on homogeneous HPCs. By examining these studies, we focus on what needs to be done and the missing research directions to analyze and evaluate the reliability of heterogeneous HPCs. Meanwhile, we also review GPU fault injection (FI) studies in order to understand the error resilience of GPU applications. Second, we collect studies proposing fault tolerance for heterogeneous HPCs in order to mitigate errors. We evaluate if these proposals are enough for future heterogeneous HPC systems.

The rest of this article is organized as follows: Section 2 summarizes the place and historical background of GPU-based heterogeneous HPCs in heterogeneous systems. In Section 3, we explain in detail the purpose and scope of this study. Section 4 evaluates the results of studies that examine the reliability features of homogeneous systems and GPU-based heterogeneous systems in a comparative manner. Section 5 lists the shortcomings of existing GPU-based heterogeneous HPCs and HPCs in general in terms of reliability. Section 6 evaluates the reliability of GPUs via fault injection experiments. Section 7 summarizes the reliability schemes developed for GPU-based heterogeneous systems. Section 8 discusses the hurdles and requirements for increasing the reliability of GPU-based heterogeneous HPCs. Section 9 concludes the article.

2 BACKGROUND: GPU-BASED HETEROGENEOUS HPCS

This section presents a brief historical development of heterogeneous systems, especially GPU-based systems. We also explain where GPU-based systems are located in heterogeneous systems, the strengths and weaknesses of GPU-based systems, and why GPU is a promising technology for HPCs.

2.1 TOP500 List

TOP500 [3] is a list with ranks of the world's fastest 500 supercomputers. This list uses High-Performance Linpack (HPL) as a metric to rank the HPCs. HPL is the maximum peak (number of flops) achieved during the implementation of the Linpack benchmark. The TOP500 lists, which are accepted as the authority for the HPC world, are published twice a year in June and November. Trends in the HPC world can be followed by analyzing these lists. Figure 1, which is created by analysis of TOP500 lists, summarizes the historical development of GPU-based heterogeneous systems. This figure presents the total number of GPU-based HPCs and the total number of heterogeneous HPCs.

The main observation that can be made from the figure is that the number of GPU-based heterogeneous HPCs is increasing among heterogeneous systems and about 90% of heterogeneous systems are GPU-based systems in the latest list (November 2018).

In fact, with the introduction of GPUs in HPCs in the 2000s, a new era has begun. Especially with the development of general programming languages, GPUs have begun to be used extensively in HPCs. Parallel to that, we encountered the first GPU-based HPCs in the TOP500 list in 2008. However, the most remarkable development in this area was in November 2010. That year, a GPU-based heterogeneous HPC, called Tianhe-1A, was able to enter the TOP500 list at the rank of number one. And the previous term, in June 2010, a GPU-based heterogeneous HPC, Nebulae, had

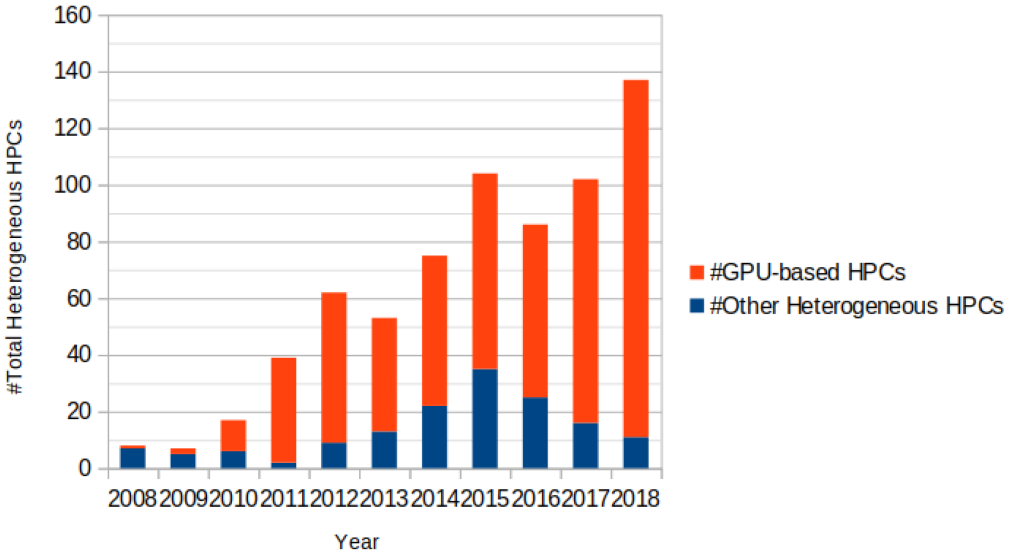

Fig. 1. Advancement of GPU-based HPCs.

already entered the list at the rank of number two. These two achievements show that 2010 is a milestone for GPU-based systems. On all lists published from that date until today, the best GPU-based HPCs have always been among the top five. However, there were heterogeneous systems at the top of this list prior to 2010 as well. For example, while in 2008 a GPU-based system appeared at rank 30th, there was a cell-based heterogeneous HPC on the top of the list. On the other hand, while there are over 120 GPU-based HPCs on the recently published list (November 2018), no cell-based systems are on the list. Cell-based systems seem to have left their place to Phi-based systems. There are 22 Phi-based heterogeneous HPCs on the recent list (excluding systems that use Phi processors as the main processors). However, this is about one-sixth of the number of GPU-based HPCs.

Now we can make comments about today's widespread heterogeneous architecture. The pros and cons of today's heterogeneous systems will shed light for future architects. However, even though today's performance is far behind the performance of an exascale system, we believe that it will only be possible to achieve an exaflop with a heterogeneous system, as the trend in the HPC market is also showing. The most important indication of this is that the world's fastest five supercomputers have heterogeneous architectures.

2.2 Advantages of Heterogeneity

Heterogeneous designs present several benefits compared to the homogeneous systems that we list in this section.

2.2.1 High Performance. While originally GPUs were designed to help CPUs, especially in games that require a lot of graphic processes, they also attracted the attention of other areas with their high performance. Especially in supercomputer applications, the GPU can provide significant performance improvements. The GPU owes this performance boost to its Single Instruction Multiple Data (SIMD) parallel architecture. This architecture is well suited for large problems that can be divided into small independent problems. The SIMD architecture provides fairly high performance because we can divide very large data into very small and simple pieces

and the same operation can be repeated for each of the parts without the need for synchronization. Indeed, many scientific problems are well suited to this specified format. That's why it's not difficult to integrate GPUs into HPCs.

It is thought that the performance increase is directly proportional to the parallelization ratio. However, even if intersynchronization is not necessary, performance may not increase linearly with parallelism. This is because each piece (thread) must have finished its work to get a final result. Even though each thread is of equal size and each process is the same, the processing of data pieces can take different times, so there may be a latency, and finally, the pieces have to wait for each other. This actually means that the performance is determined by the piece that finishes its job last. However, the amount of this latency among the GPU cores (especially new-generation technologies) is less than the CPU cores. Thus, real-time applications are developing. Since GPUs have many threads, even if they are slower than CPU threads, we can divide them into many pieces to solve big problems much more easily, so this naturally improves performance.

2.2.2 Energy Consumption. The energy requirements of today's petascale systems are measured by megawatts. It is estimated that the energy consumption of future systems will go far beyond this. Reducing the energy consumption of exascale systems to a reasonable level is one of the most important issues in this field. GPUs are the only technology that stands out in this area with low energy consumption. An affirmation of this is that all of the top 10 systems listed on the June 2017 Green500 are GPU-based systems. Also, the recent list (June 2018) includes seven GPU-based HPCs (Green500 is a list of low-energy-consuming HPCs). The most powerful HPC of the world (Summit) is also ranked sixth in the June 2018 Green500 list. This indicates that the improvement in GPU technology ignores neither performance increase nor energy efficiency. This balanced development in GPU technologies is vital for exascale systems.

2.3 Challenges of Heterogeneity

Heterogeneous systems have some weaknesses as well as many advantages. Compared to the CPU, the performance advantage of the GPU is obvious for individual use, but in heterogeneous supercomputers, the GPU cannot achieve the required performance increase unless these weaknesses are addressed [61, 124]. There are two main issues that can affect the performance of GPU-based heterogeneous systems that need to be addressed, which are memory hierarchy and reliability.

2.3.1 Memory Hierarchy. Memory is the most important issue that can affect performance for all systems. Memory technology is still far from processor technology. When we consider heterogeneous systems, the case is even worse. Because CPUs and GPUs have different execution models and different memory models optimized for individual use, these models are not suitable for heterogeneous systems. To avoid performance loss, a new optimally balanced memory model is required for using two processor types together.

2.3.2 Reliability. In general, it is quite difficult to keep large-scale systems highly reliable due to shrinking processor technology, the ever-increasing number of components, and, consequently, increased system complexity. However, reliability is crucial for HPCs where many critical computations are performed. At the same time, it directly affects system performance, because when an error occurs, some of the processes or sometimes all are repeated. This means both energy and time loss. Since today GPUs are the most used technology as coprocessors/accelerators in heterogeneous HPCs, GPU reliability is now of at least as great importance as CPU reliability. However, most studies show that GPUs are more susceptible to failures than CPUs [16, 22, 58, 77, 123]. They indicate that the GPU-related errors are not negligible for HPCs.

3 AIMS AND SCOPE

System logs contain valuable information in terms of reliability, especially for large-scale HPC systems. Examining the system logs, despite being rather burdensome, for the last few years for large-scale systems is very critical and has become more attractive for researchers and system designers. This is because, given the size and complexity of HPC systems, model-based methods are not applicable in assessing the reliability of large-scale systems. It is almost impossible to simulate today's ultra-large-scale systems, or it is not practical to understand system failures by means of methods such as fault injection or Architectural Vulnerability Factor(AVF) analysis. With the analysis of actual log data, more accurate results can be obtained than those obtained from these methods. Therefore, for large-scale systems, collecting and analyzing reliability-related log data becomes increasingly important.

Furthermore, a number of critical questions can be answered by analysis of the log data of the systems, such as the main causes of failures, which components fail more, which errors cause more failures, and how failure tolerance mechanisms can diminish these failures. Moreover, by understanding the failure behavior of existing systems, more reliable systems can be designed. In general, failure data analysis can yield three benefits: First, we can obtain the system dependency map for making better job scheduling decisions and resource allocations. Second, this analyzed data can be used to verify the performance of simulators and reliability modeling. Third, log analysis can be used as a guideline for evolving new low-overhead fault tolerance approaches that are critical to the future exascale systems.

There are many studies in the literature that have characterized the failures by analyzing log files. These studies can be categorized under two headings: Studies in the first category (Evaluation Studies) try to evaluate the reliability of the system through the assessment error/failure rate of the system, for instance [7, 16, 17, 22, 25, 41, 49, 57, 71, 88], and the studies in the second category (Improvement Studies) try to improve the reliability of the systems by developing new and more efficient fault tolerance techniques. These studies usually make use of reports provided through log analysis such as [5, 6, 31, 32, 38, 39, 47, 53, 56, 68, 69, 84, 119, 125].

The vast majority of studies in the first category have characterized the failures in homogeneous systems [25, 42, 49, 57, 71, 79, 88]. However, despite urgent need, there are only a few studies examining the logs of HPCs containing GPU cores [22, 41, 67, 112–114].

As we approach exascale HPCs, understanding the failure characteristics of heterogeneous systems is becoming more essential since the reliability wall is one of the main limits of going to the exascale systems. But unfortunately today, we do not have enough information about GPU-related failures within the HPC systems. If the failures caused by GPUs can be analyzed in more detail, these types of failures may be tolerated.

For these reasons, in this study, we are investigating the reliability of GPU-based HPCs. Particularly, we survey the studies that analyze error/failure logs of large-scale systems. Our goal is to evaluate the reliability of GPU-based large-scale systems via comparing them with homogeneous systems. More precisely, we aim to make an effort to understand the similar and different aspects of these two different systems in terms of reliability to be able to evaluate the reliability of GPUs, which is relatively new for large-scale systems. If we can foresee at what rate or how GPU-related errors affect HPCs and how often they cause system failure, we can make more rational conclusions about how much GPUs contribute to the overall performance of the system. Moreover, similar and different aspects of GPU- and CPU-related errors can give us insight into the efficiency of existing error detection and correction methods in heterogeneous systems. This also can enable overcoming shortcomings by development of new techniques and hardware technologies for heterogeneous systems. Comparing these two systems will allow us to see the requirements of future large-scale systems and present a road map for enhancing the reliability of those systems.

Table 1. Summary of the Major Factors That Can Increase the Number of Failures/Errors

Factors	References
Correlations between Failures	[25, 41, 49, 54, 60, 90, 110, 112]
Failures & Errors Distribution	[36, 37, 42, 47, 54, 56, 60, 79, 88, 103, 113]
Locality Effect	[7, 25, 41, 42, 67, 69, 90, 101, 103, 112, 113]
Effect of Resource Utilization and Workload	[25, 60, 67, 69, 90, 95, 112]
Temperature	[7, 25, 67–69, 90, 113]
Transient and Permanent Faults	[17, 22, 49, 95, 96, 102, 103]
Effect of Altitude	[17, 101, 103]
DRAM Vendors	[60, 101, 103]
Location Effects	[103, 112]
Error Modes	[8, 22, 34, 96, 101–103, 113]
Size	[15, 16, 24, 34, 88, 110]
Age	[54, 60, 90]
Memory Error Rate	[7, 22, 68, 69, 95, 96, 102]
Failure Root Causes	[22, 57, 88, 113]
DIMM Capacity/Chip Density	[60, 90]
Newer-Generation Technologies	[10, 40, 60, 66, 90, 91, 99, 122]

In order to compare the reliability of these two systems, first, we researched the studies that analyze the log data of homogeneous and heterogeneous systems from the reliability window. We have summarized these studies in Table 5 in Section 9. Table 5 includes the category of studies (Evaluation or Improvement Studies of Homogeneous or Heterogeneous HPCs). Whether to evaluate the system reliability or to improve it, we have tried to add all the studies that analyze the system logs and present the results of their analyses. Some improvement studies include analysis results.

While we reviewed the studies that analyze the log data of homogeneous and heterogeneous systems, we have also revealed some factors affecting the reliability of the systems.

After analyzing the studies in detail and determining the major factors that can increase the number of failures/errors, we compared the heterogeneous systems and homogeneous systems via these factors (see Section 4). We also summarize these factors and related references in Table 1. In the next section (Section 5), we evaluate the results of our comparative analysis and discuss important issues concerning GPU-based heterogeneous systems. We list deficiencies and our suggestions.

Although this research covers studies analyzing log data, since the number of studies analyzing GPU-based heterogeneous HPCs is insufficient, which we are trying to draw attention to by expanding the scope of this research slightly, in another section, we present the significant findings of some studies that do not perform log analysis (Section 6). In that section, we review the studies that performed fault injection experiments for GPUs. We summarize these studies in Table 4. The results of the detailed analysis of these studies contributed significantly to our understanding of the reliability of GPUs and gave insight into possible reliability issues with the integration of GPUs into HPCs.

In Section 7, we review in detail the developed checkpoint models for GPUs, and in Section 8, we discuss the challenges of this subject.

With this work we also provide a methodology to compare the reliability of GPU-based HPCs and CPU-based HPCs. To the best of our knowledge, this is the first study to compare the reliability of GPU-based and CPU-based HPCs in a systematic manner.

4 COMPARATIVE ANALYSIS OF HOMOGENEOUS HPCS WITH GPU-BASED HETEROGENEOUS HPCS

In this section, we compare the reliability of homogeneous and GPU-based heterogeneous systems in several aspects. Examining similar and different aspects of systems will provide guidance for future generations of supercomputers. Comparative analyses, especially for GPU-based HPCs, are crucial in order to design more robust systems. However, both the poor number of analyses made and the differences in terminologies and methodologies used do not allow one to reach definite conclusions.[2] So we focus on only finding some clues about the reliability vulnerabilities of existing systems and try to uncover requirements for improving system reliability.

A summary of all the studies we have analyzed in this section can be found in Table 5 in Section 9 at the end of this article.

To be able to perform a comparative analysis in a systematic manner, in this section, we describe the causes of failures or the major factors that can increase the number of failures/errors. We explain these factors in detail in light of the findings of failure/error log analysis studies for both homogeneous and GPU-based heterogeneous HPCs in a comparative manner. We summarize all these factors and their related references in Table 1.

Some of the log analysis studies in the literature have been done to understand the overall reliability of the system [25, 42, 57, 88], while others have been done to assess the reliability of only memory components [54, 60, 95, 96, 101, 102]. We considered this difference during our analysis and evaluated the findings among themselves according to the type of study. However, to avoid repetition, we evaluated all studies in the same section.

4.1 Correlations between Failures

Studies show that some failures trigger some other failures. These types of failures are called parent-child events or follow-up failures. Correlations may occur among the nodes or failures, and for some systems correlations are observed at levels that can be considered [110]. We can predict the failures more accurately if these correlations are not artificial and correctly defined. For both homogeneous and heterogeneous HPCs, strong correlations have been found among failures [25, 41, 112]. However, homogeneous and heterogeneous systems have a wide variety of parent-child failure types, and the lack of similar correlations between different systems indicates that more research is needed in this regard. For example, in a heterogeneous system, there are two types of errors that usually follow DBE on GPUs: ECC page retirement and driver error. In the same system, it is observed that the most common failure types are "GPU of the bus, Kernel Panic and SXM Power-Off. And these failures have stronger correlations with other failure types"; however, Machine Check Exceptions (MCEs) are independent from other failures [41, 112]. On the other hand, in a study with 10 different homogeneous HPCs, it was found that the same types of failures are more likely to follow each other [25]. However, the occurrence of a real correlation between these types of failures depends on whether the failures are due to hard error or soft error [49].

Some studies [54, 60, 90] investigating memory errors in homogeneous systems have also found correlations between errors. These studies particularly focused on correctable and uncorrectable memory errors. However, the results of the studies are quite different. While a recent study [54] could not find any correlation between correctable errors (CEs) and uncorrectable errors (UEs), another study [90] reported that there is a tight correlation. Besides, some temporal and spatial correlations were observed among CEs [90].

[2]Since the terminologies used in the studies differ so much, we have tried to minimize this difference as much as possible in order to compare and evaluate the findings. However, we should say that different terminologies used in the literature lead to a great deal of confusion. We think that there should be a standard.

4.2 Effect of Resource Utilization and Workload

El-Sayed and Schroeder [25] examine the job records of two homogeneous HPCs with similar hardware architecture and similar workloads to investigate the effect of usage on node reliability. In these two systems, they did not observe a strong relation between usage and failures for any node, except node 0, which is the most used node. On the other hand, they observed that the application characteristics changed the nodes' failure behavior.

While another study [95] found no relationship between errors and memory usage, they reached different conclusions for CPU usage in different time-level analyses. A recent study [60] specified that the intrinsic factor affecting the memory error rate is workload, not memory or CPU utilization. On the other hand, the study in [90] states that utilization is a more prominent factor for memory corrected errors compared to temperature.

For heterogeneous systems, a comprehensive analysis has not been done. There are only analyses related to single-bit errors (SBEs) [67, 112]. Based on this research, there is a strong correlation between SBEs and applications, but there is no correlation between SBEs and GPU resource utilization. Another study found that for applications with higher GPU memory utilization, their SBEs rates are also high [69]. However, the number of analyses in this context is not sufficient to reach a conclusion that there is no relation between utilization and failures.

4.3 Failures & Errors Distribution

Many studies investigating the statistical distribution of failures in homogeneous systems have found that "the time between failures is better fit [to] a Weibull or Gama distribution than Exponential distribution" [36, 37, 42, 47, 56, 79, 88].

Studies that analyze memory errors also do not include different results. The study in [103] emphasizes that both SRAM and DRAM "transient fault interarrival times" do not follow an exponential distribution, which is the general assumption in many simulators; instead, they follow the Weibull distribution. Another study [54] points out that while SRAM detectable uncorrectable errors (DUEs) follow Weibull distribution and DRAM DUEs follow Weibull or gamma distributions, both have the same fit. A different study suggests that for memory errors, "Pareto distribution with decreasing hazard rate" is the best fit [60].

The research of heterogeneous systems also coincides with previous results. In particular, it is emphasized that GPU-based failures have Weibull distribution [113]. However, we think that more specific analyses are needed in this regard. More accurate reliability models can be designed by knowing the distributions of the root causes of system failures, such as software-based failures or DBE-originated failures.

4.4 Locality Effect

It has been found that some nodes are more susceptible to failures for both heterogeneous and homogeneous systems; however, the reason for this tendency could not be explained satisfactorily [7, 25, 41, 42]. Applications running on nodes, nodes' position in systems, temperature, and altitude are likely to be the main reason for the explained behavior. In addition, correlations between failures may also be another reason, which means that once a failure has occurred on a node, the likelihood that the new failures will happen at the same node is quite higher than for a node that has never failed [25]. Moreover, spatial locality among failures is observed not only between nodes but also at different levels of systems [90, 101]. In the same way, strong temporal locality was observed between the failures of both heterogeneous and homogeneous systems. Temporal locality helps discover cause-and-effect relationships of failures. In a study, Tiwari et al. pointed out that there was temporal locality also among GPU-related failures, and in this way

they investigated the causes of failures [113]. In addition, spatial and temporal locality also can be beneficial for job scheduling; system administers can avoid scheduling critical works on more error-prone nodes/days.

In addition, spatial and temporal correlations of single-bit errors and double-bit errors were investigated specifically for heterogeneous systems, and strong spatial and temporal correlations were found among single-bit errors, but neither spatial nor temporal correlations were found for double-bit errors [67, 69, 112]. However, in general, we have not encountered a study investigating the correlation of all memory errors in heterogeneous systems. Two analyses for homogeneous systems have reached different results. A recent study [7] has found that there is "a very strong spatial correlation between memory errors (i.e., 99.9% of errors occurring in less than 1% of the nodes)." The same study also points out the temporal correlation among the errors. They observed that the system was encountered with one to two memory errors on a normal day, but on some days they observed that many errors follow each other. However, another study found that the probability of a node experiencing one or more DRAM faults exposed to new DRAM faults is the same as the probability of a node where no DRAM faults are seen [103]. This indicates that DRAM faults do not have spatial correlation. It seems we need more research for both homogeneous and heterogeneous systems.

4.5 Temperature

According to the study in [90] investigating the behavior of memory errors, the effect of a certain range of temperatures on DIMM errors is very limited. Furthermore, temperature variations in a certain range do not seem to have a relationship with failures [25].

However, excessively high temperatures are known to cause hardware failures. Normally we expect higher temperatures to increase memory errors, but contrary to expectations, many failures occur in the normal temperature range (30°C to 40°C), and only a few errors occur at high temperatures (over 60°C) [7]. In particular, a study reports that all of the multibit errors happen in the normal temperature range [7]. However, this does not mean there is no correlation between memory errors and temperature.

For heterogeneous systems, we have not found analyses of how temperature affects system failures. But there are a few analyses for specific types of errors, and these analyses suggest that there may be correlations between temperature and GPU soft errors; however, note that this correlation does not exist among all nodes [67]. Specifically, studies in [68, 69] found that there may be a correlation among temperature, power consumption, and GPU single-bit errors; however, they have noted that it is not easy to define and exploit this correlation. Another study [113] suggests there may be correlations between DBEs and temperature.

4.6 Transient and Permanent Faults

Much of the existing literature observed that permanent faults are the main cause of many DRAM faults in homogeneous systems [17, 49, 95, 102, 103]. Although the rate of transient faults changes very little, permanent faults decrease over time, as in the infant mortality phase of the bathtub curve [96, 102, 103]. Only in the early years of the system lifetime were DRAM faults replaced from permanent to transient faults [96].

However, for SRAMs, transient faults increase over time, permanent faults decrease, and SRAM faults often consist of transient faults seen in L2 and L3 caches [103]. One study indicates that more than 99% of faults in L3 are transient faults [96].

For heterogeneous systems, both permanent and transient faults are expected to be higher because the devices size shrinks [22], but in the literature, we have not found an analysis conducted on permanent and transient faults related to the size of the technology used in HPCs.

Table 2. SBE Ratios of Four Different HPCs

HPC	Single-Bit Errors
Jaguar (DDR2)	49.7%
Cielo (DDR3)	67.7%
Hopper (DDR3)	78.9%
BlueWaters (DDR3+DDR5)	70.01%

However, the study in [43] noted that transient faults occurred in GPU computational units as well as GPU memory.

4.7 Effect of Altitude

The study in [103] compares the error rates of two systems, where the other features are mostly similar and the altitude parameter is about 9× that of the other. They observe at least a "2.3× increase in SRAM fault rate" [17, 103]. The main reason for the increase in SRAM faults is high-energy neutrons from cosmic radiation. Another study [101] investigates the effect of altitude on DRAM devices in the same way and achieves similar results. In particular, increases are observed in "single-bit, single-column, and single-bank transient fault rates." However, although studies achieve the result that while altitude increases, memory errors increase, they emphasize that the effect of altitude on system reliability is highly dependent on error protection mechanisms and DRAM vendor selection. We have not found any analyses about the effect of altitude on memory errors for heterogeneous systems.

4.8 DRAM Vendors

Indeed, the choice of DRAM vendors affects system reliability directly. Different vendor types have different transient and permanent fault rates, as well as different fault modes. Some experience single-bit faults more; others have more multiple-bank or multiple-rank faults. The difference between the total number of DRAM faults is also dramatic. This difference is about 4× for some vendors [101, 103]. Another study [60] observed that the failure rates among some DIMM vendors can rise up to 2×.

4.9 Location Effects

Studies have been conducted on homogeneous systems and observed that a fault within a DRAM device has an uneven distribution, meaning that a DRAM fault may happen "in any region of a DRAM device" [103]. Similarly, there is no meaningful correlation between DRAM fault rate and DRAM position within a data center. However, there is a significant correlation between rack position and SRAM fault rates. Although it is not clear but may be due to temperature, at the top of the rack, SRAM has 20% higher fault rates than the bottom of the rack [103]. We have not encountered an analysis of heterogeneous systems, but we know that DBEs are seen more in the upper cages than in the lower cages of the cabinet [112].

4.10 Error Modes

Analyses show that most DRAM and SRAM errors are SBE [96, 103]. Table 2 shows the SBE rates of three different homogeneous systems and one heterogeneous system. Most of the systems have about 70% SBE rates, but the average multibit error (MBE) rate is about 30%, which shows that MBEs are at a level that can no longer be ignored [22, 101–103]. Another study has reported that while a heterogeneous HPC (which only has 308 nodes) experienced 1.32 double-bit errors per day

during a 4-month time period, a homogeneous system (which has 9,000 nodes), on the other hand, "experienced one uncorrectable DRAM error every 11 days" [34]. Although these results are not directly comparable, nevertheless, we can say that this is alarming especially for heterogeneous systems where GPU memory is protected by SECDED ECC. Because "SECDED ECC can correct single-bit errors, but can not correct double-bit errors," it can only detect them, so when a DBE is detected in the system, even if this error is a benign error, it usually causes an application crash [113]. There are still vulnerable areas in GPGPU that can lead to silent data corruption. Moreover, some large-scale applications, such as molecular dynamics, prefer to switch off ECC because it reduces performance [8].

Particularly, some studies [68, 69] have investigated the characteristics of GPU SBEs and have found some remarkable correlations. However, these studies have noted that their system does not store SBEs. Therefore, the NVIDIA-smi utility, which records both SBEs and DBEs, is used to collect SBEs. However, they indicate that the number of DBEs recorded by NVIDIA-smi does not match the number of DBEs in the log files. Thus, the findings of these studies should be taken into account by considering the explained limitation.

It is clear that more detailed analyses are needed for memory errors. The answers to questions such as which nodes are responsible for more MBEs, how many of the errors cannot be corrected, and how many errors lead to silent errors are critical for system reliability. Particularly for heterogeneous systems, the studies in this area are very insufficient due to the lack of tools to monitor and analyze more fine-grained activities.

4.11 Size

For homogeneous systems, we know that for systems with the same hardware type, while the size increases, the failure rate increases. However, this increase is not faster than linear [88]. The increase in the failure rate can be expressed as a function of the number of nodes in the system. This is extremely important for HPCs since their number of components is rapidly increasing. The failure rate of a single component may not increase in the future; nevertheless, the overall system reliability will decrease dramatically since system size increases extremely [15].

For heterogeneous systems, even though we do not have enough data to verify, it is highly possible that while the system size increases, the failure rate increases faster than linear. All works in this area emphasize this likelihood [16, 24, 34, 86].

Therefore, for large-scale systems, it will be more important to detect errors as early as possible and also understand the relationship in order to prevent spreading [110].

4.12 Age

Age is also one of the factors to consider when evaluating the reliability of system components, because with aging, the number of errors may increase or error behaviors may entirely change. As such, there are many studies that evaluate the age factor in terms of system reliability. However, the results of the studies vary. For instance, the study in [54] monitored a homogeneous system over 5 years and observed no significant "age-dependent trend" in memory error rate. Specifically, they analyzed DRAM and SRAM errors and found that variation in detectable uncorrectable errors intervals is trivial over these years. This is because memory operational lifetime may be longer than 5 years. Another study indicates that correctable and uncorrectable error rates are highly influenced by age. However, this study also emphasizes that impact severity depends on some factors such as "DIMM technology, platform and vendor" [90]. Another study [60] has analyzed chip density and age comparatively. They noticed that among systems with the same age but with a different number of cores, as the number of cores increases, the number of failures increases; on the other hand, when we keep the number of cores constant, the failure rate increases with aging.

A different study [18] has investigated how permanent and intermittent faults affected GPUs by accelerating processor aging via excessive temperature. They elevated the temperature to 170°C so that intermittent faults could occur and then observed that intermittent faults were lost when they lowered the temperature to 150°C.

4.13 Memory Error Rate

Most studies are unable to reveal the absolute number of errors of systems due to confidentiality concerns [95, 96]. Nevertheless, few studies give error rates.

In this area, one of the few studies on heterogeneous HPCs reported that about 67% of machine check exceptions are memory errors [22]. Memory errors are also a serious problem for homogeneous HPCs. While one study [102] has calculated that memory error rate is 0.066 FIT/Mbit (i.e., about a fault every 6 hours), another study [7] has collected 55,000 independent memory errors in just a year's period. So the system experiences a memory error about every 10 minutes. These studies show that memory error rates are above acceptable levels for HPCs.

4.14 Failure Root Causes

System failures in heterogeneous HPCs are usually caused by hardware, heartbeat, and software errors. However, software-related failures have the highest total node repair time. Environmental and hardware-related faults are number two and number three, respectively [22]. This study also reports that only 2.9% of failures are unknown. This shows that the system's failure diagnosis mechanism works well. Another study has listed the root cause of GPU-related failures as "Off-the bus, ECC page retirement and DBE on GPU" [113]. A different study [11] has reported common hardware failures related to GPUs: GPU has fallen off the bus, GPU killed by applications, slow GPUs. It has stated that it is quite difficult to diagnose the root causes of these failures.

On the other hand, studies for homogeneous HPCs have different results. A study [57] reports that the software halts cause more outages, but the MTTR value is very low, while the hardware outages have a longer repair time. This is due to hardware-related problems needing parts replaced and tests performed. According to another study [88], the main sources of system failures are hardware errors; the second responsible is software errors. Likewise, hardware errors require the longest repair time, and software errors follow hardware errors. Interestingly, for most of the systems analyzed in this study, the rate of unknown failures is fairly high (average 20%) and their total repair times are less than 5%. This result suggests that most of the unknown errors are soft errors that can be corrected by reboot.

4.15 DIMM Capacity/Chip Density

Studies [60, 90] in the literature state that increasing the DIMM capacity affects the error rate, but these studies have not been able to observe a consistent correlation.

On the other hand, studies have reached different results for chip density. One study [90] does not find a consistent relationship between the error rate and chip density, while another study [60] points to a strong correlation.

4.16 Newer-Generation Technologies

It is foreseen that advances in memory technology will increase memory errors on next-generation DIMMs [65, 66]. However, a study [90] examining memory errors of different systems observed that the correctable memory error rates of newer-generation systems were lower compared to older-generation systems. On the other hand, another study [60] emphasizes that DRAM cell reliability increases with new-generation technologies; however, this increase is not sufficient compared with the rapid increase in chip density. This means that total memory reliability is reduced.

Another study [40] comparing five HPCs in terms of reliability characteristics investigates whether newer HPCs with developing technologies are less reliable, as is indicated in several previous studies [10, 91, 99, 122]. The study specifically examines how HPC systems change failure characteristics during the stable operational period. They quantified scale-normalized MTBF to compare the HPCs of each other and found that newer HPCs have higher scale-normalized MTBFs. However, they underlined that comparing system reliability with merely MTBF will mislead and therefore they use scale-normalized MTBF [40]. But this metric is also insufficient to compare because it only takes into account node numbers and it assumes the other features of systems (e.g., core counts, period of the data collection, usage level of the system, etc.) are equal.

4.17 Why GPUs Are Less Reliable

So far, we have tried to give examples of field studies regarding that heterogeneous systems are less reliable than homogeneous systems. Now, based on the literature, we want to summarize the reasons for this.

Indeed, the reasons for making GPUs more error prone are related to their three peculiarities as follows:

Massive Parallelism: Heterogeneous systems allow more parallel processes to improve performance. However, this (1) raises the complexity of systems, which increases the tendency to design and software faults, and (2) raises the dependencies that require more I/O operations with unreliable communication units. An application running on a GPU can consist of millions of threads, but these threads usually run independently of each other, and any error that occurs in a thread will not affect the others [52, 86]. However, an application that executes together on GPU nodes and CPU nodes may be very vulnerable; an error in any node can spread very quickly if not detected in time. Also, an error in the shared resources of GPUs, such as a scheduler, dispatcher, or shared memory, can affect all running parallel threads, and this can cause the error to spread to one or more GPU nodes [34, 82]. A failure in such a case is quite costly compared to a homogeneous system. Further, many experimental studies show that memory errors are particularly prone to disturb multiple threads or multiple thread blocks [81, 113]. Therefore, as the system size increases, error detection and correction mechanisms will be of great importance.

High Density: Since a GPU combines many execution units, the temperature of the GPU will be quite high during the execution [120]. At the same time, a GPU can perform about 50x more operations than a CPU. Potentially, GPUs can include more than 3,000 execution units (EUs) working in parallel. Conversely, for instance, a 12-core CPU contains about 72 EUs. However, EUs are the main devices that cause the system to overheat. Therefore, it is predicted that the number of failures caused by overheating of GPUs will be at least 10x higher than CPUs [51]. As the number of EUs increases, heating increases, and overheating is one of the most important causes of soft errors. In addition, as the density of the chips increases, the performance increases and the heating also increases, and soft errors are inevitable for GPUs with high-density chips due to overheating [63].

High Utilization: As shown in Table 3, in heterogeneous systems, the number of GPUs is on the rise. The number of GPU cores already exceeds half of the total number of cores in the system, and usage of a huge number of cores makes the system more failure prone [34]. Moreover, it is foreseen that the number of applications utilizing GPUs will increase in the future, and an application will utilize more GPUs [51]. This high utilization will keep GPU components busier and more bits in those systems will be required for the architecturally correct execution. Thus, occurrence of a faulty bit in GPU systems will lead to an error with higher probability, which reduces the reliability (or Mean Time to Failure (MTTF)) of the system [64].

Table 3. Advancement of Heterogeneous HPC

Year & HPC	Total GPU Node	SM Unit (Each GPU)	CUDA Cores	Total CUDA Cores	Total SM Units	Total Cores	GPU%	CPU%
2010 & Nebula	4,640	14	448	2,078,720	64,960	120,640	54	46
2010 & Tianhe-1A	7,168	14	448	3,211,264	100,352	186,368	54	46
2012 & Titan	18,688	14	2,688	50,233,344	261,632	560,640	47	53
2017 & Piz Diant	5,320	56	3,584	19,066,880	297,920	361,760	82	18

5 REQUIREMENTS TO IMPROVE THE RELIABILITY OF HETEROGENEOUS SYSTEMS

This section includes the open reliability concerns of GPU-based heterogeneous HPCs, which require further analyses and research, as well as our recommendations on these topics.

We can list the requirements as follows:

- Reliability of a heterogeneous system depends on many factors since it is more complex and has more dependencies. Components of the system should be evaluated together in order to achieve more accurate results. Therefore, more specific and deep analysis should be done. For example, it is important not only to investigate the total failure distribution of the system but also to draw a detailed error map of the system including the distribution of all the faults that cause the failures, individually. Therefore, we need tools to monitor more fine-grained activities of GPUs.
- To make the results of the analysis meaningful enough, data should be collected over a sufficiently long period of time from multiple systems with different component counts and properties. However, only few studies, such as [88, 89, 118], have been done in this way and only for homogeneous systems. In addition, statistical correlations among events obtained only by analysis of log files do not necessarily indicate a cause-and-effect relationship [104] because many analysis works are done with data collected in a very short time period. However, it is necessary to evaluate the previous events of all components related to this event together in order to evaluate any event correctly. System administrators can generally make consistent comments on all context information of events, but the comments of researchers with log information only for a certain time period will not cover the overall system and may mislead.
- Most of the studies have analyzed the faults of a particular component, such as memory. However, studies show that some important GPU structures such as "streaming processors, warp scheduler are seriously unreliable" [109]. Moreover, GPUs have a significant amount of vulnerable execution logics and functional and control units [73]. Although for complete GPU protection there are power and space constraints, we need to take into account all GPU structures together in order to improve the reliability of GPU-based systems [73, 109].
- The analyses of error logs need to be performed with a certain standard in a systematic way in order to be able to compare the reliability of the systems. Differences in methodologies used in field data analysis studies may lead to misunderstandings. In particular, unity should be obtained in the definitions of the terms "fault, error, and failure" used in analyses. It should also be decided which of these would be more suitable to be used in comparisons (i.e., in number of faults, errors, or failures which of these are more accurate to use).
- Another crucial issue is constructing a reliability model for assessing the reliability of heterogeneous systems. There are no metrics or analytical models to compare reliability for heterogeneous systems. First, we need metrics specific to these systems that we can

measure at an acceptable level of reliability in these systems. Metrics that can be obtained by analyzing many aspects of all the factors that may affect system reliability can be used in analytical modeling.

- Calculating values such as MTTF using only logs can be inaccurate and misleading, because the error logs are gathered in many studies when the protection mechanisms are in operation. Especially in the case of memory errors, the system does not record all faults in the logs because many faults are corrected and we do not know the actual rate of these unsaved faults and their dependencies to other errors. Moreover, important information about the characteristics of silent errors can be accessed in an unprotected system. However, in the literature, we encountered only one study that analyzed the data of an unprotected system [7].
- There is only one study that has analyzed the logs of homogeneous and heterogeneous systems together in the literature [40]. However, this study does not provide a comprehensive analysis. A detailed review describing the similarities and differences of failures and their relationships to each other is not included in the literature. However, such a study may be a guide for analytical modeling. Many log analysis studies can provide early detection or prediction if evaluated together [55]. Moreover, if we can know the differences in reliability behavior between GPUs and CPUs in HPCs, we can more easily decide whether the reliability metrics and fault tolerance mechanisms developed for homogeneous systems are sufficient for heterogeneous systems as well. This study is trying to provide a road map to the stakeholders in order to satisfy these requirements.
- Error logs can also be used to develop simulators and benchmarks that can perform more realistic analyses. Especially for heterogeneous systems, we can say that these areas are still in the infant stage. However, when considering the limitations of access to the log data of heterogeneous HPCs (i.e., error logs, system event logs, data from environmental sensors), the importance of simulators is increasing. Also, heterogeneous HPCs require special benchmarks due to differences in architectures and programming models. In many GPU studies such as [97] (not just in terms of reliability), different benchmarks are used. However, since there is no standard benchmark, we cannot compare their suggestions in this way [98].
- The research shows that there are serious reliability differences between the nodes [25, 42, 79]. It is also significant to measure the reliability of individual nodes as well as to measure the total reliability of the system. This will increase performance, especially by assigning critical jobs to more reliable nodes. Especially in heterogeneous systems, reliability differences among nodes are expected to be much higher. This means that node-specific models should be developed with metrics that can measure the reliability of the nodes. Such a model can provide dynamic scheduling to the extent that failures can be predicted and so this minimizes losses.
- The memory of GPU-based heterogeneous systems seems to have serious reliability problems [22]. As GPU memories increase, it is estimated that these problems will grow more and so threaten the future of heterogeneous systems. Therefore, more effective protection methods should be developed for GPU memories.
- An error in the system is not always dangerous. Some errors may occur in parts of the application that will not affect the result. Such faults are called "benign errors." When a benign error occurs in the system, if the application can continue to normal operation, it can produce the desired, correct result. However, in today's HPCs, there is no solution to distinguish benign errors from others, so detection of benign errors can cause unnecessary

interruptions. For this reason, it is necessary to develop solutions to determine the number of such failures and to distinguish benign errors from others.

- Considering all correlations while building a reliability model is also critical for the accuracy of the model, but this is not possible in practice. Often when a model is constructed, it is assumed that the failures are independent of each other and that the components are identical. That is, the probability of failure of a component is the same as any other component. Under these assumptions, a rough, approximate model can be generated [50]. However, as the size and complexity increase, particularly in heterogeneous HPCs, the generated model gets away from the real values. Because, in reality, the failures are not independent but are tightly bound together, components are not the same and do not have the same failure probability, whereas some nodes are more failure prone. Therefore, it is not easy to model failures by using standard distribution models [118].

5.1 Log Analysis Challenges

Although HPC log files allow us to monitor the system's health and evaluate it in many ways, especially when we consider the rapid increase in HPCs' scale and complexity, analysis of system logs is a real challenge for many reasons. We can list these difficulties as follows:

- **Big data:** As the size of HPC systems increases, the size of log records increases and analyzing log files becomes a big data issue [48]. For example, for Blue Waters, approximately 3.7TB-sized system logs have been analyzed [22]. Automatically analyzing a file at this size is a real challenge; it is almost impossible to manually analyze. It needs to develop custom tools using some intelligent techniques [9].
- **Unstructured file formats:** HPC log files are naturally unstructured. Systems produce various types of log files since they collect data from different monitoring tools and a wide variety of hardware and software sensors, such as memory faults and processor usage [72]. The format of the log files may vary depending on the system or the log database. For example, the meanings and orders of the fields or separators can be different [105]. Also, many log files are not human readable since they consist of some codes, numeric values, or cryptic text.
- **Redundant data:** One problem with the contents of log files is that "log files contain much redundant data and information of varying importance" [9, 75] because HPC log data has strong spatial and temporal correlations [21]. For example, some failures may be periodically reported from the same place in the system due to the periodic recording mechanism. Also, many failures can occur with nearby timestamps in different places. The reason for this is probably a failure (such as power failure) that could lead to many other failures in different places in the system [20]. However, this causes mistakes, especially when we try to diagnose root causes of failures. Therefore, it is necessary to eliminate redundant information using some filtering techniques before analysis [107].
- **Lack of scalable and flexible tools:** Some scalable methods for analyzing log data are needed. This will require scalable and highly available database technologies that store the huge data in a flexible format and provide large-scale analytics with low latency, even in real time [74].
- **Cooperation:** Analysis of large data consists of interdisciplinary research that requires experts from different fields to collaborate. This requires scientists from different fields to work together to exchange ideas. However, in order to be able to interpret the analyses in such a study, some basic issues may need to be known by all scientists who conduct the research [13].

- **Data confidentiality:** The biggest obstacle in front of log-file analysis is the limited access to log files. Unfortunately, many large-scale HPC systems' log files are not publicly available due to the confidentiality of this data Therefore, there are few studies that perform log analysis on large-scale HPC systems.

6 EVALUATING THE RELIABILITY OF GPUS VIA FAULT INJECTION

Although the studies we focus on in this work are regarding field data analysis, as a result of our literature review, we have seen that the number of such studies is very limited especially for systems containing GPUs. One of the many reasons for this may be that it is arduous and expensive to perform such an analysis, especially for large-scale systems. However, the analysis of real data is more confident and allows us to evaluate the system in actual cases. On the other hand, field data analysis studies do not contain any information about silent error (SDC) rates. And without SDC rates, we cannot fully evaluate the impact of soft errors on applications. Furthermore, since field data analysis studies do not cover SDC rates, the soft error rates presented in these studies are below their real amounts [86].

Therefore, in this section, we will evaluate the findings of studies investigating the reliability of GPUs through fault injection experiments. Our goal is to understand the real impact of soft errors on GPU applications.

FI is a simple but useful method by which we can analyze the faults that cause application failures (i.e., incorrect output, hang, or crash). Fault injection experiments are performed by intentionally flipping a randomly selected bit over, from a running application. If an injected fault causes an incorrect output without any error message or indication, we count this error as a silent error.

In the literature, there are few fault injection studies for GPUs. These have performed FI experiments by using different methodologies at different levels. We have summarized the fault injection studies in Table 4. For each study, the table contains the major findings of the studies, GPU card models, fault injection locations, and applications used during the fault injection experiments. When analyzing fault injection works, we take heed of findings rather than the methods used. We specifically focus on SDC rates of GPU structures, the fault propagation rates in GPU structures, and application failure rates (hang, crash, or SDC). We present similar studies together.

6.1 Fault Injection Challenges in the Context of GPUs

The significant outcomes reached with the analysis of fault injection studies can be listed as follows:

Coverage Problem: In general, this is one of the most important problems of FI studies. In order to achieve statistically acceptable results, it is necessary to inject a sufficient amount of fault. Particularly in general-purpose GPU applications, fault-injected sites of applications are important because they should be fully representative of the application and fault modes. They should also cover all instruction types or execution paths that the application contains.

A general-purpose GPU application can contain thousands of threads since it provides extreme parallelism. Since FI is a very time-consuming process, it is not possible to inject faults into all threads. That's why we have to choose where to inject the faults. However, when making this selection, we should make sure that we select samples from different parts of the application that will cover all application features and elements, because this will affect the accuracy of the results.

For this problem, two different solutions have been proposed in the literature: first, with the help of some application information, statistically selecting fault injection fields from the application [45, 46], and second, simply, grouping threads according to the number of instructions

Table 4. Summary of Fault Injection Works

Ref.	GPU Card	Benchmarks	Injected Locations	Major Findings
[45] [46] [106]	Tesla K20 and K10 with CUDA 6.5 and 7.0	16 applications from Rodinia benchmark	"Condition code, predicate and general purpose registers"	7% of injected faults on average lead to SDCs. 79% of injected faults do not affect application outputs. The number of SDCs is increasing as "the number of bit-flips" increases.
[29] [28]	Tesla C2075 with CUDA 4.1	"AES, matrixMult, MUMmerGPU, B-First, LIBOR MonteCarlo"	Destination register	The failure type occurring the most is crash (18%–50%); the second type is SDC (8%–40%). Matrix multiplication and AES encryption have the first and second highest SDC rates, respectively.
[12]	Nvidia GT200	"Histogram, Matrix mult. Mergesort, Reduction, Scaler prod. Scan, Transpose, Vector addition"	Register file	They compared the CPU and GPU. They observed that the number of SDCs in GPUs is about 4 times higher than in CPUs.
[26] [34] [27]	Tesla C series	14 applications from "Parboil, Rodinia Benchmarks and CUDA SDK pkg."	Destination register, address register	In some applications, failure rates are higher than 90%. At most, crash rate is 71% and SDC rate is 38%. HashGPU has the highest SDC rate, while MonteCarlo has the lowest SDC rate.
[116]	AMD Radeon HD5870 and AMD Radeon 7970	8 applications from AMD OpenCL SDK and 14 applications from AMD-APP-SDK	"Register file, local memory, active mask stack"	They have performed a statistical fault injection to evaluate AVF of some GPU structures. They found that two GPU structures, i.e., local memory and register file, have lower AVF than CPU structures (register and cache).
[115]	GeForce GTX480	12 applications from "Rodinia benchmark, ispass2009 benchmarks, CUDA SDK package"	"Register file, shared memory, SIMT stack, instruction buffer"	The silent error rates of all applications except two, i.e., Kmeans and MergeSort, are higher than the detectable uncorrectable error rates. Although any fault in instruction buffer can lead to SDCs easily, they have not encountered any SDC in the instruction buffer either.
[123]	Tesla S1070	7 applications from Parboil and 2 applications from GPU-SDK	Selected virtual variables (ALU/FPU register)	An injected fault into a FP, pointer, or integer may cause an SDC ("with 39%, 18%, 45% average possibility," respectively). However, failures are mostly caused by integer and pointer errors.
[85]	Nvidia Kepler K20 and K40	8 applications from different benchmarks (Hotspot, LuD, NW, lavaMD, ACCL,LULESH, MSort, QSort)	Register file, instruction outputs	An injected fault with a random value leads to lower PVF and AVF for Mergesort and Quicksort than an injected fault with a single-bit flip. There is no significant difference between the two injection modes (random value/single-bit flip) for AVF and PVF values of other applications.

and selecting enough threads from each group [26, 34]. However, we do not know how these two different approaches affect the results.

Injected Levels: Some GPU FI studies in the literature perform the FI experiments at the architectural level due to scalability concerns [26, 34, 45, 46]. Conversely, some studies [12, 29, 115, 116] perform FI at the microarchitecture level. The microarchitecture level provides flexibility because it does not require a real hardware and allows very detailed analysis. On the other hand, performing FI experiments at the architectural level, namely on real hardware, is more time efficient since it allows more coarse-grained analysis than the microarchitecture level. Given the scale and level of parallelism of GPU applications, it is important to be able to conduct FI experiments through less time-consuming means.

Analyzing Methodology: Studies in the literature try to understand the failure behaviors of general-purpose GPU applications by analyzing the results of FI experiments. However, studies lack a systematic methodology. Many studies emphasize that SDC rates (0% to 40%) vary greatly from application to application [26, 27, 29, 34]. However, in the literature, we did not find a comprehensive and systematic study that tries to understand the causes of differences in SDC rates or crash rates. Only a few studies indicate that applications can be classified according to SDC rates [26, 27]. However, one of these works does not suggest a classification method, yet the method suggested by the other work is quite inadequate.

Because the behaviors of CPU and GPU applications are very different from each other, knowing specific behaviors of GPU applications can help to develop application-specific fault tolerance methods [27, 29]. Therefore, more detailed studies are needed for GPUs.

The following questions need to be addressed with FI experiments:

- Is the only reason for differences in SDC rates the features of the applications? If we use different injection methods and tools, can we observe a significant change in SDC rates? To what extent does the hardware affect SDC and crash rates?
- Do the injected locations (destination register, condition code, shared memory), instruction types, or execution paths affect the rates of SDC and crash? Do the faults injected into the functional or control units cause more SDCs or crashes than those injected into memory?
- Which fault is more likely to spread or cause more errors? Can we identify the "fault-error-failure chains" of injected faults? Are there any distinctive, specific patterns between faults and failures?
- Which characteristics of applications may affect SDC and crash rates and to what extent? Can we develop a method for testing this?

7 RELIABILITY SCHEMES FOR HETEROGENEOUS HPCS

In general, homogeneous systems are more stable than heterogeneous systems since they are analyzed more and their error behaviors are well known. Today, almost all fault tolerance mechanisms that are used in heterogeneous systems are designed and optimized for homogeneous systems [4, 19, 56, 76]. However, there are many differences between homogeneous and heterogeneous systems in terms of architecture and execution models; therefore, it is unlikely that the technologies used in homogeneous systems will be fully compatible with the heterogeneous systems.

Since the integration of GPUs in HPCs is quite fast, the reliability requirements of heterogeneous HPCs were compensated with the technologies borrowed from homogeneous HPCs. Since GPUs did not have to be highly reliable before they were used for general purposes, they have not been analyzed adequately and fault tolerance mechanisms specific for these processors have not been developed. But the reliability challenges of GPUs are increasing in terms of both software and hardware, because of use today for mission-critical tasks in HPCs.

Furthermore, in next-generation HPCs, as the share of GPUs and the amount of usage increase, systems will be more prone to soft errors and the GPU-related failure rate will increase significantly. This suggests that fault tolerance mechanisms specific to next-generation heterogeneous systems should be developed. Especially when considering the increasing size of HPCs, it is emphasized that low-overhead fault tolerance approaches are needed in order not to reduce system performance. Because of the increasing failure rates in long-running applications, the fault tolerance techniques used today will not work in the future.

Today, the most common fault tolerance technique for HPCs is Checkpoint/Restart (or Recovery) (CR). Although this technique is fairly simple, it is a fundamental fault tolerance mechanism especially used to prevent failures of long-running applications. In this technique, the stable states of a running application are periodically backed up to a reliable storage. When an error occurs, the application returns to its last stable point and restarts from that point. Thus, the part of the application that already finished correctly is recovered. Though its idea is straightforward, it has some difficulties in practice. Many studies such as [14, 56, 76] have tried to eliminate its difficulties in order to use this technique efficiently in next-generation systems, but few studies have been tried to adapt this technique to GPU-based systems [35, 51, 70].

In this section, we will provide an overview of the CR models developed for GPU-based HPCs.

7.1 Checkpoint/Recovery Schemes for GPU-Based Heterogeneous Systems

Despite the fact that CR is a prevalent and basic fault tolerance technique [100] in HPCs and many different implementations for homogeneous systems have been proposed, we have found few CR models in the literature for heterogeneous systems. In this section, we list the CR schemes developed for GPU-based heterogeneous systems.

7.1.1 HiAL-Ckpt. GPUs and CPUs have different MTBF values since their reliability attributes are rather diverse. So it is necessary to set different checkpoint intervals for CPU and GPU according to their MTBFs. Since CPUs are more reliable, their MTBF values will be larger than GPUs'. In other words, we need to define multiple GPU checkpoints between two CPU checkpoints. If an error occurs on the CPU, this can lead to system failure, because the error can affect global components. However, any error that may occur in the GPU causes only a local failure that will affect the work on that GPU, unless it is an error affecting the shared components. However, in both cases, the closest checkpoint for the GPU can be used for recovery. In HiAL-Ckpt, which provides an application-level checkpointing, while the necessary information at the CPU checkpoints is copied to the disk, at the GPU checkpoints, data is stored in the CPU memory [120].

7.1.2 CheCUDA. To increase the reliability of GPU applications, the checkpointing mechanism must be able to record GPU states. Thus, the GPU's state changes can be tracked and used during the recovery phase. CheCUDA is designed as "an add-on to the standard Berkeley Labs Checkpoint Restart (BLCR)." CheCuda uses the basic CUDA driver API to track changes in GPU states. While the CPU checkpoints are taken by BLCR, CheCuda only checks and records GPU states in the software level. However, since the BLCR does not support GPUs, the GPU content must be saved and deleted before the BLCR starts. During the recovery phase, CheCuda launches the application running on the GPU from the nearest stable state using the stored information [108].

7.1.3 Hybrid Kernel Checkpoint. This model has been developed as totally transparent to the programmer by taking advantage of the Single Instruction Multiple Threads feature of GPU applications. It combines "PTX stub inject technology and dynamic library hijack mechanism" to provide in-kernel-state checkpointing. Thus, a noticeable error in the code running on the GPU can be corrected by partially recomputing the region by following the internal state of the GPU.

Hybrid Kernel Checkpoint periodically records "the current execution state of the GPU," and in the event of a failure, it performs the roll-back operations in order to return to the nearest stable state [94].

7.1.4 CudaCR. This model proposes a scalable application-level CR scheme in order to save the in-kernel state of the GPU. To be able to capture the computation states in each kernel, it converts the GPU and CPU codes into new codes with the help of a precompiler. However, in order to make this model work correctly and keep checkpoints consistent, all threads in the same block need to be synchronized just before getting checkpoints. This is why the programmer should take care when setting checkpoints' location. An asynchronous checkpoint is possible among the blocks because CUDA does not support synchronization of threads in different blocks. However, for this to happen, threads in different blocks should not be allowed to modify the same global memory location [77].

7.1.5 HeteroCheckpoint. This model presents a unified CR mechanism by combining the GPU-CPU memory state. To ensure the coherence of the GPU-CPU combined memory state, we need to synchronize all threads first. Then all the important variables related to the application must be transferred from the device memory to the nonvolatile (NV) memory. However, some applications may have multiple kernels, and some variables may not be modified throughout the kernels. To reduce the effect of the memory bandwidth limit, these unmodified variables can be transferred to NVM in parallel, while the application's execution continues, before the checkpointing process starts. This will reduce traffic between GDRAM and NVM. However, the programmer must specify explicitly which variables will not modify any longer. Also, if a variable does not modify between two checkpoints, it will not need to be copied again. Two-bit prediction based on the checksum mechanism is proposed to figure out these unmodified variables [51].

7.1.6 PartialRC. In this model, the programmer inserts the "original FT-Sections" into the application code running on a heterogeneous system, and the PartialRC compiler converts each of the original FT-Sections into modified FT-Sections in order to follow up all GPU-operations executed in the FT-Section, which is called "GPU-Op trace." The GPU-Op trace and localization module together provide the necessary information to determine the fault location and which GPU operations need to be repeated, if there are any errors in the calculations of the GPU. However, in order to be able to perform "partial recomputing," it is necessary to provide "precise error information" for all incorrect GPU computations; this is the responsibility of the error detection mechanism used in the system. Since they assume that "CPU is reliable," they save the data in CPU memory for all checkpoint operations [121].

7.1.7 Low-Overhead Diskless CheckPoint. It can be quite expensive to save the current state of the large-scale scientific applications to the disk during checkpointing. It is clear that classic disk-based checkpointing does not hold promise for future HPCs due to their increasing I/O bottleneck, especially given that applications running on next-generation supercomputers will be larger. To deal with this problem, Diskless CheckPoint proposes recording checkpoint data to memory. However, for this method to be feasible, it is necessary to increase the memory reliability by using replication or encoding techniques. However, while the replication method is not scalable for HPCs, the encoding techniques are highly complex and time consuming. This model suggests using the Reed-Solomon encoding technique, but it needs to be done in parallel with application execution to be able to perform the encoding process faster. To do this, instead of using spare nodes, using idle CPU cores or GPUs will help to overcome some drawbacks such as energy consumption and scalability [35].

7.1.8 Fault-Tolerant Software Framework for Memory. Although the new generation of GPU DRAMs are protected by SECDED ECC, many of the works we also have presented here show that SECDED ECC is inadequate for HPCs. This scheme presents a software-only fault tolerance technique that can detect and correct bit-flip errors in GPU DRAMs. While encoding techniques are used to detect the errors, a checkpoint model helps for the correction. Since correction is more difficult than detection and takes more time, using encoding techniques in the correction process can greatly reduce performance. That's why this model keeps a copy of the GPU program data in memory, and when an error is detected with a parity-based encoding, the GPU program is restarted based on the data stored in memory [58].

8 CHECKPOINT/RECOVERY CHALLENGES AND OPPORTUNITIES

When we examine CR techniques in general, it can be said that everything we can count as challenges for homogeneous systems is also a challenge for heterogeneous systems, but heterogeneous systems also have additional challenges.

8.1 Application-Level Checkpointing versus System-Level Checkpointing

Most of the checkpointing models developed for GPU-based heterogeneous systems have been implemented as application/user-level checkpointing. The most important advantages of this are flexibility and simplicity of implementation. At this level, however, the users or programmers are responsible for choosing the optimum control points. Besides, the user/programmer must explicitly specify which data will be copied during checkpointing. This is a burden for the programmer, but this significantly reduces the size of the data to be backed up. Given that most GPU applications have their own inherent error characteristics, application-level checkpointing can improve the performance of these applications. However, we think that it is very difficult to determine the optimum control points, especially by the user. Intelligent methods are needed to automatically pick control points at the most appropriate locations.

8.2 Where Will the Checkpointing Data Be Stored?

The memory in which the checkpointing data is stored must be fairly reliable so that the application can continue from the previous stable state in the event of an error. However, today's HPC applications generate quite large amounts of checkpointing data. Writing this amount of data to remote storage causes the checkpointing time to increase significantly and the performance of the application to decrease dramatically. This is a bigger problem for heterogeneous HPCs because they have a more complex memory hierarchy than homogeneous systems and limited data transfer bandwidth despite the rapid increase in GPU memory capacity. Since the CPU is more reliable than the GPU, GPU checkpointing data is recommended to be stored in the CPU memory, but if the data transfer quality is not at the desired level for next-generation HPCs, it will not be possible to implement such models for applications that generate large amounts of data. Due to both scalability and reliability issues, GPU checkpointing data stored in GPU memory does not seem to be a viable solution either. To solve this problem, additional solutions either in software or in hardware are needed to increase the GPU data transfer bandwidth and memory reliability. Suggested approaches such as "incremental checkpointing" to improve data transfer speed and reduce runtime overhead of the applications may work if they can be implemented for GPUs. Also, if the computation can safely continue during the checkpointing, the application performance will not be significantly affected by checkpointing burdens.

8.3 In-Kernel CR or Out-of Kernel CR

GPU applications can perform massive amounts of computation in parallel. This advantage, which increases the GPU's application performance, can become a disadvantage at the same time as this can lead to a significant loss in the event of an error. In classic CPU-based checkpointing models, an error in a GPU thread causes thousands of threads' tasks to recalculate unnecessarily, because these types of checkpointing models cannot monitor and catch GPU internal states. "The device code running on the GPU is called kernel" [94]. Although kernels are initialized and controlled by the CPU, all calculations are done and the results are generated within the GPU. However, the GPU communicates with the CPU only 2x during the whole process: at the very beginning to get the data to be processed and to transfer the results of the calculations to the CPU. All of this shows us that we need to trace and record GPU states, regardless of the CPU, during the kernels' execution. Implementing this model, called "in-kernel checkpoint," is a real challenge due to the inherit complex nature of GPU applications. We can say that getting GPU internal states in arbitrary time is quite troublesome when we consider a number of factors, such as that most GPU applications have multiple kernels, variable values can be changed in different kernels, and though threads in all blocks can access global memory without restrictions, synchronization of threads in the same thread block is allowed but synchronization of threads in different blocks is not allowed. On CPUs, many operations can be performed automatically (for instance, application page-level information can be tracked easily) unlike GPUs; therefore, GPUs need some additional software-based solutions. Moreover, "the lack of direct I/O access" [51] from the GPU and the inability to efficiently obtain the GPU's internal state make it difficult to develop a GPU-specific CR model. As a result, there is no standard low-overhead in-kernel checkpoint scheme for GPU-based HPCs.

8.4 GPU Error Detection

Apart from CR systems in the literature, there are few error detection and recovery proposals developed for general-purpose GPUs. We do not think hardware-based ones [92] are applicable for today's HPCs because of the need for extra hardware support. Software-based ones [23, 58, 59] usually use coding or application-level techniques (i.e., executing a redundant copy of an application).

Although it is easier to detect the error than to correct the error, we know that the techniques used for fault detection have a negative impact on performance. Checkpoint schemes used for error recovery cannot detect errors. Therefore, these models are designed by combining with error detection techniques. Nowadays it is impossible to detect faults that occur in some components of GPUs such as logic and computational units, warp schedulers, dispatcher units, control data paths, and interconnect networks due to lack of hardware-based fault tolerance technologies except hardware ECC, which can detect bit-flip faults in DRAM, register files, and cache. For this reason, software-based approaches that are implemented with encoding techniques are used to detect these faults. However, encoding techniques are both very complex and time consuming. Since the amount of checkpointing data is ultra-large, coding techniques can significantly reduce performance.

9 SUMMARY

Table 5. Summary of Research Works

Ref.	Category	Short Description
[88]	Evaluation of the Reliability of Homogeneous HPCs	They presented statistical analysis results of error logs of 22 HPC systems with different sizes and hardware specifications.
[25]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed the failures of 10 HPC clusters. They focused on correlations between failures.
[42]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed system logs to identify failures; suggests a reliability model by exploiting the underlying statistical properties of events.
[102]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed DRAM errors "in a large high-performance computing cluster."
[103]	Evaluation of the Reliability of Homogeneous HPCs	They examined the effect of aging on DRAM faults. This study also examined SRAM faults.
[101]	Evaluation of the Reliability of Homogeneous HPCs	In particular, they examined the efficiency of error correction codes. They note that SECDED ECC is very inadequate for modern DRAMs.
[7]	Evaluation of the Reliability of Homogeneous HPCs (Although the system used in this study has one GPU, it is not clear whether logs of this part were analyzed. Also, they didn't present any result related with GPUs.)	They analyzed the memory error logs, which are obtained from an unprotected system. This means that the error correction mechanism is disabled in the system. In this way, any kind of error in memory can be recorded [103]. They found that memory errors had high temporal and spatial correlations.
[49]	Evaluation of the Reliability of Homogeneous HPCs	This work deals with hard errors. Several findings of this study indicate that the main cause of memory errors is hard errors, not soft errors.
[57]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed failure logs gathered from three HPCs
[79]	Evaluation of the Reliability of Homogeneous HPCs.	They analyzed the system logs to figure out the reliability of the system that contains the k nodes.
[17]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed logs from two large-scale systems in order to characterize DRAM and SRAM faults.
[71]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed log files of five HPCs with different features. They especially deal with challenges of distinguishing critical events from others.
[47]	Evaluation and Improvement of the Reliability of Homogeneous HPCs	They analyzed the error logs of an HPC for 5 years in order to understand characteristics of HW failures ("processors, memory, storage and network components").
[6]	Evaluation and Improvement of the Reliability of HPCs	In this study, the failure logs of five systems were analyzed in order to understand which type of failures caused a change in the state of the system. Two of these systems are heterogeneous and three of them are homogeneous systems.
[30]	Evaluation of the Reliability of HPCs	They have defined failure scenarios by analyzing a heterogeneous system.

(Continued)

Table 5. Continued

Ref.	Category	Short Description
[107]	Evaluation of the Reliability of Homogeneous HPCs	In this study, the system logs of a homogeneous supercomputer were analyzed.
[89]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed systems that have different technologies, and with the development of technology, they concluded that systems with newer hardware technologies are not necessary to be more reliable.
[36]	Evaluation of the Reliability of Homogeneous HPCs	They focused on failures that occur on one node and do not spread to other nodes. They analyzed the TBF distributions of the nodes to understand the failure behavior of each node.
[37]	Evaluation of the Reliability of Homogeneous HPCs	In order to estimate the time interval between two successive system failures, they analyzed individual failure data of the nodes.
[118]	Evaluation of the Reliability of Homogeneous HPCs	This study is particularly focused on hardware failures. They examined roughly 290,000 hardware failures collected from several systems with different configurations.
[111]	Evaluation of the Reliability of Homogeneous HPCs	They examined failure data from a homogeneous HPC in order to understand correlated failures.
[40]	Evaluation of the Reliability of HPCs	They analyzed log files of five HPCs. One of them is a GPU-based HPC. They observed that "the temporal recurrence property" was different for different types of failure but similar among the systems. And they found that "the spatial distribution of failures is not uniform" at any compute granularity across systems.
[67] & [112–114]& [41]	Evaluation of the Reliability of Heterogeneous HPCs	They explored and described several types of soft errors on a large-scale GPU-based HPC (Titan). Particularly, they focus on GPU errors.
[22]	Evaluation of the Reliability of Heterogeneous HPCs	They examined 261-day failure data of a petascale GPU-based heterogeneous system (BlueWaters).
[34]	Evaluation of the Reliability of Heterogeneous HPCs	This work analyzes the results of some experimental research on GPGPUs' reliability. They compared a homogeneous and a GPU-based heterogeneous system.
[8]	Evaluation of the Reliability of Heterogeneous HPCs	They investigate the effect of ECC errors on molecular dynamics (MD) simulations. Because the ratio of ECC errors (corrected or not corrected) on GPUs is unknown, this work explores the sources and effects of ECC errors.
[68]	Evaluation and Improvement of the Reliability of Heterogeneous HPCs	Analyzing the error data of a GPU-based large-scale heterogeneous system, they tried to understand the relationship between temperature, power consumption, and GPU soft errors. They focus on only single-bit errors (SBE).

(Continued)

Table 5. Continued

Ref.	Category	Short Description
[69]	Evaluation and Improvement of the Reliability of Heterogeneous HPCs	Similar to the previous study [68], they focused on GPU SBEs and they used the same data. They tried to present a model to predict SBEs.
[54]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed the fi5-year data collected from a homogeneous system. They are particularly focused on correctable and uncorrectable memory errors.
[96]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed the DRAM and SRAM errors collected from a production system. The analysis was performed using two of the data sources used in the previous study [54].
[95]	Evaluation of the Reliability of Homogeneous HPCs	They analyzed corrected memory errors. Unlike other studies, they analyzed errors that occur in other memory components. These are "memory controllers, buses, channels, and memory modules."

REFERENCES

[1] 2018. *pizDaint Supercomputer*. Retrieved from https://www.cscs.ch/computers/piz-daint/.

[2] 2018. *Titan Supercomputer*. Retrieved from https://www.olcf.ornl.gov/titan/.

[3] 2018. *Top500 HPC List*. Retrieved from https://www.top500.org.

[4] Muhammad Alfian Amrizal, Pei Li, Mulya Agung, Ryusuke Egawa, and Hiroyuki Takizawa. 2018. A failure prediction-based adaptive checkpointing method with less reliance on temperature monitoring for HPC applications. In *2018 IEEE International Conference on Cluster Computing (CLUSTER'18)*. IEEE, 515–523.

[5] Elisabeth Baseman, Nathan DeBardeleben, Kurt Ferreira, Scott Levy, Steven Raasch, Vilas Sridharan, Taniya Siddiqua, and Qiang Guan. 2016. Improving DRAM fault characterization through machine learning. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop*. IEEE, 250–253.

[6] Leonardo Bautista-Gomez, Ana Gainaru, Swann Perarnau, Devesh Tiwari, Saurabh Gupta, Christian Engelmann, Franck Cappello, and Marc Snir. 2016. Reducing waste in extreme scale systems through introspective analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 212–221.

[7] Leonardo Bautista-Gomez, Ferad Zyulkyarov, Osman Unsal, and Simon McIntosh-Smith. 2016. Unprotected computing: A large-scale study of DRAM raw error rate on a supercomputer. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. IEEE, 645–655.

[8] Robin M. Betz, Nathan A. DeBardeleben, and Ross C. Walker. 2014. An investigation of the effects of hard and soft errors on graphics processing unit-accelerated molecular dynamics simulations. *Concurrency and Computation: Practice and Experience* 26, 13 (2014), 2134–2140.

[9] Franck Cappello. 2009. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *International Journal of High Performance Computing Applications* 23, 3 (2009), 212–226.

[10] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. 2014. Toward exascale resilience: 2014 update. *Supercomputing Frontiers and Innovations* 1, 1 (2014), 5–28.

[11] Nicholas P. Cardo. [n.d.]. Detecting and managing GPU failures.

[12] Athanasios Chatzidimitriou, Manolis Kaliorakis, Sotiris Tselonis, and Dimitris Gizopoulos. 2017. Performance-aware reliability assessment of heterogeneous chips. In *2017 IEEE 35th VLSI Test Symposium (VTS'17)*. IEEE, 1–6.

[13] Min Chen, Shiwen Mao, and Yunhao Liu. 2014. Big data: A survey. *Mobile Networks and Applications* 19, 2 (2014), 171–209.

[14] Daniel Dauwe, Sudeep Pasricha, Anthony A. Maciejewski, and Howard Jay Siegel. 2017. An analysis of resilience techniques for exascale computing platforms. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW'17)*. IEEE, 914–923.

[15] Nathan DeBardeleben, Sean Blanchard, David Kaeli, and Paolo Rech. 2015. Field, experimental, and analytical data on large-scale HPC systems and evaluation of the implications for exascale system design. In *2015 IEEE 33rd VLSI Test Symposium (VTS'15)*. IEEE, 1–2.

[16] Nathan DeBardeleben, Sean Blanchard, Laura Monroe, Phil Romero, Daryl Grunau, Craig Idler, and Cornell Wright. 2013. GPU behavior on a large HPC cluster. In *European Conference on Parallel Processing*. Springer, 680–689.

[17] Nathan DeBardeleben, Sean Blanchard, Vilas Sridharan, Sudhanva Gurumurthi, Jon Stearley, K. Ferreira, and John Shalf. 2014. Extra bits on SRAM and DRAM errors–More data from the field. In *IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE'14)*.

[18] David Defour and Eric Petit. 2013. GPUburn: A system to test and mitigate GPU hardware failures. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII'13)*. IEEE, 263–270.

[19] Sheng Di and Franck Cappello. 2016. Adaptive impact-driven detection of silent data corruption for HPC applications. *IEEE Transactions on Parallel and Distributed Systems* 27, 10 (2016), 2809–2823.

[20] Sheng Di, Rinku Gupta, Marc Snir, Eric Pershey, and Franck Cappello. 2017. LogAider: A tool for mining potential correlations of HPC log events. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 442–451.

[21] Catello Di Martino, Marcello Cinque, and Domenico Cotroneo. 2012. Assessing time coalescence techniques for the analysis of supercomputer logs. In *2012 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12)*. IEEE, 1–12.

[22] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K. Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. 2014. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. IEEE, 610–621.

[23] Martin Dimitrov, Mike Mantor, and Huiyang Zhou. 2009. Understanding software approaches for GPGPU reliability. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. ACM, 94–104.

[24] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Chi Xuebin Chi, Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Jin Zhong Jin, Yutaka Ishikawa, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney MacCabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias S. Mueller, Wolfgang E. Nagel, Hiroshi Nakashima, Michael E. Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad Van Der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick. 2011. The international exascale software project roadmap. *International Journal of High Performance Computing Applications* 25, 1 (2011), 3–60.

[25] Nosayba El-Sayed and Bianca Schroeder. 2013. Reading between the lines of failure logs: Understanding how HPC systems fail. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*. IEEE, 1–12.

[26] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2014. GPU-QIN: A methodology for evaluating the error resilience of GPGPU applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'14)*. IEEE, 221–230.

[27] Bo Fang, Karthik Pattabiraman, Matei Ripeanu, and Sudhanva Gurumurthi. 2016. A systematic methodology for evaluating the error resilience of GPGPU applications. *IEEE Transactions on Parallel and Distributed Systems* 27, 12 (2016), 3397–3411.

[28] Bo Fang, Jiesheng Wei, Karthik Pattabiraman, and Matei Ripeanu. 2012. Evaluating error resiliency of GPGPU applications. In *2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC'12)*. IEEE, 1502–1503.

[29] Bo Fang, Jiesheng Wei, Karthik Pattabiraman, and Matei Ripeanu. 2012. Towards building error resilient GPGPU applications. In *SC Companion: High Performance Computing, Networking Storage and Analysis*.

[30] Valerio Formicola, Saurabh Jha, Daniel Chen, Fei Deng, Amanda Bonnie, Mike Mason, Jim Brandt, Ann Gentile, Larry Kaplan, Jason Repik, Jeremy Enos, Mike Showerman, Annette Greiner, Zbigniew Kalbarczyk, Ravishankar K. Iyer, and Bill Krammer. 2017. Understanding fault scenarios and impacts through fault injection experiments in Cielo. *Urbana* 51 (2017), 61801.

[31] Ana Gainaru, Franck Cappello, and William Kramer. 2012. Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems. In *2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS'12)*. IEEE, 1168–1179.

[32] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. 2012. 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, 77.

[33] Al Geist and Daniel A. Reed. 2017. A survey of high-performance computing scaling challenges. *International Journal of High Performance Computing Applications* 31, 1 (2017), 104–113.

[34] L. Bautista Gomez, Franck Cappello, Luigi Carro, Nathan DeBardeleben, Bo Fang, Sudhanva Gurumurthi, Karthik Pattabiraman, Paolo Rech, and M. Sonza Reorda. 2014. GPGPUs: How to combine high computational power with

high reliability. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 341.

[35] Leonardo Bautista Gomez, Akira Nukada, Naoya Maruyama, Franck Cappello, and Satoshi Matsuoka. 2010. Low-overhead diskless checkpoint for hybrid computing systems. In *2010 International Conference on High Performance Computing (HiPC'10)*. IEEE, 1–10.

[36] Narasimha Raju Gottumukkala, Chokchai Box Leangsuksun, Narate Taerat, Raja Nassar, and Stephen L. Scott. 2007. Reliability-aware resource allocation in HPC systems. In *2007 IEEE International Conference on Cluster Computing*. IEEE, 312–321.

[37] Narasimha Raju Gottumukkala, Raja Nassar, Mihaela Paun, Chokchai Box Leangsuksun, and Stephen L. Scott. 2010. Reliability of a system of k nodes for high performance computing applications. *IEEE Transactions on Reliability* 59, 1 (2010), 162–169.

[38] Jiexing Gu, Ziming Zheng, Zhiling Lan, John White, Eva Hocks, and Byung-Hoon Park. 2008. Dynamic meta-learning for failure prediction in large-scale systems: A case study. In *37th International Conference on Parallel Processing, 2008 (ICPP'08)*. IEEE, 157–164.

[39] Prashasta Gujrati, Yawei Li, Zhiling Lan, Rajeev Thakur, and John White. 2007. A meta-learning failure predictor for blue gene/l systems. In *International Conference on Parallel Processing, 2007 (ICPP'07)*. IEEE, 40–40.

[40] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: Long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 44.

[41] Saurabh Gupta, Devesh Tiwari, Christopher Jantzi, James Rogers, and Don Maxwell. 2015. Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. IEEE, 37–44.

[42] Thomas J. Hacker, Fabian Romero, and Christopher D. Carothers. 2009. An analysis of clustered failures on large supercomputing systems. *Journal of Parallel and Distributed Computing* 69, 7 (2009), 652–665.

[43] Imran S. Haque and Vijay S. Pande. 2010. Hard data on soft errors: A large-scale assessment of real-world error rates in GPGPU. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 691–696.

[44] Siva Kumar Sastry Hari, Sarita V. Adve, Helia Naeimi, and Pradeep Ramachandran. 2013. Relyzer: Application resiliency analyzer for transient faults. *IEEE Micro* 33, 3 (2013), 58–66.

[45] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. 2015. Sassifi: Evaluating resilience of GPU applications. In *Proceedings of the Workshop on Silicon Errors in Logic-System Effects (SELSE'15)*.

[46] Siva Kumar Sastry Hari, Timothy Tsai, Mark Stephenson, Stephen W. Keckler, and Joel Emer. 2017. Sassifi: An architecture-level fault injection tool for GPU application resilience evaluation. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'17)*. IEEE, 249–258.

[47] Eric Heien, Derrick Kondo, Ana Gainaru, Dan LaPine, Bill Kramer, and Franck Cappello. 2011. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 45.

[48] Marcus Hilbrich, Matthias Weber, and Ronny Tschüter. 2013. Automatic analysis of large data sets: A walk-through on methods from different perspectives. In *2013 International Conference on Cloud Computing and Big Data (CloudCom-Asia'13)*. IEEE, 373–380.

[49] Andy A. Hwang, Ioan A. Stefanovici, and Bianca Schroeder. 2012. Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 111–122.

[50] Elmira Yu Kalimulina. 2017. Analysis of system reliability with control, dependent failures, and arbitrary repair times. *International Journal of System Assurance Engineering and Management* 8, 1 (2017), 180–188.

[51] Sudarsun Kannan, Naila Farooqui, Ada Gavrilovska, and Karsten Schwan. 2014. Heterocheckpoint: Efficient checkpointing for accelerator-based systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. IEEE, 738–743.

[52] David B. Kirk and W. Hwu Wen-Mei. 2016. *Programming Massively Parallel Processors: A Hands-On Approach*. Morgan Kaufmann.

[53] Zhiling Lan, Jiexing Gu, Ziming Zheng, Rajeev Thakur, and Susan Coghlan. 2010. A study of dynamic meta-learning for failure prediction in large-scale systems. *Journal of Parallel and Distributed Computing* 70, 6 (2010), 630–643.

[54] Scott Levy, Kurt B. Ferreira, Nathan DeBardeleben, Taniya Siddiqua, Vilas Sridharan, and Elisabeth Baseman. 2018. Lessons learned from memory errors observed over the lifetime of Cielo. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 43.

[55] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Morris Jette, and Ramendra Sahoo. 2006. Bluegene/l failure analysis and prediction models. In *International Conference on Dependable Systems and Networks, 2006 (DSN'06)*. IEEE, 425–434.

[56] Yudan Liu, Raja Nassar, Chokchai Leangsuksun, Nichamon Naksinehaboon, Mihaela Paun, and Stephen L. Scott. 2008. An optimal checkpoint/restart model for a large scale high performance computing system. In *IEEE International Symposium on Parallel and Distributed Processing, 2008 (IPDPS'08)*. IEEE, 1–9.

[57] Charng-Da Lu. 2013. Failure data analysis of HPC systems. *arXiv preprint arXiv:1302.4779* (2013).

[58] Naoya Maruyama, Akira Nukada, and Satoshi Matsuoka. 2010. A high-performance fault-tolerant software framework for memory on commodity GPUs. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10)*. IEEE, 1–12.

[59] Naoya Maruyama, Akira Nukada, and Satoshi Matsuoka. 2009. Software-based ECC for GPUs. In *2009 Symposium on Application Accelerators in High Performance Computing (SAAHPC'09)*, Vol. 107.

[60] Justin Meza, Qiang Wu, Sanjeev Kumar, and Onur Mutlu. 2015. Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. IEEE, 415–426.

[61] Sparsh Mittal and Jeffrey S. Vetter. 2015. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 69.

[62] Sparsh Mittal and Jeffrey S. Vetter. 2016. A survey of techniques for modeling and improving reliability of computing systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 4 (2016), 1226–1238.

[63] Shubhendu S. Mukherjee, Joel Emer, and Steven K. Reinhardt. 2005. The soft error problem: An architectural perspective. In *11th International Symposium on High-Performance Computer Architecture, 2005 (HPCA-11'05)*. IEEE, 243–247.

[64] Shubhendu S. Mukherjee, Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003 (MICRO-36'03)*. IEEE, 29–40.

[65] Onur Mutlu. 2013. Memory scaling: A systems architecture perspective. In *2013 5th IEEE International Memory Workshop (IMW'13)*. IEEE, 21–25.

[66] Onur Mutlu and Lavanya Subramanian. 2015. Research problems and opportunities in memory systems. *Supercomputing Frontiers and Innovations* 1, 3 (2015), 19–55.

[67] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H. Rogers. 2016. A large-scale study of soft-errors on GPUs in the field. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA'16)*. IEEE, 519–530.

[68] Bin Nie, Ji Xue, Saurabh Gupta, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2017. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'17)*. IEEE, 22–31.

[69] Bin Nie, Ji Xue, Saurabh Gupta, Tirthak Patel, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. 2018. Machine learning models for GPU error prediction in a large scale HPC system. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'18)*. IEEE, 95–106.

[70] Akira Nukada, Hiroyuki Takizawa, and Satoshi Matsuoka. 2011. NVCR: A transparent checkpoint-restart library for NVIDIA CUDA. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW'11)*. IEEE, 104–113.

[71] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007 (DSN'07)*. IEEE, 575–584.

[72] Adam J. Oliner, Alex Aiken, and Jon Stearley. 2008. Alert detection in system logs. In *8th IEEE International Conference on Data Mining, 2008 (ICDM'08)*. IEEE, 959–964.

[73] David J. Palframan, Nam Sung Kim, and Mikko H. Lipasti. 2014. Precision-aware soft error protection for GPUs. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. IEEE, 49–59.

[74] Byung H. Park, Saurabh Hukerikar, Ryan Adamson, and Christian Engelmann. 2017. Big data meets HPC log analytics: Scalable approach to understanding systems at extreme scale. In *2017 IEEE International Conference on Cluster Computing (CLUSTER'17)*. IEEE, 758–765.

[75] Antonio Pecchia, Domenico Cotroneo, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. 2011. Improving log-based field failure data analysis of multi-node computing systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN'11)*. IEEE, 97–108.

[76] Swann Perarnau and Leonardo Bautista-Gomez. 2016. Monitoring strategies for scalable dynamic checkpointing. In *2016 7th International Green and Sustainable Computing Conference (IGSC'16)*. IEEE, 1–8.

[77] Behnam Pourghassemi and Aparna Chandramowlishwaran. 2017. cudaCR: An in-kernel application-level checkpoint/restart scheme for CUDA-enabled GPUs. In *2017 IEEE International Conference on Cluster Computing (CLUSTER'17)*. IEEE, 725–732.

[78] Fritz G. Previlon, Babatunde Egbantan, Devesh Tiwari, Paolo Rech, and David R. Kaeli. 2017. Combining architectural fault-injection and neutron beam testing approaches toward better understanding of GPU soft-error resilience. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS'17)*. IEEE, 898–901.

[79] Narasimha Raju, Y. Liu Gottumukkala, Chokchai B. Leangsuksun, Raja Nassar, and Stephen Scott. 2006. Reliability analysis in HPC clusters. In *Proceedings of the High Availability and Performance Computing Workshop*. 673–684.

[80] Paolo Rech, Caroline Aguiar, R. Ferreira, Christopher Frost, and Luigi Carro. 2012. Neutron radiation test of graphic processing units. In *2012 IEEE 18th International On-Line Testing Symposium (IOLTS'12)*. IEEE, 55–60.

[81] P. Rech, C. Aguiar, C. Frost, and L. Carro. 2013. An efficient and experimentally tuned software-based hardening strategy for matrix multiplication on GPUs. *IEEE Transactions on Nuclear Science* 60, 4 (2013), 2797–2804.

[82] Paolo Rech, Laércio Lima Pilla, Philippe Olivier Alexandre Navaux, and Luigi Carro. 2014. Impact of GPUs parallelism management on safety-critical and HPC applications reliability. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'14)*. IEEE, 455–466.

[83] Felipe Rosa, Fernanda Kastensmidt, Ricardo Reis, and Luciano Ost. 2015. A fast and scalable fault injection framework to evaluate multi/many-core soft error reliability. In *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS'15)*. IEEE, 211–214.

[84] Ramendra K. Sahoo, Adam J. Oliner, Irina Rish, Manish Gupta, José E. Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. 2003. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–435.

[85] Fernando Fernandes dos Santos and Paolo Rech. 2017. Analyzing the criticality of transient faults-induced SDCS on GPU applications. In *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*. ACM, 1.

[86] Hamid Sarbazi-Azad. 2016. *Advances in GPU Research and Practice*. Morgan Kaufmann.

[87] Horst Schirmeier, Martin Hoffmann, Christian Dietrich, Michael Lenz, Daniel Lohmann, and Olaf Spinczyk. 2015. FAIL*: An open and versatile fault-injection framework for the assessment of software-implemented hardware fault tolerance. In *2015 11th European Dependable Computing Conference (EDCC'15)*. IEEE, 245–255.

[88] Bianca Schroeder and Garth Gibson. 2010. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 337–350.

[89] Bianca Schroeder and Garth A. Gibson. 2007. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, Vol. 78. IOP Publishing, 012022.

[90] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. 2009. DRAM errors in the wild: A large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37. ACM, 193–204.

[91] John Shalf, Sudip Dosanjh, and John Morrison. 2010. Exascale computing technology challenges. In *International Conference on High Performance Computing for Computational Science*. Springer, 1–25.

[92] Jeremy W. Sheaffer, David P. Luebke, and Kevin Skadron. 2007. A hardware redundancy and recovery mechanism for reliable scientific computation on graphics processors. In *Graphics Hardware*, Vol. 2007. 55–64.

[93] Justin Y. Shi, Moussa Taifi, Abdallah Khreishah, and Jie Wu. 2011. Sustainable GPU computing at scale. In *2011 IEEE 14th International Conference on Computational Science and Engineering (CSE'11)*. IEEE, 263–272.

[94] Lin Shi, Hao Chen, and Ting Li. 2013. Hybrid CPU/GPU checkpoint for GPU-based heterogeneous systems. In *International Conference on Parallel Computing in Fluid Dynamics*. Springer, 470–481.

[95] Taniya Siddiqua, Athanasios E. Papathanasiou, Arijit Biswas, and Sudhanva Gurumurthi. 2013. Analysis and modeling of memory errors from large-scale field data collection. In *Workshop on Silicon Errors in Logic – System Effects (SELSE'13)*.

[96] Taniya Siddiqua, Vilas Sridharan, Steven E. Raasch, Nathan DeBardeleben, Kurt B. Ferreira, Scott Levy, Elisabeth Baseman, and Qiang Guan. 2017. Lifetime memory reliability data from the field. In *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'17)*. IEEE, 1–6.

[97] Matthew D. Sinclair, Johnathan Alsop, and Sarita V. Adve. 2015. Efficient GPU synchronization without scopes: Saying no to complex consistency models. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 647–659.

[98] Matthew D. Sinclair, Johnathan Alsop, and Sarita V. Adve. 2017. HeteroSync: A benchmark suite for fine-grained synchronization on tightly coupled GPUs. In *2017 IEEE International Symposium on Workload Characterization (IISWC'17)*. IEEE, 239–249.

[99] Marc Snir, Robert W. Wisniewski, Jacob A. Abraham, Sarita V. Adve, Saurabh Bagchi, Pavan Balaji, Jim Belak, Pradip Bose, Franck Cappello, Bill Carlson, Andrew A. Chien, Paul Coteus, Nathan A. DeBardeleben, Pedro C. Diniz, Christian Engelmann, Mattan Erez, Saverio Fazzari, Al Geist, Rinku Gupta, Fred Johnson, Sriram Krishnamoorthy, Sven Leyffer, Dean Liberty, Subhasish Mitra, Todd Munson, Rob Schreiber, Jon Stearley, and Eric Van Hensbergen. 2014. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications* 28, 2 (2014), 129–173.

[100] Lizandro D. Solano-Quinde, Brett M. Bode, and Arun K. Somani. 2010. Coarse grain computation-communication overlap for efficient application-level checkpointing for GPUs. In *2010 IEEE International Conference on Electro/Information Technology (EIT'10)*. IEEE, 1–5.

[101] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B. Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory errors in modern systems: The good, the bad, and the ugly. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 297–310.

[102] Vilas Sridharan and Dean Liberty. 2012. A study of DRAM failures in the field. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11.

[103] Vilas Sridharan, Jon Stearley, Nathan DeBardeleben, Sean Blanchard, and Sudhanva Gurumurthi. 2013. Feng shui of supercomputer memory positional effects in DRAM and SRAM faults. In *2013 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*. IEEE, 1–11.

[104] Jon Stearley, Robert Ballance, and Lara Bauman. 2012. A state-machine approach to disambiguating supercomputer event logs. In *2012 Workshop on Managing Systems Automatically and Dynamically (MAD'12)*. 155–192.

[105] Jon Stearley and Adam J. Oliner. 2008. Bad words: Finding faults in Spirit's syslogs. In *8th IEEE International Symposium on Cluster Computing and the Grid, 2008 (CCGRID'08)*. IEEE, 765–770.

[106] Mark Stephenson, Siva Kumar Sastry Hari, Yunsup Lee, Eiman Ebrahimi, Daniel R. Johnson, David Nellans, Mike O'Connor, and Stephen W. Keckler. 2015. Flexible software profiling of GPU architectures. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 185–197.

[107] Narate Taerat, Nichamon Naksinehaboon, Clayton Chandler, James Elliott, Chokchai Leangsuksun, George Ostrouchov, Stephen L. Scott, and Christian Engelmann. 2009. Blue gene/l log analysis and time to interrupt estimation. In *International Conference on Availability, Reliability and Security, 2009 (ARES'09)*. IEEE, 173–180.

[108] Hiroyuki Takizawa, Katsuto Sato, Kazuhiko Komatsu, and Hiroaki Kobayashi. 2009. CheCUDA: A checkpoint/restart tool for CUDA applications. In *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 408–413.

[109] Jingweijia Tan, Nilanjan Goswami, Tao Li, and Xin Fu. 2011. Analyzing soft-error vulnerability on GPGPU microarchitecture. In *2011 IEEE International Symposium on Workload Characterization (IISWC'11)*. IEEE, 226–235.

[110] Thanadech Thanakornworakij, Raja Nassar, Chokchai Box Leangsuksun, and Mihaela Paun. 2011. The effect of correlated failure on the reliability of HPC systems. In *2011 9th IEEE International Symposium on Parallel and Distributed Processing with Applications Workshops (ISPAW'11)*. IEEE, 284–288.

[111] Thanadech Thanakornworakij, Raja Nassar, Chokchai Box Leangsuksun, and Mihaela Paun. 2013. Reliability model of a system of k nodes with simultaneous failures for high-performance computing applications. *International Journal of High Performance Computing Applications* 27, 4 (2013), 474–482.

[112] Devesh Tiwari, Saurabh Gupta, George Gallarno, Jim Rogers, and Don Maxwell. 2015. Reliability lessons learned from GPU experience with the Titan supercomputer at Oak Ridge leadership computing facility. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 38.

[113] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, Luigi Carro, and Arthur Bland. 2015. Understanding GPU errors on large-scale HPC systems and the implications for system design and operation. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA'15)*. IEEE, 331–342.

[114] Devesh Tiwari, Saurabh Gupta, James H. Rogers, and Don E. Maxwell. 2015. *Experience with GPUs on the Titan Supercomputer from a Reliability, Performance and Power Perspective*. Technical Report. Oak Ridge National Laboratory (ORNL), Oak Ridge, TN. Oak Ridge Leadership Computing Facility (OLCF).

[115] Sotiris Tselonis and Dimitris Gizopoulos. 2016. GUFI: A framework for GPUs reliability assessment. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'16)*. IEEE, 90–100.

[116] Alessandro Vallero, Dimitris Gizopoulos, and Stefano Di Carlo. 2017. SIFI: AMD southern islands GPU microarchitectural level fault injector. In *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS'17)*. 138–144.

[117] Alessandro Vallero, Sotiris Tselonis, Dimitris Gizopoulos, and Stefano Di Carlo. 2018. Multi-faceted microarchitecture level reliability characterization for NVIDIA and AMD GPUs. In *2018 IEEE 36th VLSI Test Symposium (VTS'18)*. IEEE, 1–6.

[118] Guosai Wang, Lifei Zhang, and Wei Xu. 2017. What can we learn from four years of data center hardware failures? In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'17)*. IEEE, 25–36.

[119] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Online system problem detection by mining patterns of console logs. In *9th IEEE International Conference on Data Mining, 2009 (ICDM'09)*. IEEE, 588–597.

[120] Xinhai Xu, Yufei Lin, Tao Tang, and Yisong Lin. 2010. HiAL-Ckpt: A hierarchical application-level checkpointing for CPU-GPU hybrid systems. In *2010 5th International Conference on Computer Science and Education (ICCSE'10)*. IEEE, 1895–1899.

[121] Xin-Hai Xu, Xue-Jun Yang, Jing-Ling Xue, Yu-Fei Lin, and Yi-Song Lin. 2012. PartialRC: A partial recomputing method for efficient fault recovery on GPGPUs. *Journal of Computer Science and Technology* 27, 2 (2012), 240–255.
[122] Xuejun Yang, Zhiyuan Wang, Jingling Xue, and Yun Zhou. 2012. The reliability wall for exascale supercomputing. *IEEE Transactions on Computers* 61, 6 (2012), 767–779.
[123] Keun Soo Yim, Cuong Pham, Mushfiq Saleheen, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2011. Hauberk: Lightweight silent data corruption error detector for GPGPU. In *2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS'11)*. IEEE, 287–300.
[124] Mohamed Zahran. 2017. Heterogeneous computing: Here to stay. *Communications of the ACM* 60, 3 (2017), 42–45.
[125] Ziming Zheng, Zhiling Lan, Rinku Gupta, Susan Coghlan, and Peter Beckman. 2010. A practical failure prediction with location and lead time for blue gene/p. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W'10)*. IEEE, 15–22.

Received May 2018; revised January 2019; accepted November 2019