# Hybrid Checkpointing using Emerging Non-Volatile Memories for Future Exascale Systems

XIANGYU DONG and YUAN XIE
Pennsylvania State University
and
NAVEEN MURALIMANOHAR and NORMAN P. JOUPPI
Hetlett-Packard Labs

The scalability of future massively parallel processing (MPP) systems is being severely challenged by high failure rates. Current centralized hard disk drive (HDD) checkpointing results in overhead of 25% or more at petascale. As system becomes more vulnerable as the node count keeps increasing, novel techniques that enable fast and frequent checkpointing are critical to the future exascale system implementation.

In this work, we first introduce one of the emerging non-volatile memory technologies, *Phase-Change Random Access Memory* (PCRAM), as a proper candidate for the fast checkpointing device. After a thorough analysis of MPP systems failure rates and failure sources, we then use PCRAM to propose a hybrid local/global checkpointing mechanism, which not only provides a faster checkpoint storage, but also boosts the effectiveness of other orthogonal techniques such as incremental checkpointing and background checkpointing. Three variant implementations of the PCRAM-based hybrid checkpointing are designed to be adopted at different stages and to offer a smooth transition from the conventional in-disk checkpointing to the instant in-memory approach. Analyzing the overhead by using a hybrid checkpointing performance model, we show the proposed approach only incurs less than 3% performance overhead on a projected exascale system.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*Memory Technologies*; B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; C.5.1 [**Computer System Implementation**]: Large and Medium Computers—*Super Computers*; D.4.5 [**Operating Systems**]: Reliability—*Checkpoint/restart*

General Terms: checkpoint, petascale, exascale, phase-change memory, optimum checkpoint model

Additional Key Words and Phrases: hybrid checkpoint, in-memory checkpoint, in-disk checkpoint, incremental checkpoint, background checkpoint, checkpoint prototype

Extension of Conference Paper.
The conference paper is published in 2009 International Conference for High Performance Computing, Networking, Storage and Analysis with the title "Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems [Dong et al. 2009]." As an extension of the conference paper, this paper adds actual experiment data of hybrid checkpoint overhead obtained from self-developed prototype platforms and demonstrates how the proposed hybrid checkpointing scheme can revive incremental checkpoints and enable background checkpoints.

## 1. INTRODUCTION

MPP systems are designed to solve complex mathematical problems that are highly computation intensive and typically take many days to complete. Although the individual nodes in MPP systems are designed to have a high Mean-Time-to-Failure (MTTF), the reliability of the entire system degrades significantly as the number of nodes increases. One of the extreme examples is that the "ASCI Q" supercomputer at Los Alamos National Laboratories had an MTTF of less than 6.5 hours [Reed 2004]. This system reliability issue will be amplified in the future exascale era where the system will likely have five to ten times more nodes compared to today's petaFLOPS systems.

Checkpoint-restart is a classic fault-tolerance technique that helps large-scale computing systems recover from unexpected failures or scheduled maintenance. As the scale of future MPP systems keeps increasing and the system MTTF keeps decreasing, it is foreseeable that the checkpoint protection with higher frequency is required. However, the current state-of-the-art approach, which takes a snapshot of the entire memory image and stores it into a globally accessible storage (typically built with disk arrays), as shown in Fig. 1, is not a scalable approach and not feasible for the exascale system in the future. The scalability limitations are twofold. Firstly, the conventional storage device, such as the hard disk drive (HDD), is extremely hard to scale further due to physics limitations; secondly, storage modules used in modern MPP systems are designed to be separate from the main compute node, which ensures the robustness of the data storage but is inherently not scalable for checkpointing since the it limits the available bandwidth and causes compute nodes to compete for the global storage resource. Due to these reasons, lots of contemporary MPP systems have already experienced a non-negligible amount of performance loss when using the checkpoint-restart technique. Table I [Cappello 2009] lists the reported checkpoint time of some MPP systems, which clearly shows the checkpoint time can be as long as 30 minutes. As the application size grows along with the system scale, the poor scaling of the current approach will quickly increase the checkpoint time to several hours. As this trend continues, very soon the checkpoint time will surpass the failure period, which means the risk of ending up with an infinite execution time.

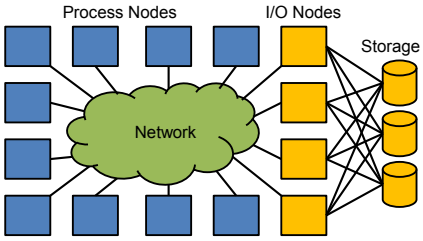


Fig. 1. The typical organization of the contemporary supercomputer. All the permanent storage devices are taken control by I/O nodes. There is no local permanent storage for each node.

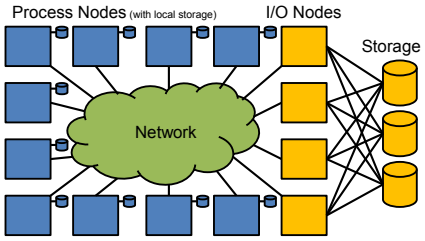


Fig. 2. The proposed new organization that supports hybrid checkpoints. The primary permanent storage devices are still connected through I/O nodes, but each process node also has a permanent storage.

Table I. Time to take a checkpoint on some machines of the Top500

| Systems | Max performance | Checkpoint time (minutes) |
|---|---|---|
| LLNL Zeus | 11 teraFLOPS | 26 |
| LLNL BlueGene/L | 500 teraFLOPS | 20 |
| Argonne BlueGene/P | 500 teraFLOPS | 30 |
| LANL RoadRunner | 1 petaFLOPS | $\sim 20$ |

Although the industry is actively looking at ways to reduce failure rates of computing systems, it is impractical to manufacture fail-safe components such as processor cores, memories, etc. Therefore, the only feasible solution is to design more efficient checkpointing schemes.

In this work, we leverage the emerging non-volatile memory technology like phase-change RAM (PCRAM) and propose a hybrid checkpointing scheme with both local and global checkpoints. The proposed PCRAM-based checkpointing scheme fully takes advantage of the PCRAM fast access property and keeps the checkpoint/restart technique still effective for the future exascale MPP systems[1]. A hybrid checkpointing performance model is established to evaluate the overhead of using this technique. It shows that the PCRAM-based hybrid checkpointing only incurs less than 3% performance loss on a projected exascale system. In addition, as a bonus effect, this new checkpointing scheme also boosts the effectiveness of incremental checkpointing and enables background global checkpointing, both of which further reduce the checkpoint overhead.

## 2. BACKGROUND

In this section, we first discuss the scalability issue of the conventional checkpointing mechanism and then give the background information on PCRAM, which is the key technology that enables low-cost hybrid checkpointing.

### 2.1 Scalability Issues of Checkpointing

Checkpoint-restart is the most widely-used technique to provide fault-tolerance for MPP systems. There are two main categories of checkpointing: *coordinated checkpointing* takes a consistent global checkpoint snapshot by flushing the in-transit messages and capturing the local state of each process node simultaneously; *uncoordinated checkpointing* reduces network congestion by letting each node take checkpoints at different time but maintaining all the exchanged messages among nodes a in log to reach a consistent checkpoint state. For large-scale applications, coordinated checkpointing is more popular due to its simplicity [Oldfield et al. 2007]. However, neither of them is a scalable approach. There are two primary obstacles that prevent performance scaling.

[1]Fault detection and silent data corruption is another significant problem by itself in the supercomputing community, and it is out of the scope of this work. However, it is still reasonable to assume that the time required to detect a failure is much less than the checkpoint interval, even in this work the interval might be as fast as 0.1 seconds. Therefore, we neglect the overhead caused by failure detection when we evaluate the performance of our approaches.

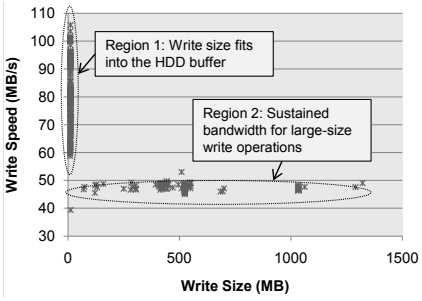


Fig. 3. The hard disk drive bandwidth with different write size.

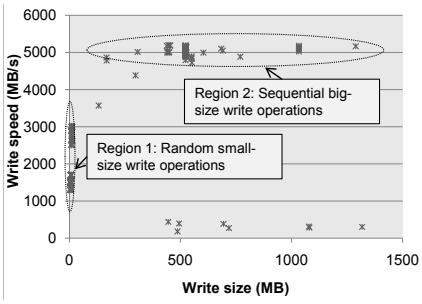


Fig. 4. The main memory bandwidth with different write size.

2.1.1 *Bottleneck 1: HDD Data Transfer Bandwidth.* As shown in Fig. 1, the in-practice checkpoint storage device is HDD, which implies that the most serious bottleneck of in-disk checkpointing is the sustained transfer rate of HDDs (<150MB/s). The significance of this problem is demonstrated by the fact that the I/O generated by HDD-based checkpointing consumes nearly 80% of the total file system usage even on today's MPP systems [Oldfield et al. 2007], and the checkpoint overhead accounts for over 25% of total application execution time in a petaFLOPS system [Grider et al. 2007]. Although a distributed file system, like *Lustre*, can aggregate the file system bandwidth to hundreds of GB/s, in such systems the checkpoint size also gets aggregated by the scale of nodes, nullifying the benefit.

As the HDD data transfer bandwidth is not easily scaled up due to its mechanical nature, it is necessary to change the future checkpoint storage from in-disk to in-memory. In order to quantify speed difference between the in-disk and in-memory checkpointing, we measure their peak sustainable speed using a hardware configuration with 2 Dual-Core AMD Opteron 2220 Processors, 16GB of ECC-protected registered DDR2-667 memory, and West Digital 740 hard disk drives operating at 10,000 RPM with a peak bandwidth of 150MB/s reported in the datasheet.

As a block device, the HDD has a large variation on its effective bandwidth depending upon the access pattern. In our system, although the data sheet reports a peak bandwidth of 150MB/s, the actual working bandwidth is much smaller. We measure the actual HDD bandwidth by randomly copying files with different sizes and use system clock to track the time spent. The result is plotted in Fig. 3, which shows all the points fall into two regions: one is near the y-axis, and the other is at the 50MB/s line. When the write size is relatively small, the effective write bandwidth of the HDD can be as high as 100MB/s and as low as 60MB/s depending on the status of the HDD internal buffer. However, it can be observed that when the write size is in megabyte scale, the effective write bandwidth of HDD drops dramatically and the actual value is 50MB/s, which is only one third of its peak bandwidth of 150MB/s.

On contrary, the result of in-memory checkpointing speed is shown in Fig. 4. Similar to the HDD bandwidth, all the collected data fall into two regions. However, unlike the relationship between the HDD bandwidth and write size, the attainable bandwidth is higher when the write size is large due to the benefit achieved from

spatial locality. This is desirable for checkpointing since checkpoints are usually large. In addition, the achievable bandwidth is very close to 5333MB/s, which is the theoretical peak bandwidth of the DDR2-667 memory used in this experiment. Therefore, compared to the in-disk checkpointing speed, the attainable in-memory speed can be two orders of magnitude faster.

In section 2.2, we discuss how to leverage the emerging PCRAM technology to implement the in-memory checkpointing.

2.1.2 *Bottleneck 2: Centralized Checkpoint Storage.* Another bottleneck of the current checkpointing system, as shown in Fig. 1, comes form the centralized checkpoint storage. Typically, several nodes in system are assigned to be the I/O nodes that are in charge of the HDD accesses. Thus, the checkpoints of each node (including computer nodes and I/O nodes) have to go through the I/O nodes via network connections before reaching their final destinations, which consumes a large part of the system I/O bandwidth and causes burst congestion. As the system scale keeps grows, the physical distance between the checkpoint sources and targets is increasing. Thereby, it only causes unacceptable performance, but also wastes lots of power consumption on data transfers.

To solve this bottleneck, later in this paper, we propose a hybrid checkpointing mechanism that uses both local and global checkpoints, in which the local checkpoint is fast and does not need any network connection while the global checkpoint is still preserved to provide the full fault coverage. The details of this hybrid checkpointing mechanism is discussed in Section 4.

## 2.2 Phase-Change Memory (PCRAM)

Recently, many emerging non-volatile memory technologies, such as as magnetic RAM (MRAM), ferroelectric RAM (FeRAM), and phase-change RAM (PCRAM), show their attractive features like the fast read access, high density, and non-volatility. Among these new memory technologies, PCRAM is considered to be the most promising one since compared to other emerging nonvolatile memories such as MRAM and FeRAM, PCRAM has excellent scalability, which is critical to the success of any emerging memory technologies. More importantly, as a non-volatile memory technology, it is highly feasible to use PCRAM as the hard disk substitution with much faster access speed.

2.2.1 *PCRAM Mechanism.* Unlike SRAM, DRAM or NAND flash technologies that use electrical charges, PCRAM changes the state of a Chalcogenide-based material, such as alloys of germanium, antimony, or tellurium ($GeSbTe$, or $GST$), to store a logical "0" or "1." For instance, GST can be switched between the crystalline phase (SET or "1" state) and the amorphous phase (RESET or "0" state) with the application of heat. The crystalline phase shows high optical reflectivity and low electrical resistivity, while the amorphous phase is characterized by low reflectivity and high resistivity. Due to these differences, phase-change materials can be used to build both memory chips and optical disks. As shown in Fig. 5, every PCRAM cell contains one GST and one access transistor. This structure has a name of "1T1R" where T refers to the access transistor, and R stands for the GST resistor.

To read the data stored in a PCRAM cell, a small voltage is applied across

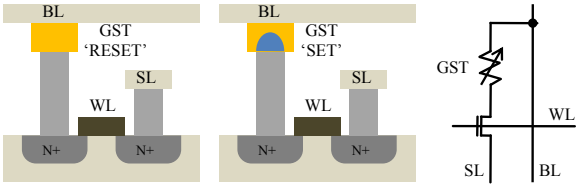


Fig. 5. The schematic view of a PCRAM cell with NMOS access transistor (BL=Bitline, WL=Wordline, SL=Sourceline).

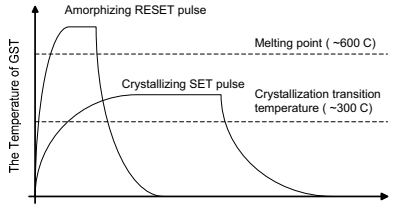


Fig. 6. The temperature-time relationship during SET and RESET operations.

the GST. Since the SET state and RESET state have a large variance on their equivalent resistances, data are sensed by measuring the pass-through current. The read voltage is set sufficiently high to invoke a sensible current but low enough to avoid write disturbance. Usually, the read voltage is clamped between $0.2V$ to $0.4V$ [Hanzawa et al. 2007]. Similar to traditional memories, the word line connected to the gate of the access transistor is activated to read values from PCRAM cells.

The PCRAM write operation is characterized by its SET and RESET operations. As illustrated in Fig. 6, the SET operation crystallizes GST by heating it above its crystallization temperature, and the RESET operation melt-quenches GST to make the material amorphous. The temperature during each operation is controlled by applying the appropriate current waveform. For SET operation, a moderate current pulse is applied for a longer duration to heat the cell above the GST crystallization temperature but below the melting temperature; for REST operation, a high power pulse heats the memory cell above the GST melting temperature. Recent PCRAM prototype chips demonstrate that the RESET latency can be as fast as $100ns$ and the peak SET current can be as low as $100\mu A$ [Pellizzer et al. 2004; Hanzawa et al. 2007].

The cell size of PCRAM is mainly constrained by the current driving ability of the NMOS access transistor. The achievable cell size can be as small as $10 - 40F^2$ [Pellizzer et al. 2004; Hanzawa et al. 2007], where $F$ is the feature size. When NMOS transistors are substituted by diodes, the PCRAM cell size can be reduced to $4F^2$ [Zhang et al. 2007]. Related research [Pirovano et al. 2003] shows PCRAM has excellent scalability as the required SET current can be reduced with technology scaling. Although multi-bit cell is available recently [Bedeschi et al. 2009], we use single-bit cell in this work for faster access.

Comparing to other storage technologies, such as SRAM, DRAM, NAND flash, and HDD, PCRAM shows its relatively good properties in terms of density, speed, power, and non-volatility. As listed in Table II, the PCRAM read speed is compara-

Table II. Comparison among SRAM, DRAM, NAND flash, HDD, and PCRAM.

| | SRAM | DRAM | NAND flash | PCRAM | HDD |
|---|---|---|---|---|---|
| Cell size | $> 100F^2$ | $6 - 8F^2$ | $4 - 6F^2$ | $4 - 40F^2$ | - |
| Read time | $\sim 10ns$ | $\sim 10ns$ | $5\mu s - 50\mu s$ | $10ns - 100ns$ | $\sim 4ms$ |
| Write time | $\sim 10ns$ | $\sim 10ns$ | $2 - 3ms$ | $100 - 1000ns$ | $\sim 4ms$ |
| Standby power | Cell leakage | Refresh power | Zero | Zero | $\sim 1W$ |
| Endurance | $10^{18}$ | $10^{15}$ | $10^5$ | $10^8 - 10^{12}$ | $10^{15}$ |
| Non-volatility | No | No | Yes | Yes | Yes |

ble to those of SRAM and DRAM. While its write operation is slower than SRAM and DRAM, it is still much faster than its non-volatile counterpart – NAND flash. More importantly, the PCRAM write endurance is within the feasible range for the checkpointing application. Pessimistically assuming the PCRAM write endurance of $10^8$ and checkpoint interval of $10s$, the lifetime of the PCRAM checkpointing module can still be more than 30 years, while the lifetime of its NAND flash counterpart is less than 30 hours. We expect the PCRAM write endurance will be higher than $10^{10}$ in 2017, so that an even more aggressive checkpoint interval, i.e. $0.1s$, would not be a problem for PCRAM lifetime.

## 3. INTEGRATING PCRAM MODULES INTO MPP SYSTEMS

PCRAM can be integrated into the computer system in the similar way to the traditional DRAM Dual-Inline Memory Modules (DIMMs). In this section, *PCRAM-DIMM* is proposed to integrate the PCRAM resources into MPP systems without much engineering effort. An in-house PCRAM simulation tool, called *PCRAM-sim* [Dong et al. 2009], is used to simulate the performance of this approach.

While some of the PCRAM prototypes show the PCRAM read latency is longer than $50ns$ [Pellizzer et al. 2004; Hanzawa et al. 2007; Zhang et al. 2007; Bedeschi et al. 2009], the read latency (from address decoding to data sensing) can be reduced to around $10ns$ by cutting PCRAM array bitlines and wordlines into small segments [Dong et al. 2009]. However, the PCRAM write latency reduction is limited by the long SET pulse ($\sim 100ns$), and in order to improve the write bandwidth, the data word width has to be increased. As a result, the conventional DRAM-DIMM organization cannot be directly adopted as each DRAM chip on the DIMM only has the word width of 8 bits, and thus the write bandwidth is only $0.08GB/s$, far below the DDR3-1333 bandwidth of $10.67GB/s$.

To solve the bandwidth mismatch between the DDRx bus and the PCRAM chip, two modifications are made to organize the new PCRAM-DIMM,

(1) As shown in Fig. 8, the configuration of each PCRAM chip is changed to x72 (64 bits of data and 8 bits of ECC protection), while the 8x prefetching scheme is retained for compatibility with the DDR3 protocol. As a result, there are $72 \times 8$ data latches in each PCRAM chip, and during each PCRAM write operation, 576 bits are written into the PCRAM cell array in parallel;

(2) The 18 chips on DIMMs are re-organized in an interleaved way. For each data transition, only one PCRAM chip is selected. A 18-to-1 data mux/demux is added on DIMMs to select the proper PCRAM chip for each DDR3 transition.

Consequently, the PCRAM write latency of each PCRAM chip can be overlapped. The overhead of this new DIMM organization includes: (1) one 1-to-18

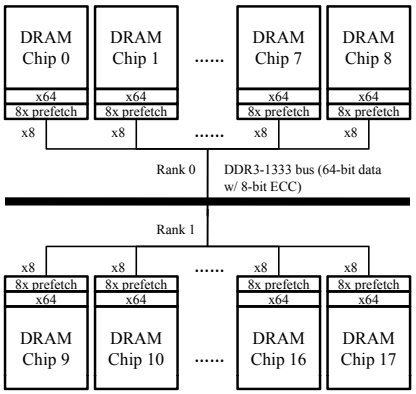


Fig. 7. The organization of a DRAM DIMM.

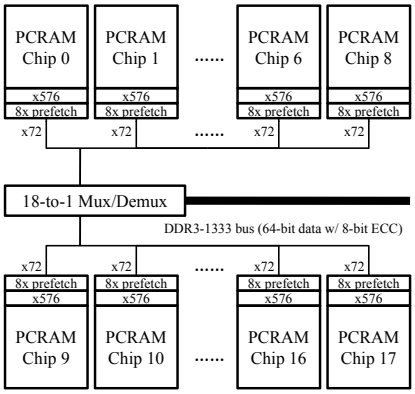


Fig. 8. The organization of the proposed PCRAM DIMM.

Table III. Different configurations of the PCRAM chips.

| Process | Capacity | # of Bank | Read/RESET/SET | Leakage | Die Area |
|---|---|---|---|---|---|
| 65nm | 512Mb | 4 | $27ns/55ns/115ns$ | $64.8mW$ | $109mm^2$ |
| 65nm | 512Mb | 8 | $19ns/48ns/108ns$ | $75.5mW$ | $126mm^2$ |
| 45nm | 1024Mb | 4 | $18ns/46ns/106ns$ | $60.8mW$ | $95mm^2$ |
| 45nm | 1024Mb | 8 | $16ns/46ns/106ns$ | $62.8mW$ | $105mm^2$ |

data mux/demux; (2) 576 sets of data latches, sense amplifiers, and write drivers on each PCRAM chip. The mux/demux can be implemented by a circuit that decodes the DDR3 address to 18 chip select signals (CS#). The overhead of data latches, sense amplifiers, and write drivers are evaluated using *PCRAMsim*.

Various configurations are evaluated by *PCRAMsim* and the results are listed in Table III.

Based primarily on SET latency and area efficiency, we use the 45nm 1024Mb 4-bank PCRAM chip design as a guide, and all the PCRAM-DIMM simulations in Section 6 are based on this configuration. Meanwhile, the write bandwidth of PCRAM-DIMM is $64bit \times 8 \times 18/106ns = 10.8GB/s$, which is compatible with the DDR3-1333 bandwidth $10.66GB/s$. In addition, according to our *PCRAM-sim* power model, for each 576-bit RESET and SET operation, it consumes total dynamic energy of $31.5nJ$ and $19.6nJ$, respectively. Therefore, assuming that "0" and "1" are written uniformly, the average dynamic energy is $25.6nJ$ per 512 bits, and the 1024Mb PCRAM-DIMM dynamic power under write operations is $25.6nJ/512b \times 10.8GB/s \approx 4.34W$. The leakage power of the 18-chip PCRAM-DIMM is estimated to be $60.8mW \times 18 = 1.1W$.

## 4. LOCAL/GLOBAL HYBRID CHECKPOINT

Integrating PCRAM into future MPP systems and using PCRAM as the fast in-memory checkpoint storage remove the first performance bottleneck, the slow HDD speed. However, the second bottleneck, the centralized I/O storage, still exists. To further remove this bottleneck, a hybrid checkpointing scheme with both local and global checkpoints is proposed. This scheme works efficiently as it is found that most of the system failures can be locally recovered without the involvement of

other nodes.

### 4.1 Motivations

Historically, most of the contemporary MPP systems use diskless nodes as it is easier to provide the reliability service (such as striped disks) at large scale instead of at each node. As a result, there is no local storage device and all the checkpoints are stored globally. While the centralized storage can be well-provisioned and maintained such that 24x7 availability is achieved, this solution is not scalable and the checkpointing overhead is too severe when the node count keeps increasing and all the nodes compete for the single resource at the same time. Another reason why local storage device is not provided at each node locally is due to the slow access speed of HDD, which is the mainstream storage today. Since the peak bandwidth of HDD (less than 200MB/s) is lower than the typical network bandwidth (e.g. 10Gb/s Ethernet bandwidth), diskless node accessing remote storage does not impacts the execution performance assuming that there is no competition for the network resources. However, if PCRAM-DIMM is deployed as the fast checkpoint storage device that can provide bandwidth of several tens of GB/s, the network bandwidth becomes the primacy bottleneck and severely degrades the checkpoint speed.

In summary, deploying PCRAM-DIMM into MPP nodes makes it valuable to include local checkpoints. Together with global checkpoints that ensure the overall system reliability, a local/global hybrid checkpoint scheme is promising for the future exascale MPP systems.

### 4.2 Hybrid Checkpoint Scheme

We propose local checkpoints that periodically backup the state of each node in their own private storage. Every node has a dedicated local storage for storing its system state. Similar to its global counterpart, the checkpointing is done in a coordinated fashion. We assume that a global checkpoint is made from an existing local checkpoint. Fig. 9 shows the conceptual view of the hybrid checkpoint scheme,

— *Step 1:* Each node dumps the memory image to their own local checkpoints;

— *Step 2:* After several local checkpoint interval, a global checkpoint is initiated, and the new global checkpoints are made from the latest local checkpoints;

— *Step 3:* When there is a failure but all the local checkpoints are accessible, the local checkpoints are loaded to restore the computation;

— *Step 4:* When there is a failure and parts of the local checkpoints are lost (in this case, Node 3 is lost), the global checkpoints (which might be obsolete compared to the latest local checkpoints) are loaded, and the failure node is substituted by a backup node.

This two-level hybrid checkpointing gives us an opportunity to tune the local to global checkpoint ratio based on failure types. For example, a system with high transient failures can be protected by frequent local checkpoints and a limited number of expensive global checkpoints without losing performance. The proposed local/global checkpointing is also effective in handling failures during the checkpoint operation. Since the scheme does not allow concurrent local and global checkpointing, there will always be a stable state for the system to rollback even when a failure

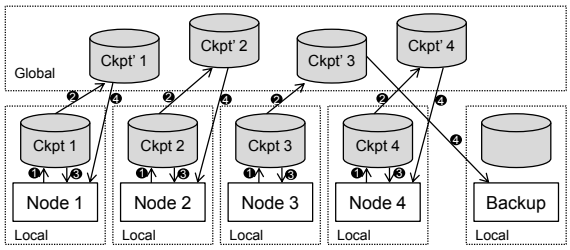


Fig. 9. The local/global hybrid checkpoint model.

occurs during the checkpointing process. The only time the rollback operation is not possible is when a node fails completely in the middle of making a global checkpoint. While such failure events can be handled by maintaining multiple global copies, the probability of a global failure in the middle of a global checkpoint is less than 1%. Hence, we limit our proposal to a single copy of local and global checkpoints. Whether the MPP system can be recovered using a local checkpoint after a failure depends on the failure type. In this work, all the system failures are divided into two categories:

—*Failures that can be recovered by local checkpoints:* In this case, the local checkpoint in the failure node is still accessible. If the system error is a transient one, (i.e., soft error, accidental human operation, or software bug), the MPP system can be simply recovered by rebooting the failure node using its local checkpoint. If the system error is due to a software bug or hot plug/unplug, the MPP system can also be recovered by simply rebooting or migrating the computation task from one node to another node using local checkpoints.

—*Failures that have to be recovered by global checkpoints:* In the event of some permanent failures, the local checkpoint in the failed node is not accessible any more. For example, if the CPU, the I/O controller, or the local storage itself fails to work, the local checkpoint information will be lost. This sort of failure has to be protected by a global checkpoint, which requires storing system state in either neighboring nodes or a global storage medium.

As a hierarchical approach, whenever the system fails, the system will first try to recover from local checkpoints. If one of the local checkpoints is not accessible, the system recovery mechanism will restart from the global checkpoint.

### 4.3 System Failure Category Analysis

The effectiveness of the proposed local/global hybrid checkpointing depends on how much failure can be recovered locally. A thorough analysis of failure rates of MPP systems shows that a majority of failures are transient in nature [Michalak et al. 2005] and can be recovered by a simple reboot operation. In order to quantitatively learn the failure distribution, we studied the failure events collected by the Los Alamos National Laboratory (LANL) during 1996-2005 [Los Alamos National Laboratory 2009]. The data covers 22 high-performance computing systems, including a total of 4,750 machines and 24,101 processors. The statistics of the failure root cause are shown in Table IV.

Table IV. The statistics of the failure root cause collected by LANL during 1996-2005.

| Cause | Occurrence | Percentage |
|---|---|---|
| Hardware | 14341 | 60.4% |
| Software | 5361 | 22.6% |
| Network | 421 | 1.8% |
| Human | 149 | 0.6% |
| Facilities | 362 | 1.5% |
| Undetermined | 3105 | 13.1% |
| Total | 23739 | 100% |

We conservatively assume that *undetermined failures* have to rely on global checkpoints for recovery, and assume that the failures caused by *software*, *network*, *human*, and *facilities* can be protected by local checkpoints:

— If nodes halt due to *software failures* or *human mal-manipulation*, we assume some mechanisms (i.e., timeout) can detect these failures and the failure node will be rebooted automatically.

— If nodes halt due to *network failures* (i.e., widely-spread network congestion) or *facilities downtime* (i.e. global power outrage), automatic recovery is impossible and manual diagnose/repair time is inevitable. However, after resolving the problem, the system can simply restart using local checkpointing.

The remaining *hardware failure* accounts to more than 60% of total failures. However, according to research on the fatal soft error rate of the "ASCI Q" system at LANL in 2004 [Michalak et al. 2005], it is estimated that about 64% of the hardware failures are attributed to soft errors. Hence, observing the failure trace, we have the following statistics: $60.4\% \times 64\% = 38.7\%$ soft errors, and $60.4\% \times (1 - 64\%) = 21.7\%$ hard errors. As soft errors are transient and it is highly possible that the same error would not happen again after the system is restored from the latest checkpoint, local checkpoints are capable of covering all the soft errors. However, hard errors usually mean there is permanent damage to the failure node and the node should be replaced. In this case, the local checkpoint stored on the failure node is lost as well, hence only global checkpoints can protect the system from hard errors. As a result, in total, we estimate that 65.2% of failure can be corrected by local checkpoints and only 34.8% of failure needs global checkpoints.

Further considering the soft error rate (SER) will greatly increase as the device size shrinks, we project that SER increased 4 times from 2004 to 2008. Therefore, we make a further estimation for the petaFLOPS system in 2008 that 83.9% of failures need local checkpoints and only 16.1% failures need global ones. This failure distribution biased to local errors provides a significant opportunity for the local/global hybrid checkpointing scheme to reduce the overhead as we show in Section 6. Since the soft error rate is critical to the effectiveness of the hybrid checkpointing, a detailed sensitivity study on SER is also demonstrated in Section 6.6.

### 4.4 Theoretical Performance Model

In an MPP system with checkpointing, the optimal checkpoint frequency is a function of both failure rates and checkpoint overhead. A low checkpoint frequency reduces the impact of checkpoint overhead on performance but loses more useful

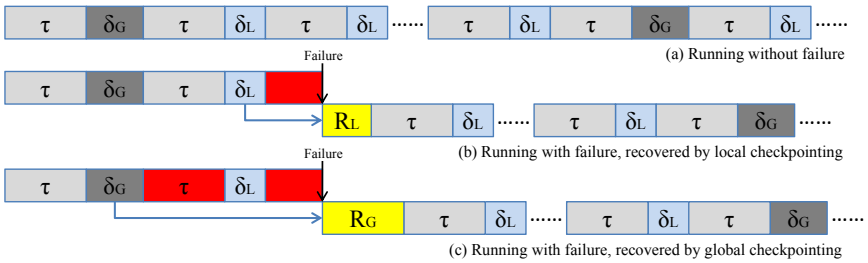


Fig. 10. A conceptual view of execution time broken by the checkpoint interval: (a) an application running without failure; (b) an application running with a failure, where the system rewinds back to the most recent checkpoint, and it is recovered by the local checkpoint; (c) an application running with a failure that cannot be protected by the local checkpoint. Hence, the system rewinds back to the most recent global checkpoint. The red block shows the computation time wasted during the system recovery.

Table V. Local/Global hybrid checkpointing parameters.

| | |
|---|---|
| $T_S$ | The original computation time of a workload |
| $p_L$ | The percentage of local checkpoints |
| $p_G$ | $1 - p_L$, the percentage of global checkpoints |
| $\tau$ | The local checkpoint interval |
| $\delta_L$ | The local checkpoint overhead (dumping time) |
| $\delta_G$ | The global checkpoint overhead (dumping time) |
| $\delta_{eq}$ | the equivalent checkpoint overhead in general |
| $R_L$ | The local checkpoint recovery time |
| $R_G$ | The global checkpoint recovery time |
| $R_{eq}$ | The equivalent checkpoint time in general |
| $q_L$ | The percentage of failure covered by local checkpoints |
| $q_G$ | $1 - q_L$, the percentage of failure that have to be covered by global checkpoints |
| $MTTF$ | The system mean time to failure, modeled as 5 year/number of nodes |
| $T_{total}$ | The total execution time including all the overhead |

work when failures take place, and vice versa. Young [Young 1974] and Daly [Daly 2006] derived expressions to determine the optimal checkpoint frequency that strikes the right balance between the checkpoint overhead and the amount of useful work lost during failures. However, their models do not support local/global hybrid checkpointing. In this work, we extend Daly's work [Daly 2006] and derive a new model to calculate the optimal checkpoint frequencies for both local and global checkpoints.

Let us consider a scenario with the following parameters as listed in Table V and divide the total execution time of a checkpointed workload, $T_{total}$, into four parts:

$$T_{total} = T_S + T_{dump} + T_{rollback,recovery} + T_{extra-rollback} \quad (1)$$

where $T_S$ is the original computation time of a workload, $T_{dump}$ is the time spent on checkpointing, $T_{rollback,recovery}$ is the recovery cost when a failure occurs (no matter it is local or global), and $T_{extra-rollback}$ is the extra cost to discard more useful work when a global failure occurs.

The checkpoint dumping time is simply the product of the number of checkpoints,

$T_S/\tau$, and the equivalent dumping time per checkpoint, $\delta_{eq}$, thus

$$T_{dump} = \frac{T_S}{\tau}(\delta_{eq}) \tag{2}$$

where

$$\delta_{eq} = \delta_L \cdot p_L + \delta_G \cdot p_G \tag{3}$$

and the parameters $\delta_L$ and $\delta_G$ are determined by the checkpoint size, local checkpoint bandwidth, and global checkpoint bandwidth.

When failure occurs, at least one useful work slot has to be discarded as the red slot shown in Fig. 10(b) and the second red slot shown in Fig. 10(c). Together with the recovery time, this part of overhead can be modeled as follows with the approximation that the failure occurs half way through the compute interval on average,

$$T_{rollback,recovery} = \left(\frac{1}{2}(\tau + \delta_{eq}) + R_{eq}\right)\frac{T_{total}}{MTTF} \tag{4}$$

where $T_{total}/MTTF$ is the expected number of failure and the average recovery time $R_{eq}$ is expressed as

$$R_{eq} = R_L \cdot q_L + R_G \cdot q_G \tag{5}$$

and the recovery time $R_L$ and $R_G$ are equal to the checkpoint dumping time (in a reversed direction) $\delta_L$ and $\delta_G$ plus the system rebooting time. Here, $q_L$ and $q_G$ are the percentage of the failure recovered by local and global checkpoints, respectively, and their values are determined in the same way as described in Section 4.3 at different system scales.

Additionally, if a failure has to rely on global checkpoints, more useful computation slots will be discarded as the first red slot shown in Fig. 10(c). In this case, as the average number of local checkpoints between two global checkpoints is $p_L/p_G$, the number of wasted computation slots, on average, is approximated to $p_L/2p_G$. For example, if $p_L = 80\%$ and $p_G = 20\%$, there are $80\%/20\% = 4$ local checkpoints between two global checkpoints, and the expected number of wasted computation slots is $p_L/2p_G = 2$. Hence, this extra rollback cost can be modeled as follows,

$$T_{extra-rollback} = \frac{p_L q_G}{2p_G}(\tau + \delta_L)\frac{T_{total}}{MTTF} \tag{6}$$

Eventually, after including all the overhead mentioned above, the total execution time of a checkpointed workload is,

$$\begin{aligned} T_{total} \;=\; & T_S + \frac{T_S}{\tau}(\delta_{eq}) + \left(\frac{1}{2}(\tau + \delta_{eq}) + R_{eq}\right)\frac{T_{total}}{MTTF} \\ & + \frac{p_L q_G}{2p_G}(\tau + \delta_L)\frac{T_{toal}}{MTTF} \end{aligned} \tag{7}$$

It can be observed from the equation that a trade-off exists between the checkpoint frequency and the rollback time. Since many variables in the equation have strict lower bounds and can take only discrete values, we use MATLAB to optimize the two critical parameters, $\tau$ and $p_L$, using a numerical method. It is also feasible to derive closed-form expressions for $\tau$ and $p_L$ to enable run-time adjustment for

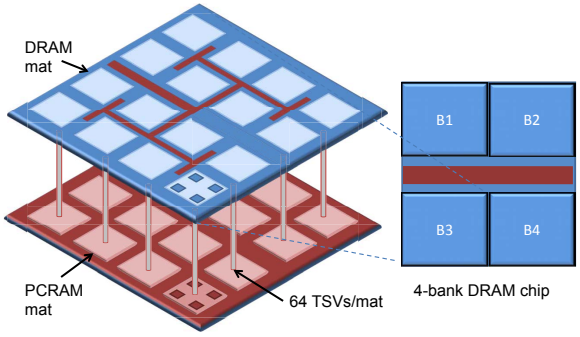


Fig. 11. A conceptual view of 3D-PCRAM: the DRAM module is stacked on top of the PCRAM module.

any changes of workload size and failure distribution, but they are out of the scope of this paper. A detailed analysis on checkpoint interval and local/global ratio under different MPP system configurations is discussed in Section 6.

## 5. ORTHOGONAL TECHNIQUES

The PCRAM hybrid local/global checkpointing scheme is not only an approach to solve the scalability issue of future exascale systems by itself, but also provides the extendability to be combined with other techniques.

### 5.1 3D-PCRAM: Deploying PCRAM atop DRAM

The aforementioned PCRAM-DIMM scheme still has performance limitations: copying from DRAM to PCRAM has to go through the processor and the DDR bus; it not only pollutes the on-chip cache but also has the DDR bandwidth constraint.

As the ultimate way to integrate PCRAM in a more scalable way, 3D-PCRAM scheme is proposed to deploy PCRAM directly atop DRAM. By exploiting emerging 3D integration technology [Xie et al. 2006] to design the 3D PCRAM/DRAM chip, it becomes possible to dramatically accelerate the checkpoint latency and hence reduce the checkpoint overhead to the point where it is almost a negligible percentage of program execution.

For backward-compatibility, the interface between DRAM chips and DIMMs is preserved. The 3D-PCRAM design has four key requirements:

(1) The new model should incur minimum modifications to the DRAM die, while exploiting 3D integration to provide maximum bandwidth between PCRAM and DRAM;

(2) We need extra logic to trigger the data movement from DRAM to PCRAM only when the checkpoint operation is needed and only where the DRAM bits are dirty;

(3) We need a mechanism to provide the sharp rise in supply current during PCRAM checkpointing; and

(4) There should be an effective way to transfer the contents of DRAM to PCRAM without exceeding the thermal envelope of the chip.

Table VI. 3D stacked PCRAM/DRAM memory statistics and the comparison between 3D-PCRAM and PCRAM-DIMM.

| | |
|---|---|
| Bank size | $32MB$ |
| Mat count | 16 |
| Required TSV pitch | $< 74\mu m$ |
| ITRS TSV pitch projection for 2012 | $3.8\mu m$ |
| 3D-PCRAM delay (independent of memory size) | $0.8ms$ |
| PCRAM-DIMM delay (2GB memory) | $185ms$ |
| 3D-PCRAM bandwidth (2GB DIMM) | $2500GB/s$ |
| PCRAM-DIMM bandwidth | $10.8GB/s$ |

Table VII. Temperature estimations of 3D-PCRAM modules.

| Scenario | Local checkpoint interval | Package temperature |
|---|---|---|
| DRAM Only | - | 319.17K |
| 1-Layer PCRAM stacked | 1.00s | 319.57K |
| 1-Layer PCRAM stacked | 0.10s | 320.54K |
| 1-Layer PCRAM stacked | 0.01s | 330.96K |

These four challenges are solved individually as follows:

(1) To reduce the complexity of the 3D stacked design, we use the same number of banks in the PCRAM and DRAM dies. Since the diode-accessed PCRAM cell size is similar to that of DRAM, we can model PCRAM banks of similar size to its DRAM counterpart. When making connections between dies, for the ultimate bandwidth, a cell-to-cell connection is desired. However, such a design needs very high density Through-Silicon-Vias (TSVs) and hence has low area efficiency. Thus, we opt for connections at the granularity of *mats*. A *mat* is a self-contained module with a set of memory cells and logic capable of storing or retrieving data (in *PCRAMsim*, a mat is composed of four sub-arrays). For the proposed 3D design, we make connections between the input bus of a mat in the DRAM to the corresponding mat in the PCRAM as shown in Fig. 11. Assuming a typical bank has 16 mats, we calculate that the required TSV pitch is less than $74\mu m$. ITRS [International Technology Roadmap for Semiconductors ] shows the achievable TSV density is about $3.8\mu m$ that far exceeds our requirements. Table VI shows the detailed specifications.

(2) To control the data transfer from DRAM to PCRAM, we include an address generator circuit and a multiplexer for each DRAM mat. An address generator is essentially a counter which retrieves the contents of a DRAM mat and sends it to its PCRAM counterpart when triggered. To hide the high write penalty of PCRAM, we use the multiplexer to interleave the writes between four sub-arrays in the PCRAM mat. To employ an incremental checkpointing technique, dirty page management is required for every page in the DRAM. This only costs 1-bit of overhead for each page, and avoids unnecessary transfers from DRAM to PCRAM.

(3) Although high-density TSVs can provide ultra-wide bandwidth as high as 2.5TB/s in our demonstration, an ultra-high peak current is also needed for parallel PCRAM cell writes. In such a case, the transient power consumption can be as high as 700W. However, this peak power is only required within an extremely short interval of 0.8ms and the actual energy consumption is as low as 0.56J. To handle this short period of power consumption, we include a super capacitor (about 0.6F)

on each 3D PCRAM/DRAM DIMM.

(4) To confirm that our 3D-PCRAM scheme will not cause thermal problems, we evaluated the impact of heat from 3D stacked PCRAM memory on the DRAM DIMMs. We obtain the estimated temperature listed in Table VII using HotSpot[Huang et al. 2008]. Note that the increase in temperature is negligible as long as the checkpoint interval is longer than 0.1s. Hence, for all our experiments (Section 6), we set the lower bound of local checkpoint interval to be 0.1 seconds.

### 5.2 Redundant Bit Suppression

As PCRAM write operations are energy-expensive and cause cell to wear out, it is better to write as few bits as possible. Fortunately, it is obvious that there is lots of redundancy between two successive full checkpoints, and using conditional write can eliminate the unnecessary bit flips. Removing the redundant bit-write can be implemented by preceding a write with a read. In PCRAM operations, reads are much faster than writes, so the delay increase here is trivial. The comparison logic can be simply implemented by adding an XNOR gate on the write path of a cell [Zhou et al. 2009].

### 5.3 Background Global Checkpointing

The existence of local checkpoints in the hybrid scheme makes it possible to overlap global checkpointing with program execution. Later in Section 6, we see there are multiple local checkpoint operations between two global checkpoints. Based on this property, the source of global checkpoints can be no longer the actually memory image of each node, but the latest local checkpoint. In this way, even the global checkpointing is slower (as it needs global network communication), the global checkpoint operation can be conducted in background and does not halt the program execution any more.

In order to find whether background checkpointing can effectively hide latency, we developed a prototype platform by modifying existing *Berkeley Labs Checkpoint/Restart* (BLCR) [Duell et al. 2002] and OpenMPI solutions. As PCRAM is not yet available to the commercial market, we use half of the DRAM main memory space to be the local checkpoint storage. This device emulation is reasonable since the future PCRAM can be also be mounted on a Dual-Inline Memory Module (DIMM). As mentioned in Section 0**??**, data on PCRAM DIMM can be interleaved across PCRAM chips so that write operations can be performed at the same rate as DRAM without any stalls [Dong et al. 2009]. The BLCR kernel is modified to add "dump to memory" feature. We modify **uwrite** kernel function that is responsible for BLCR to enable memory-based checkpointing. As BLCR library is an independent module which merely controls the program execution, it can directly execute existing MPI application binaries without any changes to the source code. We further extend the kernel function to track and log the overhead of checkpointing overhead. The overhead of each checkpoint-to-memory operation is measured by: 1. *kmalloc* that allocates memory; 2. *memcpy* that copies data to the newly-allocated memory space; 3. *free* the allocated memory. However, in Linux 2.6 kernel, *kmalloc* has a size limit of 128K, thus each actual memory-based checkpoint operation is divided into many small ones. This constraint slightly impacts on the memory write efficiency.

Table VIII. Execution time of a 1-thread program without global checkpointing, with global checkpointing, and with background global checkpointing. (unit: Second)

| | 1 | 2 | 3 | 4 | 5 | 6 | Average |
|---|---|---|---|---|---|---|---|
| w/o checkpointing | 6.24 | 6.29 | 6.34 | 6.33 | 6.33 | 6.32 | 6.31±0.0014 |
| w/ foreground checkpointing | 9.18 | 9.69 | 7.03 | 7.03 | 6.99 | 7.03 | 7.83±1.58 |
| w/ background checkpointing | 6.36 | 6.35 | 6.36 | 6.37 | 6.22 | 6.39 | 6.34±0.0037 |

Table IX. Execution time of a 2-thread program without global checkpointing, with global checkpointing, and with background global checkpointing. (unit: Second)

| | 1 | 2 | 3 | 4 | 5 | 6 | Average |
|---|---|---|---|---|---|---|---|
| w/o checkpointing | 18.15 | 18.08 | 21.80 | 18.17 | 18.88 | 17.99 | 18.85±2.20 |
| w/ foreground checkpointing | 25.40 | 24.85 | 24.97 | 23.25 | 21.05 | 22.46 | 23.66±2.92 |
| w/ background checkpointing | 18.41 | 23.69 | 21.90 | 18.44 | 18.33 | 18.32 | 19.84±5.53 |

Table X. Execution time of a 4-thread program without global checkpointing, with global checkpointing, and with background global checkpointing. (unit: Second)

| | 1 | 2 | 3 | 4 | 5 | 6 | Average |
|---|---|---|---|---|---|---|---|
| w/o checkpointing | 14.15 | 14.11 | 14.31 | 14.10 | 14.15 | 13.34 | 14.03±0.12 |
| w/ foreground checkpointing | 20.03 | 16.78 | 17.02 | 17.56 | 19.65 | 18.67 | 18.29±1.89 |
| w/ background checkpointing | 19.10 | 22.46 | 20.47 | 19.87 | 18.82 | 19.58 | 20.05±1.73 |

By using this prototype platform, we studied the following three scenarios:

(1) *Without checkpointing:* The program is executed without triggering any checkpointing activities. This is the actual execution time of the program.

(2) *With foreground checkpointing:* The program is executed with checkpoint enabled. Every checkpointing operation stalls the program, and takes snapshots into **HDD** directly.

(3) *With background checkpointing:* The program is executed with checkpoint enabled. Every checkpointing operation stalls the program, takes snapshots into **memory**, and then copies them to HDD in the background.

While background checkpointing stalls the program to make local checkpoints, the overhead is significantly small due to the low DDR latency compared to HDD or network latencies. The background checkpointing makes it feasible to overlap the slow in-HDD global checkpoint process with program execution. In this experiment, the in-memory local checkpoint is implemented by *ramfs*, which mounts a portion of main memory as a file system. To study the impact of the number of involved cores on background checkpointing, 1-thread, 2-thread, and 4-thread applications are run in a quad-core processor, respectively[2]. The results are listed in Table VIII to Table X, which show the total execution time with a single checkpointing operation performed in the middle of the program. Each configuration is run multiple times and the average value is considered for the evaluation.

We observe from the results that:

— The foreground checkpointing always takes about 25% performance loss due to low HDD bandwidth and this value is consistent with previous analytical evaluation [Oldfield et al. 2007].

[2]3-thread application is not included in the experiment setting because some benchmark only allow radix-2 task partitioning.

— When main memory is used for taking checkpoints, the checkpoint overhead for a 1-thread application is around 0.5% (as listed in Table VIII. This overhead is 50 times smaller than the foreground case, and it is consistent with our previous finding that in-memory checkpointing is 50 times faster than in-HDD checkpointing.).

— The background checkpoint overhead increases from 0.5% to 5% when the application to be checkpointed becomes multi-thread. This is because of conflicts in row buffer due to interleaving of workload accesses with checkpointing. In addition, the MPI synchronization overhead is another source of the extra latency, since our checkpointing scheme is *coordinated*.

— The background checkpointing becomes ineffective when the number of threads equals to the number of available processor cores. Its associated overhead is even larger than the foreground case. It is because in that case there is no spare processor core to handle the I/O operation generated by the background checkpointing activity.

Therefore, as along as designers ensure that spare processor units are available on each node when partitioning computation tasks, the background checkpointing technique is a strong tool to hide the global checkpoint latency.

### 5.4 Incremental Checkpointing

Since the in-memory checkpointing makes it possible to take checkpoints every few seconds, it reduces the overhead of incremental checkpointing. As the checkpoint interval decreases, the probability of polluting a clean page becomes smaller, hence, the average size of an incremental checkpoint decreases.

To measure the size difference between full checkpoints and incremental checkpoints, we developed another prototype platform since the BLCR+OpenMPI solution does not inherently support incremental checkpointing. The prototype consists of two parts: a primary thread that launches the target application and manages checkpoint intervals; a checkpoint library to be called by application. A running shell spawns a new process to run the application that requires checkpointing. After that the shell periodically sends **SIGUSR1** signal to the application. The **SIGUSR1** signal handler is registered as a function to store checkpoints to hard disk or main memory. This approach requires modification to the source code although the changes are limited to a couple of lines to invoke the handler. The incremental checkpoint feature is implemented using the bookkeeping technique. After taking a checkpoint, all the writable pages are marked as read-only using *mprotect* system call. When a page is overwritten, a page fault exception occurs, sends the **SIGSEGV** signal, and the page fault exception handler saves the address of the page in an external data structure. The page fault signal handler also marks the accessed page as writable by using *unprotect* system call. At the end of the checkpoint interval it is only necessary to scan the data structure that tracks the dirty pages. In this prototype, register file and data in main memory are considered as the major components of a whole checkpoint. Other components, such as pending signal and file descriptor, are not stored during the checkpointing operation because their attendant overhead can be ignored.

By using this prototype platform, we trigger checkpoint operations with the inter-

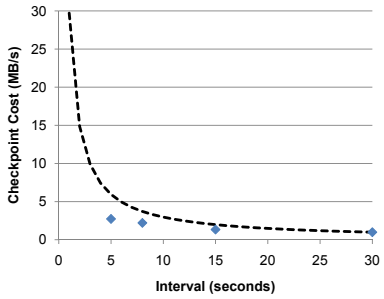


Fig. 12. Incremental checkpoint size (dot) and full checkpoint size (line) of CG.B

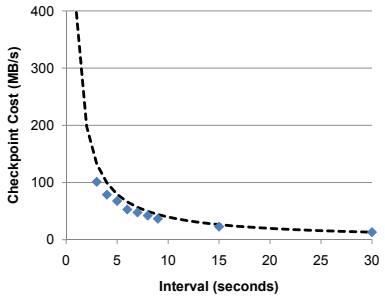


Fig. 13. Incremental checkpoint size (dot) and full checkpoint size (line) of MG.C

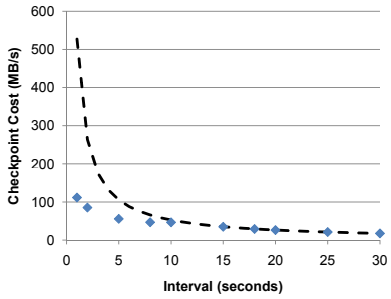


Fig. 14. Incremental checkpoint size (dot) and full checkpoint size (line) of BT.C

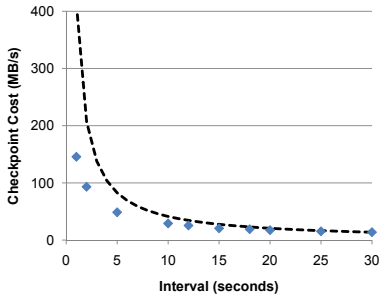


Fig. 15. Incremental checkpoint size (dot) and full checkpoint size (line) of UA.C

val ranging from 1 second to 30 seconds. Five workloads from the NPB benchmark with CLASS B and CLASS C configurations are tested. In order to have a fair comparison, a new metric, *checkpoint size per second*, is used to quantify the timing cost of checkpointing by assuming the checkpointing bandwidth is stable during the process. Fig. 12 to Fig. 15 show the checkpoint size of both schemes for different intervals.

It can be observed that, in all the five workloads, incremental checkpoint size is almost the same as the full checkpoint size when the checkpoint interval is greater than 20 seconds. This shows that incremental checkpointing scheme is not effective when the interval is not sufficiently small. Hence, checkpointing process that involves accessing HDD or network transfers cannot benefit from incremental checkpointing. This could be the reason why the most popular checkpoint library, BLCR [Duell et al. 2002], does not support incremental checkpointing. As interval size goes down, all the workloads except MG.C show a large reduction in checkpoint cost with incremental checkpointing. Based on this observation, it is clear that frequent checkpointing, which is only possible by using PCRAM-based checkpointing, is critical to benefit from incremental checkpointing schemes.

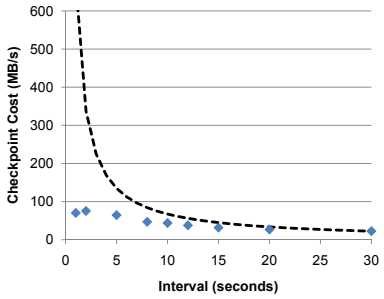


Fig. 16. Incremental checkpoint size (dot) and full checkpoint size (line) of IS.C

## 6. EXPERIMENT METHODOLOGY

The primary goal of this work is to improve the checkpoint efficiency and prevent checkpointing from becoming the bottleneck to MPP scalability. In this section, the analytical equations derived in Section 4.4 is mainly used to estimate the checkpoint overhead. In addition, simulations are also conducted to get the quantitative parameters such as the checkpoint size.

### 6.1 Checkpointing Scenarios

In order to show how the proposed local/global hybrid checkpoint using PCRAM can reduce the performance and power overhead of checkpoint operations, we study the following 4 scenarios:

— *Pure-HDD*: The conventional checkpoint approach that only stores checkpoints in HDD globally.

— *DIMM+HDD*: Store checkpoints in PCRAM DIMM locally and in HDD globally. In each node, the PCRAM DIMM capacity is equal to the DRAM DIMM capacity.

— *DIMM+DIMM*: Store local checkpoints in PCRAM DIMM and store neighbors' checkpoints in another in-node PCRAM DIMM as the global checkpoints. In each node, the PCRAM DIMM capacity is thrice as the DRAM DIMM capacity: one copy for the latest local checkpoint, one copy for the global checkpoint that stores the neighbor's local checkpoint, and one copy for the global checkpoint that stores own local checkpoint with the same time stamp as the global checkpoint.

— *3D+3D*: Same as *DIMM+DIMM*, but deploy the PCRAM resource using 3D-PCRAM rather than PCRAM-DIMM.

The bottleneck of each scenario is listed in Table XI.

### 6.2 Scaling Methodology

We use the specification of the IBM Roadrunner Supercomputer [Grider et al. 2007], achieving a sustained performance of 1.026 petaFLOPS on LINPACK, to model the petaFLOPS baseline MPP system.

**Socket Count:** Roadrunner has a total of 19,872 processor sockets and achieves an average of 52 gigaFLOPS per socket. We assume that the future processors can scale their performance with future increases in transistor count to 10 teraFLOPS

Table XI. Bottleneck factor of different checkpoint schemes.

| | Local medium | Local bottleneck |
|---|---|---|
| Pure-HDD | - | - |
| DIMM+HDD | Self's PCRAM DIMM | Memory bandwidth |
| DIMM+DIMM | Self's PCRAM DIMM | Memory bandwidth |
| 3D+3D | Self's 3D DIMM | 3D bandwidth |
| | Global Medium | Global Bottleneck |
| Pure-HDD | HDD on I/O nodes | HDD, Network bandwidth |
| DIMM+HDD | HDD on I/O nodes | HDD, Network bandwidth |
| DIMM+DIMM | Neighbor's PCRAM DIMM | Network bandwidth |
| 3D+3D | Neighbor's 3D DIMM | Network bandwidth |

Table XII. The specification of the baseline petascale system and the projected exascale system.

| | 1 petaFLOPS | 1 exaFLOPS |
|---|---|---|
| FLOPS | $10^{15}$ | $10^{18}$ |
| Year | 2008 | 2017 |
| # of sockets | 20,000 | 100,000 |
| Compute/IO node ratio | 15:1 | 15:1 |
| Memory per socket | 4GB | 210GB |
| Phase-change memory BW | 10GB/s | 32GB/s |
| Network BW | 3.5GB/s | 400GB/s |
| Aggregate file system BW | 220GB/s | 1600GB/s |
| Normalized SER | 1 | 32 |
| Transient error percentage | 91.5% | 99.7% |

per socket by the year 2017 [Vantrease et al. 2008]. Hence, to cross the exaFLOPS barrier, it is necessary to increase the socket count by 5X (from 20,000 to 100,000). This implies that the number of failures in exascale MPP systems will increase by at least 5X even under the assumption that the future 10-teraFLOPS socket retains the same MTTF as today.

**Memory per Socket:** The memory requirement of future MPP systems is proportional to the computational capabilities of the projected processor. Typical MPP workloads that solve various non-linear equations can adjust the scheduling granularity and thread size to suit the configuration of a processor. Therefore, as the computing power of a processor scales from 52 gigaFLOPS to 10 teraFLOPS, the application memory footprint in each processor will also increase. In general, the memory capacity required per socketis proportional to $(FLOPS)^{3/4}$ [3]. The current generation Roadrunner employs 4GB per Cell processor. Based on the above relation, a future socket with 10-teraFLOPS capability will require 210 GB of memory.

**Phase-Change Memory Bandwidth:** Both DRAM main memory access time and PCRAM DIMM checkpoint time are constrained by the memory bus bandwidth. The last decade has seen roughly a 3X increase in memory bandwidth because of the increased bus frequency and the prefetch depth. However, it is not clear whether similar improvements are possible in the next ten years. Preliminary DDR4 projections for the year 2012 show a peak bandwidth of 16GB/s. For

[3]Consider most MPP systems are used to solve differential equations and other numerical method problems, the required FLOPS scales up with 3 spacial dimensions and 1 temporal dimension, but the required memory size only scales up with 3 spatial dimensions.

Table XIII. Memory usage of NPB suite.(Unit: Percentage of the memory capacity)

| Workload | Memory Usage | Workload | Memory Usage |
|---|---|---|---|
| BT.C | 16.8% | CG.C | 21.7% |
| DC.B | 25.0% | EP.C | 0.1% |
| FT.B | 100% | IS.C | 25.0% |
| LU.C | 14.6% | MG.C | 82.4% |
| SP.C | 17.7% | UA.C | 11.4% |

our projected exaFLOPS system in 2017, we optimistically assume a memory bus bandwidth of 32GB/s. Nevertheless, note that the 3D-PCRAM checkpointing is not limited by memory bandwidth as mentioned in Section 5.1.

**Network Bandwidth and :** As electrical signals become increasingly difficult at high data rates, optical data transmission is a necessary part of the exascale system. We assume the network bandwidth is scaled from 12GB/s to 400PB/s by using optical interconnects [Kash 2009].

**Aggregate File System Bandwidth:** The HDD-based file system bandwidth is assumed to be scaled from 220GB/s (the specification of IBM Roadrunner) to 1.6TB/s (proposed in ClusterStor's Colibri system).

**Soft Error Rate (SER) and System MTTF:** The failure statistics of Roadrunner are not available yet in the literature, and the accurate projection of overall MTTF for future processors is beyond the scope of this paper. In this work, we simply assume the *hard error rate (HER)* and *other error (i.e. software bug) rate (OER)* remain constant, and only consider the scaling of *soft errors*. A study from Intel [Borkar 2005] shows that when moving from 90nm to 16nm technology the soft error rate will increase by 32X. Therefore, the total error rate (TER) of exaFLOPS system is modeled as,

$$\begin{aligned} TER_{EFLOPS} &= HER_{EFLOPS} + SER_{EFLOPS} + OER_{EFLOPS} \\ &= HER_{PFLOPS} + 32 \times SER_{PFLOPS} + OER_{PFLOPS} \end{aligned} \quad (8)$$

**Checkpoint Size:** To evaluate the checkpoint overhead for various system configurations, we need the average amount of data written by each node. Since it is hard to mimic the memory trace of a real supercomputer, we execute the NAS Parallel Benchmark (NPB) [NASA 2009] on an actual system to determine the memory footprint of different workloads. The workloads are chosen from NPB CLASS-C working set size except for workloads DC and FT that employs CLASS-B working set since they are the most complex level that our environment can handle. Table XIII shows the memory usage of workloads that is projected for our baseline petaFLOPS system. We employ the same scaling rule applied for memory size to project the checkpoint size for future systems, thus the memory usage percentage remains the same.

Table XII shows the MPP system configurations for a petaFLOPS and a projected exaFLOPS system. For the configurations between these two ends, we scale the specification values according to the time frame. For all our evaluations we assume the timing overhead of initiating a coordinated checkpoint is $1ms$, which is reported as the latency of data boardcasting for hardware broadcast trees in BlueGene/L [Adiga et al. 2002].

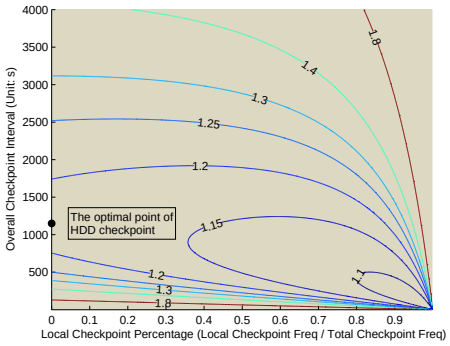


Fig. 17. Effect of checkpoint interval and ratio on execution time of Pure-HDD (at the points where X-axis is 0 so that all checkpoints are global).

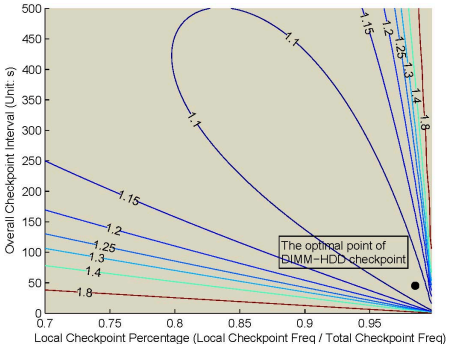


Fig. 18. Effect of checkpoint interval and ratio on execution time of DIMM+HDD (a zoom-in version of Fig. 17 on the bottom right corner).

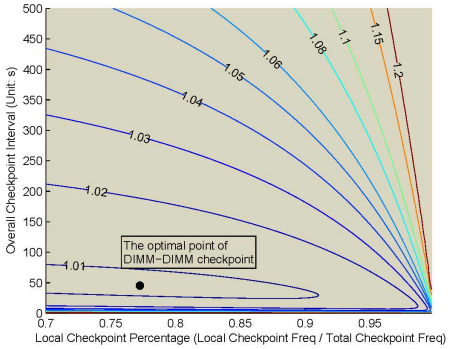


Fig. 19. Effect of checkpoint interval and ratio on execution time of DIMM+DIMM.

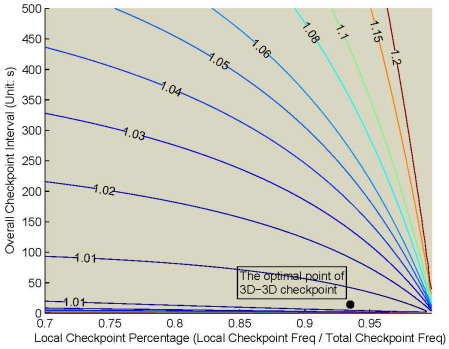


Fig. 20. Effect of checkpoint interval and ratio on execution time of 3D+3D.

## 6.3 Performance Analysis

For all our evaluations, we employ the equations derived in Section 4.4 to determine the execution time of workloads in various systems and scenarios.

For a given system, based on the system scale and the checkpoint size, the optimal checkpoint frequency can be decided. For this checkpoint frequency, an inherent trade-off exists between the proportion of local and global checkpoints. For example, as the fraction of local checkpoints increases, the overall checkpoint overhead drops, but the recovery time from global checkpoints rises; on the other hand, as the fraction of global checkpoints increases, the recovery time decreases, but the total execution time can take a hit because of the high checkpoint overhead. This trade-off is actually modeled by Equation. 7 in Section 4.4, and the optimal values of the checkpoint interval ($\tau$) and the percentage of local checkpointing ($p_L$) can be found.

This effect is illustrated in Figs. 17-20 for the different scenarios listed in Table XI for a petaFLOPS system when the workload DC.B is simulated. The performance value is normalized to the computation time. Not surprisingly the *Pure-HDD*

Table XIV. The checkpoint overhead and system availability estimations.

| | Pure-HDD | DIMM+HDD | DIMM+DIMM | 3D+3D |
|---|---|---|---|---|
| Checkpoint overhead (1 PFLOPS) | 17.9% | 7.1% | 0.9% | 0.6% |
| System availability (1 PFLOPS) | 84.8% | 93.4% | 99.1% | 99.4% |
| Checkpoint overhead (10 PFLOPS) | 97.3% | 16.1% | 0.8% | 0.6% |
| System availability (10 PFLOPS) | 50.7% | 86.1% | 99.2% | 99.4% |
| Checkpoint overhead (100 PFLOPS) | - | 83.4% | 2.9% | 1.3% |
| System availability (100 PFLOPS) | 0% | 54.5% | 97.2% | 98.7% |
| Checkpoint overhead (1 EFLOPS) | - | - | 9.4% | 2.6% |
| System availability (1 EFLOPS) | 0% | 0% | 91.4% | 97.5% |

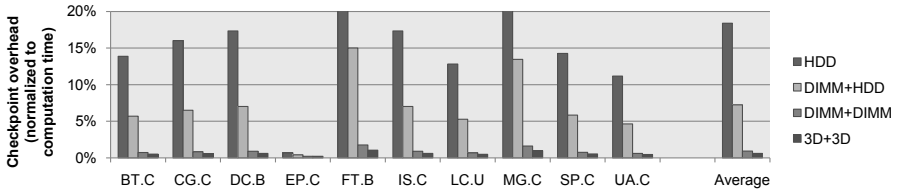


Fig. 21. The checkpoint overhead comparison in a 1-petaFLOPS system (normalized to the computation time).

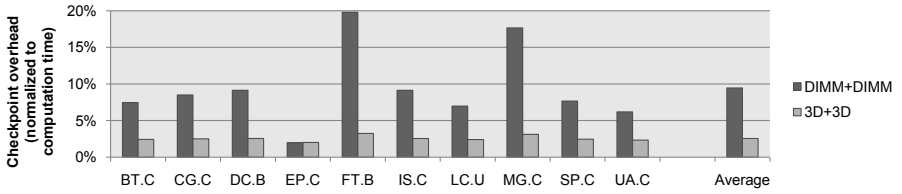


Fig. 22. The checkpoint overhead comparison in a 1exaFLOPS system (normalized to the computation time).

scheme, where all the checkpoints are performed globally using HDD (local checkpoint percentage is 0%), takes the maximum hit in performance. *DIMM+HDD*, including in-node PCRAM as local checkpointing storage, reduces the normalized checkpoint overhead from 17.9% to 7.1% with a local checkpointing percentage above 98%. As we change the global checkpointing medium from HDD to PCRAM-DIMM (*DIMM+DIMM*), the checkpoint overhead is dramatically reduced to 0.9% because HDD, the slowest device in the checkpoint scheme, is removed. In addition, since the overhead of global and local checkpoints are comparable in *DIMM+DIMM*, the optimal frequency for local checkpointing reduces to 77.5%. The *3D+3D* scheme that employs 3D DRAM/PCRAM hybrid memory has the least checkpoint overhead. We notice that the local checkpoint percentage in this case goes back to over 93% because the ultra-high 3D bandwidth enables a local checkpointing operation to finish almost instantly. Although the checkpoint overhead reduction achieved by *3D+3D* is similar to that of *DIMM+DIMM* in this case, we will see later that *3D+3D* does make a difference when future MPP systems reach the exascale.

Fig. 21 shows the checkpoint overhead in a petascale system by using *pure-HDD*, *DIMM+HDD*, *DIMM+DIMM*, and *3D+3D*, respectively. *DIMM+HDD* reduces the checkpoint overhead by 60% compared to *pure-HDD* on average. Moreover,

the ideal "instant checkpoint" is almost achieved by implementing *DIMM+DIMM* and *3D+3D*. As listed in Table XIV, the greatly reduced checkpoint overhead directly translates to the growth of effective computation time, or equivalent system availability.

The advantages of *DIMM+DIMM* and *3D+3D* are clear as the MPP system is scaled towards the exascale level where *pure-HDD* and *DIMM+DIMM* are not feasible any more; Fig. 22 demonstrates the results. It can be found that both of *DIMM+DIMM* and *3D+3D* are still workable, and more importantly, the average overhead of *3D+3D* is still less than 5% even in the exascale system. The resulting system availability estimations are listed in Table XIV. It shows that our intermediate PCRAM-DIMM and ultimate 3D-PCRAM checkpointing solutions can provide the failure resiliency required by future exascale systems with affordable overhead.

### 6.4 Power Analysis

Although the proposed techniques are targeted primarily to reduce the checkpoint overhead, they are useful for power reduction as well:

— Since PCRAM is a non-volatile memory technology, it does not consume any power when the system is not taking checkpoints. For example as shown in Table XIV, using *3D+3D* PCRAM checkpoints, during more than 95% of system running time the PCRAM modules can be turned off. Other approaches, i.e. battery-backed DRAM checkpointing, will inevitably leak power even when no checkpoints are being taken. Note that the nap power of a 2GB DRAM-DIMM is about 200mW [Meisner et al. 2009], using battery-backed DRAM checkpointing in 1-petaFLOPS systems will inevitably waste about 20kW power. In contrast, our PCRAM checkpointing module does not consume any power during the computation time.

— With future supercomputers dissipating many mega watts, it is important to keep high system availability to ensure that the huge power budget is effectively spent on useful computation tasks. As listed in Table XIV, *DIMM+DIMM* can maintain the system availability above 91% and *3D+3D* can achieve near 97% system availability even on the exascale level.

### 6.5 Scalability

Recall the motivation of the 3D PCRAM checkpointing is to maintain the checkpoint overhead under an acceptable level even when the MPP system reaches the exascale and the entire MPP system is highly unreliable. Hence we evaluate how different checkpointing schemes (as listed in Table XI) scale when the system scale goes up from today's petascale systems to future's exascale systems. In this analysis, we also consider the benefit achieved from incremental and background checkpointing.

Fig. 23 shows the effect of introducing local checkpointing on the total number of nodes in the system. It is clear that even with the incremental checkpointing optimization, the slow HDD checkpointing has trouble scaling beyond 1 petaFLOPS without taking a heavy hit in performance. Although the introduction of local PCRAM-DIMM checkpointing helps scale beyond 5 petaFLOPS, the poor scaling of HDD bandwidth hampers the benefit beyond 20 petaFLOPS. The use of PCRAM-DIMM for both local and global checkpoints further raises the bar to a

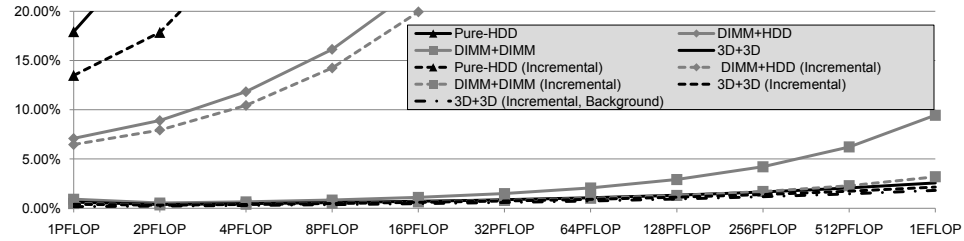


Fig. 23. The average estimated checkpoint overhead from petascale systems to exascale systems (normalized to computation time).

0.5 exaFLOPS system. Beyond that, due to the increase in transient errors and poor scaling of memory buses, its overhead increases sharply. The proposed hybrid checkpointing combined together with the 3D PCRAM/DRAM memory shows excellent scalability properties and incurs less than 3% overhead even beyond exascale systems.

Moreover, observing the incremental checkpointing curves in Fig. 23, it can be found that applying the incremental checkpoint in the conventional pure-HDD checkpoint does not extend the pure-HDD curve too much. However, when it is combined with PCRAM-based local/global hybrid checkpointing, this technique shows its great enhancement to the baseline schemes. That is because in our PCRAM hybrid checkpoint, the checkpoint interval can be set relatively low, and thus the number of dirty pages created during this interval or the incremental checkpoint size is dramatically reduced. This shows that when the 3D-PCRAM checkpoint is used together with the incremental checkpoint technique, the overall checkpoint overhead is only 2.1%, which can be translated into a MPP system availability of 97.9%. This negligible overhead makes the 3D-PCRAM checkpointing scheme an attractive method to provide reliability for future exascale systems.

### 6.6 SER Sensitivity Study

The effectiveness of the PCRAM-based local/global hybrid checkpointing depends on how many system failures can be recovered by local checkpoints. In our basic assumption, the soft error rate will increase by 32X in the exascale system. Combined with the 5X socket increase assumption, we find that the system MTTF almost degrades 116X. While our proposed PCRAM-based checkpointing is insensitive to this system MTTF degradation because over 99% of total failures are locally recoverable based on this assumption, the conventional HDD-based checkpointing is very sensitive to this change.

Although we believe aggressive soft error rate scaling is reasonable considering future "deep-nano" semiconductor processes, we cannot eliminate the possibility that the device unreliability can be hidden by some novel technologies in the future. In addition, our baseline setting, "ASCI Q", is widely considered as an unreliable system due to its non-ECC caches. Therefore, in order to avoid any exaggeration of the conventional checkpointing scalability issue, the scalability trend is re-evaluated with a new assumption that the soft error rate will remain at the same level as today's technology. Fig. 24 shows another set of checkpoint overhead projection curves based on this new assumption.

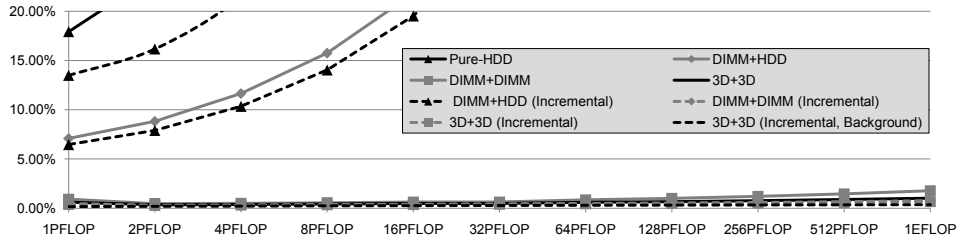


Fig. 24. The new checkpoint overhead projection based on the assumption that SER remains constant from petascale to exascale (normalized to computation time).

As expected, the checkpoint overhead decreases as the number of soft errors is reduced. However, even with this new assumption, the conventional HDD-based technique (*pure-HDD*) still has trouble scaling beyond the 8-petaFLOPS scale. In contrast, the overhead of our PCRAM-based approach (*DIMM+DIMM* and *3D+3D*) is further reduced to less than 3% by utilizing orthogonal techniques such as incremental checkpointing and RDMA.

## 7. RELATED WORK

As checkpointing-recovery is the widely-used technique for fault-tolerance in MPP, the related research is abundant [Elnozahy et al. 2002]. The coordinated protocol proposed by Chandy and Lamport [Chandy and Lamport 1985] is the most commonly used scheme due to its simplicity of implementation. In this approach, nodes are synchronized to ensure a consistent state before taking a checkpoint. Several techniques are proposed to reduce the checkpoint overhead by either reducing checkpoint size or using diskless checkpointing. Plank *et al.* [Plank et al. 1999] proposed a manual approach that is known as "memory exclusion". In "memory exclusion", the programmers are responsible for differentiating critical data from more temporary data that could be removed from the checkpoint image. Although compilers can manage the exclusion, this is not a general transparent method. Other work of reducing the checkpoint size mostly relies on the incremental checkpointing technique [Sancho et al. 2004; Naksinehaboon et al. 2008] that consists of saving only the differences between two consecutive checkpoints. The OS memory management subsystem is leveraged to decide the dirty data. Another way to reduce the time to checkpoint is to avoid checkpoint on the parallel file system and instead to use in-memory checkpointing. In diskless checkpoint [Silva and Silva 1998; Plank et al. 1998], all computing nodes store their checkpoint image in their memory. Additional nodes are necessary to store a checksum of the computing node in-memory checkpoints. All these approaches are proven to be effective to reduce the checkpoint overhead.

Oliner *et al.* [Oliner et al. 2006] introduced a theory of *cooperative checkpointing* that uses global knowledge of the state and health of the machine to improve performance and reliability by dynamically initiating checkpoints. However, in order to reduce the checkpoint cost, the technique skips some scheduled checkpoints according to the risk of system failure. This decision depends on the accuracy of risk estimation. Unfortunately, an accurate failure prediction or risk estimation is a challenging problem.

Bronevetsky *et al.*[Bronevetsky et al. 2008] presented a novel compiler analysis for optimizing automated checkpointing. Their work is a hybrid compiler/runtime approach, where the compiler optimizes certain portions of an otherwise runtime checkpointing solution, and then reduces the checkpoint size.

This previous research on checkpoint optimization reduces the checkpoint size, dynamically tunes the checkpoint interval, and sacrifices the system reliability by only supporting limited numbers of node failures. In contrast, our study in this paper shows how to take advantage of emerging PCRAM technology to dramatically improve the checkpoint dumping rate, and is complementary to other advanced checkpointing ideas.

Chiueh and Deng [Chiueh and Deng 1996] proposed a diskless checkpointing mechanism that employs volatile DRAM for storing both local and global checkpoints. Their idea is to split the DRAM memory in each node into four segments and employ three-fourths of the memory to make checkpoints. Sobe [Sobe 2003] also analyzed the overhead reduction by introducing the idea of local checkpoint storage and augmentation with parity, stored on another host. However, his research is still constrained in using HDD as the checkpoint storage. Bronevetsky and Moody [Bronevetsky and Moody 2009] showed the necessity of using node-local storage to build a scalable checkpoint/restart (SCR) library and used ramdisk to demonstrate unprecedented checkpoint write speed approach 1TB/sec. While these proposals are similar to this work, the introduction of the PCRAM modules eliminate the drawbacks of using the volatile DRAM or the slow HDD and SSD as the checkpoint targets.

## 8. CONCLUSION

Checkpointing has been an effective tool for providing reliable and available MPP systems. However, our analysis showed that current checkpointing mechanisms incur high performance penalties and are woefully inadequate in meeting future system demands. To improve the scalability of checkpointing, we introduce the emerging PCRAM technology into the supercomputer system as a fast checkpoint device. More importantly, we propose a hybrid checkpointing technique that takes checkpoints in both private and globally accessible memory, which not only improves the checkpoint performance by itself but also brings extra benefits through incremental and background checkpointing. We then develop a theoretical model based on failure rates and system configuration to identify the optimal local/global checkpoint interval that maximizes system performance. A thorough analysis of failure rates shows that a majority of failures are recoverable using local checkpoints, and local checkpoint overhead plays a critical role for MPP scalability. To improve the efficiency of local checkpoints and maximize fault coverage we propose PCRAM-DIMM checkpointing. PCRAM-DIMM checkpointing enables MPP systems to scale up to 500 petaFlops with tolerable checkpoint overhead. To provide reliable systems beyond this scale, we leverage emerging 3D die stacking and propose 3D PCRAM/DRAM memory for checkpointing. After combining all the effects, our proposed checkpointing scheme incurs less than 3% overhead in an exascale system by making near instantaneous checkpoints.

## ACKNOWLEDGMENTS

This project is supported in part by NSF grants 0702617, 0720659, 0903432 and SRC grants. We also wish to thank Richard Kaufmann for sharing his original ideas and providing helpful discussions.

## REFERENCES

Adiga, N., Almasi, G., Almasi, G., Aridor, Y., Barik, R., et al. 2002. An Overview of the BlueGene/L Supercomputer. In *SC '02: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 60–71.

Bedeschi, F., Fackenthal, R., Resta, C., Donze, E. M., Jagasivamani, M., et al. 2009. A Bipolar-Selected Phase Change Memory Featuring Multi-Level Cell Storage. *IEEE Journal of Solid-State Circuits 44,* 1, 217–227.

Borkar, S. Y. 2005. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro 25,* 6, 10–16.

Bronevetsky, G., Marques, D. J., Pingali, K. K., et al. 2008. Compiler-Enhanced Incremental Checkpointing for OpenMP Applications. In *PPoPP '08. Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 275–276.

Bronevetsky, G. and Moody, A. 2009. Scalable I/O Systems via Node-Local Storage: Approaching 1 TB/sec File I/O. Tech. Rep. LLNL-TR-415791, Lawrence Livermore National Laboratory.

Cappello, F. 2009. Fault Tolerance in Petascale/ Exascale Systems: Current Knowledge, Challenges and Research Opportunities. *International Journal of High Performance Computing Applications 23,* 3, 212–226.

Chandy, K. M. and Lamport, L. 1985. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems 3,* 1, 63–75.

Chiueh, T.-C. and Deng, P. 1996. Evaluation of Checkpoint Mechanisms for Massively Parallel Machines. In *FTCS '96. Proceedings of the 26th Annual Symposium on Fault Tolerant Computing*. 370–379.

Daly, J. T. 2006. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems 22,* 3, 303–312.

Dong, X., Jouppi, N., and Xie, Y. 2009. PCRAMsim: A System-Level Phase-Change RAM Simulator. In *ICCAD '09. Proceedings of the International Conference on Computer-Aided Design*.

Dong, X., Muralimanohar, N., Jouppi, N., Kaufmann, R., and Xie, Y. 2009. Leveraging 3D PCRAM Technologies to Reduce Checkpoint Overhead for Future Exascale Systems. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*.

Duell, J., Hargrove, P., and Roman, E. 2002. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart. Tech. Rep. LBNL-54941, Lawrence Berkeley National Laboratory.

Elnozahy, E. N., Alvisi, L., Wang, Y.-M., and Johnson, D. B. 2002. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys 34,* 3, 375–408.

Grider, G., Loncaric, J., and Limpart, D. 2007. Roadrunner System Management Report. Tech. Rep. LA-UR-07-7405, Los Alamos National Laboratory.

Hanzawa, S., Kitai, N., Osada, K., et al. 2007. A 512kB Embedded Phase Change Memory with 416kB/s Write Throughput at 100$\mu$A Cell Write Current. In *ISSCC '07. Proceedings of the 2007 IEEE International Solid-State Circuits Conference*. 474–616.

Huang, W., Sankaranarayanan, K., Skadron, K., et al. 2008. Accurate, Pre-RTL Temperature-Aware Design Using a Parameterized, Geometric Thermal Model. *IEEE Transactions on Computers 57,* 9, 1277–1288.

International Technology Roadmap for Semiconductors. Process Integration, Devices, and Structures 2007 Edition. `http://www.itrs.net/`.

Kash, J. 2009. Photonics in Supercomputing: The Road to Exascale. In *Integrated Photonics and Nanophotonics Research and Applications*. Optical Society of America, IMA1.

Los Alamos National Laboratory. 2009. Reliability Data Sets, `http://institutes.lanl.gov/data/fdata/`.

Meisner, D., Gold, B. T., and Wenisch, T. F. 2009. PowerNap: Eliminating Server Idle Power. In *ASPLOS '09. Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. 205–216.

Michalak, S. E., Harris, K. W., Hengartner, N. W., et al. 2005. Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratory's ASCI Q Supercomputer. *IEEE Transactions on Device and Materials Reliability 5,* 3, 329–335.

Naksinehaboon, N., Liu, Y., Leangsuksun, C., Nassar, R., Paun, M., and Scott, S. L. 2008. Reliability-aware approach: An incremental checkpoint/restart model in hpc environments. In *CCGRID '08. Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*. 783–788.

NASA. 2009. NAS Parallel Benchmarks. `http://www.nas.nasa.gov/Resources/Software/npb.html`.

Oldfield, R. A., Arunagiri, S., Teller, P. J., et al. 2007. Modeling the Impact of Checkpoints on Next-Generation Systems. In *MSST '07. Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies*. 30–46.

Oliner, A., Rudolph, L., and Sahoo, R. 2006. Cooperative Checkpointing Theory. In *IPDPS '06. Proceedings of the 20th International Parallel and Distributed Processing Symposium*. 14–23.

Pellizzer, F., Pirovano, A., Ottogalli, F., et al. 2004. Novel $\mu$Trench Phase-Change Memory Cell for Embedded and Stand-Alone Non-Volatile Memory Applications. In *Proceedings of the 2004 IEEE Symposium on VLSI Technology*. 18–19.

Pirovano, A., Lacaita, A. L., Benvenuti, A., et al. 2003. Scaling Analysis of Phase-Change Memory Technology. In *IEDM '03. Proceedings of the 2003 IEEE International Electron Devices Meeting*. 29.6.1–29.6.4.

Plank, J. S., Chen, Y., Li, K., Beck, M., and Kingsley, G. 1999. Memory Exclusion: Optimizing the Performance of Checkpointing Systems. *Software – Practice and Experience 29,* 2.

Plank, J. S., Li, K., and Puening, M. A. 1998. Diskless Checkpointing. *IEEE Transanctions on Parallel Distributed Systems 9,* 10, 972–986.

Reed, D. 2004. High-End Computing: The Challenge of Scale. Director's Colloquium, May 2004.

Sancho, J. C., Petrini, F., Johnson, G., and Frachtenberg, E. 2004. On the Feasibility of Incremental Checkpointing for Scientific Computing. In *IPDPS '04. Proceedings of the 18th International Parallel and Distributed Processing Symposium*. 58–67.

Silva, L. M. and Silva, J. G. 1998. An Experimental Study about Diskless Checkpointing. In *EUROMICRO '98. Proceedings of the 24th Conference on EUROMICRO*. Vol. 1. 395–402.

Sobe, P. 2003. Stable Checkpointing in Distributed Systems without Shared Disks. In *IPDPS '03. Proceedings of the 17th International Parallel and Distributed Processing Symposium*. 214–223.

Vantrease, D., Schreiber, R., Monchiero, M., et al. 2008. Corona: System Implications of Emerging Nanophotonic Technology. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*. 153–164.

Xie, Y., Loh, G. H., Black, B., and Bernstein, K. 2006. Design Space Exploration for 3D Architectures. *ACM Journal of Emerging Technologies in Computing Systems 2,* 2, 65–103.

Young, J. W. 1974. A First Order Approximation to the Optimal Checkpoint Interval. *Communications of the ACM 17*, 530–531.

Zhang, Y., Kim, S.-B., McVittie, J. P., et al. 2007. An Integrated Phase Change Memory Cell With Ge Nanowire Diode For Cross-Point Memory. In *Proceedings of the 2007 IEEE Symposium on VLSI Technology*. 98–99.

Zhou, P., Zhao, B., Yang, J., and Zhang, Y. 2009. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *ISCA '09: Proceedings of the International Symposium on Computer Architecture*. 14–23.