Identifying Opportunities for Byte-Addressable Non-Volatile Memory in Extreme-Scale Scientific Applications

Dong Li†, Jeffrey S. Vetter†, Gabriel Marin†, Collin McCurdy†, Cristian Cira⋆, Zhuo Liu⋆ and Weikuan Yu⋆

†Oak Ridge National Laboratory, ⋆Auburn University
{lid1,vetter,maring,cmcurdy}@ornl.gov, {cmc0031,zhuoliu,wkyu}@auburn.edu

***Abstract*—Future exascale systems face extreme power challenges. To improve power efficiency of future HPC systems, non-volatile memory (NVRAM) technologies are being investigated as potential alternatives to existing memories technologies. NVRAMs use extremely low power when in standby mode, and have other performance and scaling benefits. Although previous work has explored the integration of NVRAM into various architecture and system levels, an open question remains: do specific memory workload characteristics of scientific applications map well onto NVRAMs' features when used in a hybrid NVRAM-DRAM memory system? Furthermore, are there common classes of data structures used by scientific applications that should be frequently placed into NVRAM? In this paper, we analyze several mission-critical scientific applications in order to answer these questions. Specifically, we develop a binary instrumentation tool to statistically report memory access patterns in stack, heap, and global data. We carry out hardware simulation to study the impact of NVRAM for both memory power and system performance. Our study identifies many opportunities for using NVRAM for scientific applications. In two of our applications, 31% and 27% of the memory working sets are suitable for NVRAM. Our simulations suggest at least 27% possible power savings and reveal that the performance of some applications is insensitive to relatively long NVRAM write-access latencies.**

I. Introduction

Exascale computing platforms are expected to become available later this decade. Although the precise details of the exascale systems are not yet known, it is rather certain that these systems will bring with them grand challenges in several different dimensions. The most pervasive of these challenges is managing power consumption [1]. Future exascale systems will require much higher energy efficiency than today's systems. To illustrate, the Jaguar supercomputer in Oak Ridge National Laboratory runs at 0.25 Gigaflops/Watt. For a 1 exaflops supercomputer, at least 40 Gigaflops/Watt is required in order to manage power consumption: an improvement of two orders of magnitude is required. In current systems, main memory accounts for up to 40% of the server energy, comparable to or slightly higher than the processor's contribution. Furthermore, power consumption by main memory can result in resiliency, scalability and cost issues.

Byte-addressable, non-volatile, solid-state memory devices (NVRAMs) such as phase-change memory (PCRAM), spin-torque transfer memory (STTRAM), and resistive RAM (RRAM), have been investigated as potential alternatives to existing memory technologies in future computing systems. These new memory devices provide the non-volatility of conventional disks while providing speeds approaching those of DRAM. Unlike DRAM, NVRAMs are not suffering from either the leakage power or the need to refresh memory cells - phenomena which account for more than 35% of the memory subsystem power consumption for memory-intensive workloads. Therefore, using NVRAM may potentially bring significant power savings to future exascale systems. NVRAM is attractive for reasons other than its impressive power savings potential, as well. NVRAM could provide substantial bandwidth for checkpointing and, since it would enable checkpointing to be brought under the control of hardware, would drastically reduce latency. This will become increasingly important in exascale systems, given the aforementioned resiliency challenge, and limited external I/O bandwidth. Finally, NVRAM provides a viable alternative to support memory paging in HPC (high performance computing) systems, and is fast enough to be competitive with increasing DRAM capacity.

Recent studies have highlighted the possible impact of NVRAM technologies on performance and energy when they are used as a straightforward replacement for DRAM or hard-disk drives. Some solutions have been proposed at the architecture level to avoid NVRAM's limitations while improving systems' power efficiency [2], [3], [4], [5]. Other research focuses on redefining system interfaces or operating primitives [6], [7], [8] to accommodate NVRAM. However, many of these analyses have not considered NVRAM's potential impact on applications. It is not clear whether large-scale mission critical scientific applications can really benefit from NVRAM or, if so, how many opportunities there might be. For instance, it may be possible to co-design applications to better leverage the strengths of NVRAM. To the best of our knowledge, these questions have not yet been adequately addressed, and until we answer them, we will not be able to effectively argue for including hybrid NVRAM memory architectures in future Exascale systems.

In this paper, we analyze the workload characteristics of several applications, and assess several computing metrics

that are important for using NVRAM. Specifically, we make the following contributions.

- We analyze four mission-critical, large-scale scientific applications, and use the empirical results to drive the understanding of each application's design goals for individual data structures. We further generate several general observations for application data structures that apply broadly to many applications beyond our initial set.
- We implement a PIN-based binary instrumentation tool to measure and classify application data structures that may be a good fit for NVRAM in a hybrid DRAM-NVRAM memory architecture
- We perform power simulation based on an existing power simulation tool to estimate power consumption for NVRAM systems, using these memory traces collected from the real scientific applications.
- We carry out full-system simulation to study the performance sensitivity of these applications to memory access latencies of different NVRAM systems.

The rest of this paper is organized as follows. Section II provides background information for NVRAM. Section III describes in more details our tool implementation – NV-SCAVENGER. Sections IV and V explain how we apply simulation to investigate power and performance of NVRAM. Section VI briefly reviews the four applications involved in this study. Section VII presents our experimental analysis and observations. Section VIII discusses related work and Section IX concludes the paper.

II. Background

NVRAMs have zero standby power, high density, non-volatility and byte addressability [9], which can be very beneficial for HPC systems. However, they also have limitations. The first limitation is their relatively long and asymmetric access latencies, compared to conventional DRAM. For example, STT-RAM write latency is about four times that of DRAM, although its read latency is about the same as that of DRAM. PCRAM write and read latencies are 10 and 2 times longer than those of DRAM. The second limitation is their high dynamic power consumption for write operations. For example, PCRAM's write energy/bit for resetting memory cells is about 50 times higher than that of DRAM. The third limitation is their limited write endurance. For example, today's state-of-the-art processor technology has demonstrated that the write endurance for PCRAM is around 10^8 and $10^{9.7}$, much worse than that of DRAM (10^{16}).

Based on these characteristics, we divide NVRAMs into three categories: (1) NVRAMs with long access latencies for both read and write operations (e.g., PCRAM and Flash memory); (2) NVRAMs with long write latencies and read access latencies comparable to DRAM (e.g., STTRAM); (3) NVRAMs with performance very close to and even slightly better than DRAM (e.g., RRAM). The first category of memory technologies are relatively mature and have been commercialized. The third category of memories are far from mature and most current research for them is limited at the device level [10], [11]. In this paper, we target the first two categories of NVRAMs that are expected to be more easily adopted in future Exascale HPC systems.

NVRAMs must be properly managed to alleviate their limitations. For the first NVRAM category, memory accesses should be controlled such that performance and device endurance is within acceptable constraints. In particular, write accesses must be rigorously managed because their longer latency and higher impact to system performance. For the second category of NVRAMs, frequently written memory pages should not be placed in NVRAMs while read-intensive pages should be. For both categories, a general management policy is to place memory pages in NVRAMs as much as possible while avoiding performance-critical frequent accesses (especially write accesses) to NVRAM, such that energy savings are maximized and performance losses are minimized.

To leverage NVRAMs, researchers have proposed hybrid memory systems that combine DRAM and NVRAM [2], [12], [4], [3]. A hybrid memory system can be hierarchical, using DRAM as a cache to reduce NVRAM access latency, or horizontally putting NVRAM and DRAM side-by-side behind the bus. Data movement between the two memories can be realized by extending architectures with OS support [3]. The first design does not fit well for many scientific applications. For workloads with poor locality, the DRAM cache actually lowers performance and increases energy consumption. Previous work has shown that real world applications can exhibit very low spatial and temporal locality [13]. This is especially true for some large-scale scientific simulations with irregular memory access patterns. Therefore, our discussion in this paper focuses on the second hybrid memory system.

Given the benefits of NVRAM, we characterize the access patterns of scientific applications to explore opportunities for data placement in hybrid memory systems. Three metrics are used to quantify NVRAM opportunities. The first metric is the read/write ratio. A higher read/write ratio translates to less intensive write workloads, which are favored by NVRAM, especially the second category of NVRAM. The second metric is the memory size of a memory object. Since the static power savings are directly related to memory size, we want to put as much application data into NVRAM as possible. The third metric is the memory reference rate. This metric is complementary to the first metric for some corner cases. In particular, a memory object with a high read/write ratio may still account for a large fraction of write memory accesses, which are not favored by the first category of NVRAMs. Using this metric we can identify this type of data structures and avoid placing them into NVRAM. In

our analysis, we use these three metrics to evaluate each time step of the main computation. We then compare the results across time steps to monitor the variance of the access patterns for each memory object. For a dynamic page placement solution [3], this information is valuable because it reflects how the usage of memory objects changes and if the change in memory usage benefits NVRAMs (e.g. from low read/write ratio to high read/write ratio).

III. Analysis Tool

To investigate workload characteristics of applications, we have developed a binary instrumentation tool based on PIN [14], named NV-SCAVENGER. It instruments every instruction and then statistically reports NVRAM related access patterns. A naive design of such a tool would be to detect every memory reference, and, then, count the number of memory references and their type at the granularity of the whole address space. Although this method is fast in instrumentation time, it leaves out many opportunities for leveraging NVRAM because access patterns can vary greatly from one memory block to another. Alternatively, we can detect access patterns at a very fine granularity (e.g., each memory byte). Since NVRAM is byte-addressable, putting each byte of NVRAM friendly memory into NVRAM will maximize the power savings. However, this approach would have intolerable instrumentation costs and large memory requirements to store access information. In our work, we investigate access patterns at an appropriate granularity (i.e., an application memory object). A memory object can be an application data structure, such as a data array, that saves the computation state, or it can also be a stack frame associated with a subroutine invocation. We differentiate between memory objects in heap, data segment, and stack, because it helps us to better understand how the applications uses these memory objects, and it provides insightful information for refactoring applications. For example, if there are NVRAM friendly heap memory objects, we can then extend dynamic memory allocation to explicitly direct the allocation of memory into NVRAM; if there are NVRAM friendly stack memory objects, we can then consider a better stack implementation with better control over the stack data placement.

In addition to reporting access patterns of memory objects, we further implement a configurable cache hierarchy simulator within the tool. It takes memory references from the instrumentation tool as the input, and outputs memory traces filtered by the cache hierarchy. As a result, memory traces represent main memory accesses due to last level cache misses and cache evictions. The memory traces are then used by our memory power simulator to estimate the power consumption of NVRAM. The main diagram of NV-SCAVENGER is illustrated in Figure 1. In the next subsections, we will describe how we implement NV-SCAVENGER for memory objects in applications' stack, heap, and global data segments.

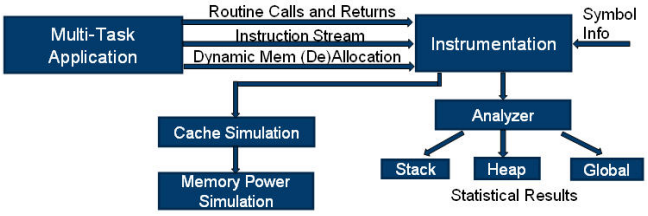

Figure 1: Diagram of the analysis tool

A. Application Program Stack

The stack memory of applications is ignored in previous NVRAM research [8]. This is partly due to the assumption that stack data is volatile and frequently accessed thus it is not a good candidate for NVRAM. However, we consider stack data in our work, since scientific applications can have significant amount of memory references to stack data (Section VII). Many computation kernels access data allocated on stack. It is pretty common in scientific applications to increase the stack size and put more computation data on the stack. Therefore, it is important to investigate access patterns to the stack.

NV-SCAVENGER instruments stack data with two different methods. In the first method, we record the number of read and write operations to the entire program stack. In particular, for each memory reference, we record the current stack pointer besides the memory reference information (i.e., address and operation type). We also record the maximum value that the stack pointer has had during the execution of the program. Assuming that the stack pointer grows downwards, if the effective memory address stays between the maximum stack pointer and the current stack pointer, this memory reference is counted as a stack memory reference. This method involves a simple recording of memory references and a comparison to the stack pointer register. Therefore it is light-weighted and much faster than the second method. The second method instruments stack data at a finer granularity. In particular, it not only positions memory references within the entire program stack during execution, but also identifies which routine's stack frame is accessed. To implement this functionality, we instrument all function calls and return points so that we can maintain a shadow stack in NV-SCAVENGER. This provides the convenience of unwinding the stack. We also record the base frame address at each routine call. For each memory reference, we traverse through our call stack to attribute the effective memory address to the corresponding routine's frame. It is possible that the currently executing routine may access a frame underneath the current routine's frame. In this case, the memory reference is attributed to the underneath frame. This makes sense when considering data placement, because it is the previously called routine that really allocates data on the stack. In addition, to identify the routine, we use the starting address of the routine as its signature, because we

can easily obtain routine name and image name based on this address using the PIN API.

B. Heap

One straightforward method to identify a memory object as belonging to the heap is to intercept calls to the memory allocation routines. In particular, we search the memory allocation routines within the program image and then insert instrumentation points at the routine entry and exit points to obtain a memory object's base address and the data size. The special memory allocation *realloc()* is treated as a call to the memory deallocation routine followed by a regular call to *malloc()*.

Because of frequent memory allocation and deallocation operations on the program heap, some already deallocated heap objects may share the same virtual memory address with an active heap memory object. Therefore, we associate a flag with each heap memory object to mark the status of memory objects. Whenever a memory object is deallocated, we set its associated flag to indicate that it is dead. This behavior is implemented by instrumenting *free()* at its entry point.

To identify a heap memory object, we use multiple fields as its signature, including the base address, the size, the line number and the file name for the function call, and the starting addresses of the routines currently active in the shadow stack. We maintain a callstack in NV-SCAVENGER as described in the stack section. With these fields, we can distinguish between heap objects while associating them with the application code. Under this scheme, it is still possible that memory objects allocated during different execution phases have the same signature. For example, a heap memory region may be allocated in the middle of each computation iteration with the same callstack, base address and memory size, and then deallocated at the end of each iteration. We regard these different memory objects as the same one in NV-SCAVENGER, because they appear within the same program context and tend to have the same access pattern. If these memory objects are NVRAM friendly, they should always be placed into NVRAM whenever the program invokes dynamic memory allocation at the specific execution points with the same program context. This approach reduces the number of heap memory objects we have to track and introduces more application information into our analysis.

We instrument memory allocation and deallocation at the system library level, instead of at the application level. This brings a lot of convenience for analyzing Fortran 90/99 dynamic memory allocation. In Fortran 90/99, a single memory allocation routine can allocate multiple memory regions, each one corresponding to a memory object. Instrumenting at a low-level can easily help detect each allocation and provide the convenience for analyzing access patterns at a finer granularity, while instrumenting at the high level cannot achieve this.

During instrumentation we only consider heap data explicitly allocated by the application. We do not consider memory allocations in third-party system libraries because the application developers usually have less control over the placement of that data and this paper focuses on studying the characteristics of the application itself.

C. Global

We obtain global data information (i.e., symbol name, base address and memory size) from the application's executable using *libdwarf*. We associate a memory object with its symbol name and base address. However, this simple identification mechanism does not work well for FORTRAN common blocks, because a common block allows one program unit to have a different view of a shared memory block from other program units. In particular, the data within a common block can be re-partitioned and re-identified with different variable names across procedure units. Therefore, different memory identification may point to memory regions with overlapped data blocks. To solve this problem, we regard the memory objects with overlapped data blocks as one single memory object whose address range is the union of individual memory regions. We choose the combined symbol name of individual memory objects to identify the new memory object.

D. Improve Instrumentation Performance

Our initial implementation of NV-SCAVENGER slows down an application's execution significantly, from hundreds to thousands fold. With this substantial overhead, instrumenting a large parallel application is impractical. Therefore, we invested considerable effort in optimizing instrumentation performance.

This significant overhead for NV-SCAVENGER's runtime instrumentation comes from the extensive instrumentation functionality it needs. One possible solution is to offload major instrumentation functionality into an offline tool. In particular, we move the cache simulation and memory reference attribution for different type of memory objects into an offline tool, keeping only the functionality of tracing memory references in the online instrumentation tool. This solution reduces the instrumentation overhead significantly. Meanwhile, it avoids running applications repeatedly and provides flexibility for data processing. However, it is not a scalable solution. A short serial HPC application can easily produce a trace of tens of gigabytes of data. Post-processing the trace by I/O operations, even though the trace file is compressed, is also very slow. The instrumentation time plus post-processing time will be even longer than that of our initial instrumentation tool. So we stick to the original design, i.e., computing statistics on the address stream on-the-fly without storing raw traces. Furthermore, we cut the

original design into three tools to process stack, heap and global data separately. We run the three tools in parallel to collect memory access patterns.

To further improve the performance of NV-SCAVENGER, we use a memory buffer to temporarily store memory traces. Any memory reference is simply placed into the buffer until the buffer is full. All addresses in the buffer are then processed at once. This scheme delays data analysis and reduces the frequency of interferences with the program data cache caused by data processing. This method also simplifies the instrumentation code per memory access and reduces state saving overhead for instrumentation.

We also apply two methods to speed up the updating of reference records for memory objects. In particular, for any memory reference, we must search all recorded memory objects to identify which memory object is accessed and then update access records for that memory object. To speed up searching, we divide the memory address space into many buckets and distribute the memory objects into the buckets based on their address range. To decide which memory object a memory reference belongs to, we apply a memory masking scheme to the reference address to choose the bucket corresponding to this address, and then search for memory objects within the chosen bucket. To avoid clustering memory objects into very few buckets and invalidating the bucket scheme, we dynamically divide the memory address space so that the memory objects can be evenly distributed between buckets. We also employ a small software cache using LRU algorithm to save information for most often used memory objects. This scheme provides a shortcut for updating access records for memory objects.

Another possible performance improvement for instrumentation is to use sampling, a technique widely employed in hardware simulation to reduce instrumentation points. With a tool like SimPoint [15], one can detect phases in program behavior and use periodic sampling to produce representative results when an appropriate sampling period is chosen. However, sampling is not applicable to our case study, because we intend to establish a memory access panorama for all memory objects. Sampling can lead to the loss of access information for many memory objects, which in turn causes improper data placement.

IV. Memory Power Simulation

Our memory power simulator is based on DRAMSim2 [16]. It is designed to be a power estimation model of the memory controller, the device modules and the buses of any random access memory (RAM). This simulator uses trace files collected from NV-SCAVENGER and outputs the simulated power consumption for NVRAM. It can be integrated into full system simulators too.

The simulator has three modules. The first one, the memory system, integrating the other two, acts as an interface to other full system simulator components or, in our case, to the trace files. When the power simulator is integrated with a full system simulator that provides timing information, power estimates can be accurately computed. In the absence of timing information, when the simulator is fed with memory traces, memory requests are processed by the memory system at full speed. Our simulation reports the average memory power in this case. The second module is the memory controller. It regulates the flow of transactions to and from the NVRAM devices. This includes address mapping, row policy and bank state updates. Last but not least, the third module simulates memory ranks. This module is responsible for tracking down the errors in scheduling, handling the command transactions issued by the memory controller and powering up or down the banks. Our power simulator for NVRAM includes power components for burst power (i.e., the cost for reading/writing memory cells), background power, and activation/precharge power (depending on the availability of hardware parameters). Refresh power is 0 for NVRAM.

We make a few assumptions when estimating power for NVRAM. We assume that the peripheral circuitry (e.g., DIMM interface, row buffers, and row and column decodes) for NVRAM is the same as that of DRAM. Therefore, the circuitry has the same performance and power characteristics for NVRAM and DRAM. We also assume that the same memory protocol is used for NVRAM and DRAM. These assumptions are also used in previous NVRAM research [5], [12], [3]. For PCRAM, the reset current of memory cells is twice as big as the set current. In our power simulation, we assume that the set operation has the same operation current as the reset. Thus our power estimation should be bigger than that of the real PCRAM memory and provides a power consumption upper bound. For STTRAM and MRAM, the published hardware information is very limited and we do not have read/write current values for them. Therefore, we use those of PCRAM (40mA for read and 150mA for write) for the power simulation of STTRAM and MRAM. However, we expect that the real read/write currents of STTRAM and MRAM should be smaller than those of PCRAM. Hence, our power estimation for STTRAM and MRAM should be a power consumption upper bound.

V. Performance Simulation

We use PTLsim, a x86/x86-64 cycle accurate system simulator [17] to study the application performance with different NVRAMs. PTLsim models out-of-order processor cores with a full cache hierarchy, memory subsystem, and supporting hardware. PTLsim uses the co-simulation technique in which the simulator runs directly on a reference machine supporting the instruction set being simulated. Therefore, it can context switch between native mode and simulated mode completely transparent to the user code, enabling detailed simulation of specific program segments. For our study, we change the memory access latency to simulate

different NVRAM systems. Since the current simulator does not differentiate between read and write latencies, we assume the read latency is the same as the write latency. Because NVRAMs usually have longer latencies for writes than for reads, our simulation in fact provides a performance lower bound. In addition, we do not simulate a hybrid memory system due to the limitations of the simulator. Instead, we assume main memory is completely replaced with NVRAM.

Our simulation based on the above assumptions is a preliminary study. However, it provides insight about how sensitive the application performance is to the long memory access latencies of NVRAMs. Generally, memory access latency can be hidden by overlapping with computation and by memory parallelism. Architectural features such as prefetching can also hide memory access time. Moreover, if the application has good data locality, a large fraction of accesses to the main memory can be avoided. Therefore, using a memory system with a longer access latency does not necessarily lead to proportionally worse performance. We use simulation to understand how the application performance varies when the memory access latency changes.

VI. Applications

We investigate the following four production-level mission-critical large-scale scientific applications that are expected to run on extreme-scale platforms. The applications characteristics are summarized in Table I.

- **Nek5000** [18] is a dynamic solver simulating unsteady incompressible fluid flow with thermal and passive scalar transport on two- and three-dimensional domains. It is widely used in a broad range of scientific research, including thermal hydraulics of reactor cores, transition in vascular flows, ocean current modeling and combustion.
- **CAM** [19], the community atmosphere model, is a global climate model that provides state-of-art computer simulations of the Earth's past, present, and future climate states. Coupling CAM with other climate system models, scientists can address many important scientific questions (e.g., study the effects of greenhouse gases increase and investigate the interactions among the physical, chemical and biogeochemical subsystems).
- **GTC** [20] is is a massively parallel, particle-in-cell code for turbulence simulation in support of the burning plasma experiment, the crucial next step in the quest for next generation (fusion) energy.
- **S3D** [21] is a massively parallel direct numerical solver (DNS) for the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry. S3D can greatly advance our basic understanding of turbulent combustion processes and thus improve efficiency of combustion devices.

L1 cache (private)	Separate instruction cache and data cache, 32KB each, 4-way, 64byte cache line size, no-write-allocate
L2 cache (private)	1MB, 16-way, LRU, 64byte cache line size, write-allocate

Table II: Cache configuration

Feature	Value
CPU cores (2.266GHz, x86)	out of order, one thread per core, two quad-core processors
TLB per-core size	32 entries
L1 cache	8 banks, 1 CPU cycle hit
L2 cache	5 CPU cycle hit
Size of load fill request queue	64 entries
Size of miss buffer	64 entries
Memory devices	2GB, 16 banks, 16 ranks
Device width	4
JEDEC DATA BUS BITS	64
Num mem rows, Num mem cols	1024

Table III: System configuration

Scientific applications tipically include three execution phases: a pre-computing phase for initialization and input parsing, a main computation loop, and a post-processing phase for data aggregation and result output. We specifically instrument the main computation loop because it usually dominates execution time and yields major power saving opportunities for NVRAMs. For heap data, many memory objects are allocated during the pre-computing phase. To study access patterns for those memory objects, we instrument memory (de)allocation routines throughout the whole applications, but memory references to those objects are recorded only during the main computation loop.

VII. Application Results

This section presents the results of our investigation for the four applications described in Section VI. Tables II, III, and IV list the parameters of the simulation. Table II shows the parameters of the two-level cache used in both NV-SCAVENGER and the PTLsim simulator. Tables III and IV list the hardware parameters of the computing and memory systems simulated under this study. We collect data for the first 10 iterations of the main computation loop of each application.

A. Stack Data

We first apply the fast version of the tool to the four applications. The results are summarized in Table V. We first notice that references to stack data account for a large

Memory	Real read latency	Real write latency	Performance simulation
DRAM	10ns	10ns	10ns
PCRAM	20ns	100ns	100ns
STTRAM	10ns	20ns	20ns
MRAM	12ns	12ns	12ns

Table IV: Memory access latencies of different memory systems

Application	Input Problem Size	Description	Memory footprint per task
Nek5000	2D eddy problem	Fluid flow simulation	824MB
CAM(v3.1)	Default test case	Atmosphere model	608MB
GTC(v2)	Poloidal grid points=392, Track_particles=1, toroidal_grids=2, particle per cell for electron=7	Turbulence plasma simulation	218MB
S3D	Grid dimensions: 60x60x60	Turbulence combustion simulation	512MB

Table I: Applications characteristics

Application	Read/write ratio	Reference percentage
Nek5000	6.33	75.6%
CAM	20.39 (11.46)	76.3%
GTC	3.48	44.3%
S3D	6.04	63.1%

Table V: Stack data analysis

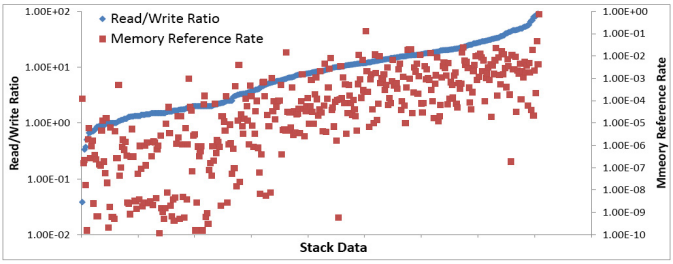

Figure 2: Read/write ratios and memory reference rates for the CAM stack data

fraction of total memory references. For Nek5000 and CAM, the percentage of references to stack data is higher than 70%. This shows that optimizing placement for stack data is very important for scientific applications. We further notice that read/write ratios for three of the applications are greater than 1, but less than 7. In other words, the number of reads is only moderately higher than the number of writes. This access pattern for stack data is not NVRAM friendly. On the other hand, CAM is characterized by a relatively large read/write ratio. Except for the first iteration whose read/write ratio is 11.46, the other 9 iterations have read/write ratios as high as 20.39. To further investigate the reason for this high read/write ratio, we use the slow version of the tool to analyze CAM stack data at finer granularity. The result is displayed in Figure 2.

We noticed that 43.3% of all the objects on the stack have read/write ratios larger than 10 and 3.2% have read/write ratios larger than 50. Accesses to these memory objects account for 68.9% and 8.9% of total memory references, respectively. To investigate the reason for the high read/write ratios, we further look into a few routines that have the highest read/write ratios. One of these routines uses its local variables to store interpolation coefficients derived from input arguments at the beginning of the routine. These coefficients are then frequently read during computation. Another routine widely uses local variables to periodically save temporal computation results that the later computation repeatedly reads and derives new results. A third routine uses stack to save some computation dependent constants. These constants are only needed in this routine and thus not declared as global data. All of these routines are called intensively from the main computational kernels, which explains why they have high memory reference rates. The high read/write ratios of these memory objects make them potential candidates to be placed in NVRAM. However, possible performance loss and short device endurance must also be considered due to these memory objects' high reference rates.

B. Global Data and Heap Data

We report analysis results for global data and heap data together for brevity. Figures 3-6 display read/write ratios, memory reference rates, and memory sizes for all global and heap memory objects of the four applications. We first notice that read-only data structures are common in all four applications. For Nek5000 and CAM we have about 59MB and 94MB read-only data, accounting for 7.1% and 15.5% of the memory footprints, respectively. These read-only data structures can be categorized into three common scenarios.

- **Auxiliary data structures.** In Nek5000 we found two data structures that are intermediate computation results derived from the mass matrices. These data structures store inversed mass matrices and "element-lagged" mass matrices. They are created during the pre-computing phase and then used as auxiliary matrices to facilitate computing. In CAM we found constants related to the Legendre transform, cosine and sine of longitudes in the global grid, a hash table of the field names to accelerate output processing and index arrays to facilitate look-ups. For GTC, we found auxiliary radial interpolation arrays used to relate particle positions. For S3D, we found look-up tables that contain coefficients for linear interpolation.
- **Computing-dependent read-only data.** These data structures include boundary conditions (e.g., 70 different types of boundary conditions in Nek5000), geometry arrays for maintaining the locality information for a grid (e.g., physics grid longitudes in CAM), and mass matrices (e.g., velocity mass matrix and temperature mass matrix in Nek5000). In many cases, these parameters are read in from a file, or calculated during initialization, and then read many times during the computation. One interesting feature of some of this read-only data is that the data may be read-only for specific input problems but read and written with other input problems. This is due to the random nature

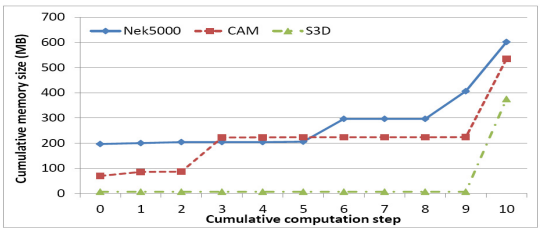

Figure 7: Cumulative distribution of memory usage across time steps

of many scientific simulations. The access patterns to this data can vary for different inputs.

- **Physical invariants.** For example, thermal conductivity for soil minerals and saturated soils in CAM, and convective characteristics data and strain rate invariants in Nek5000.

The sizes of some of the above data structures are proportional to the problem definition. With large-scale scientific applications, these data structures can be easily scaled to use large amounts of physical memory.

Nek5000 and CAM have many data structures whose read/write ratios are larger than 50. These data structures can be placed into NVRAM too, especially NVRAM of the second category (see Section II). These data structures use 38.6 MB and 4.8MB physical memory in Nek5000 and CAM, respectively. Furthermore, except for GTC, most memory objects in the applications have more read than write accesses (i.e., read/write ratio>1). Such objects are friendly to NVRAMs with slightly longer write latencies but similar read latencies as DRAM, such as STTRAM.

C. Memory Usage Variance

Besides the read/write ratio, memory reference rate, and memory size metrics, understanding how access patterns change in time is also important for determining data placement. If there are temporal NVRAM-friendly access patterns, a dynamic data placement scheme like [3] will have a chance to migrate data between DRAM and NVRAM to save power. We study memory usage variances across iterations of the main computation loop. A majority of scientific applications have computation loops dominating application execution. A computation loop provides a natural way to partition application execution. In our study, we instrument each iteration and then statistically present how the memory access patterns vary across iterations.

Figure 7 depicts the cumulative distribution of memory usage for memory objects across computation iterations (or time steps). The figures statistically reflect how memory objects are touched across computation iterations. The number 0 in the cumulative computation step axis stands for the pre-computing and post-processing phases. A data point (x,y) within the figures represents that there are y MB memory objects used in no more than x iterations. The figures do not include those short-term heap memory objects, but those long-term heap memory objects that are allocated at the pre-computing phase and used throughout the whole computation. The short-term heap memory objects are only temporarily allocated and then deallocated in the middle of the computation. Due to the volatility of these memory objects, their cumulative memory size does not represent a real opportunity for NVRAM. From the figures, we can find that many memory objects in Nek5000 and CAM are unevenly touched across computation iterations. For Nek5000, there are about 200MB of global data (24.3% of the memory footprint) not used in the main computation. Some of the data is used in the pre-computing phase to prepare computation methods (e.g., generating diagonal matrices); some of them are used in the post-processing phase, for example, aggregating data from MPI tasks. We identify about 70MB (11.5% of the memory footprint) and 7.1MB of such data for CAM and S3D, respectively. This data is suitable for being placed in NVRAMs with their low standby power. We also notice that some memory objects in Nek5000 and CAM are unevenly touched. They may only be used within a few computation iterations. These memory objects are candidates to be migrated, depending on the memory usage variance. We do not show the cumulative distribution of memory usage for GTC, because almost all of its memory objects are either used throughout the whole computation steps or used as short-term heap memory objects. In other words, the memory objects in GTC are pretty much evenly touched across iterations.

Figures 8-11 display the variances of read/write ratios and memory reference rates for memory objects across iterations. To statistically display the variance for all memory objects, we normalize the read/write ratio and the memory reference rate of each memory object at each iteration by the corresponding one in the first iteration. We display the distribution of these normalized values for each iteration. From the figures, we observe that the read/write ratios and the memory reference rates are pretty stable for most memory objects across iterations. There are more than 60% memory objects whose normalized values stay within [1,2) for each iteration. For S3D and GTC, almost all memory objects have their memory reference rates unchanged across iterations. This is good news for those NVRAM friendly memory objects. If their access patterns stay the same most of the time, it is safe to place them in NVRAM without data migration (i.e., no migration overhead). On the other hand, we notice that Nek5000 has quite diverse reference rates across iterations. To leverage NVRAM for those pages, a memory reference monitor working at a fine time granularity should be applied to dynamically decide the optimal location of a memory page for a hybrid memory system.

D. Power

Table VI displays the average power consumption for the four applications with different NVRAMs. The power data is normalized by the DRAM power data. With NVRAMs,

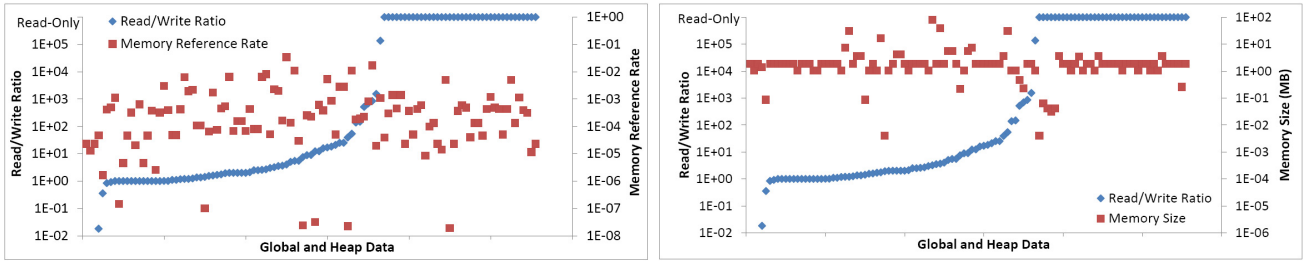

Figure 3: Read/write ratios, memory reference rates and memory object sizes for memory objects in Nek5000

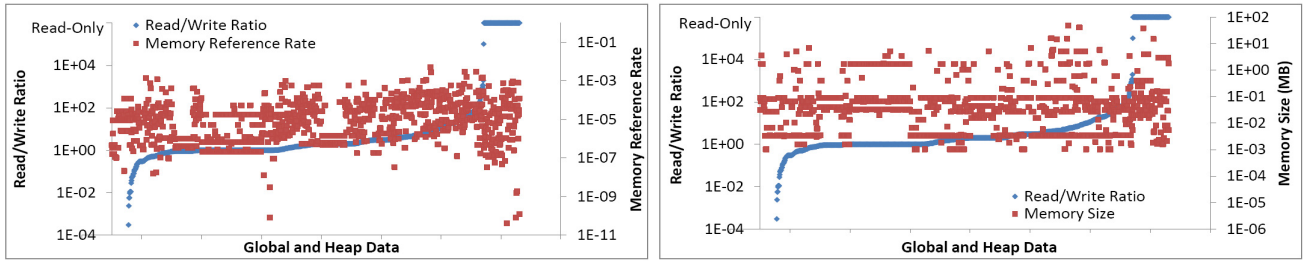

Figure 4: Read/write ratios, memory reference rates and memory object sizes for memory objects in CAM

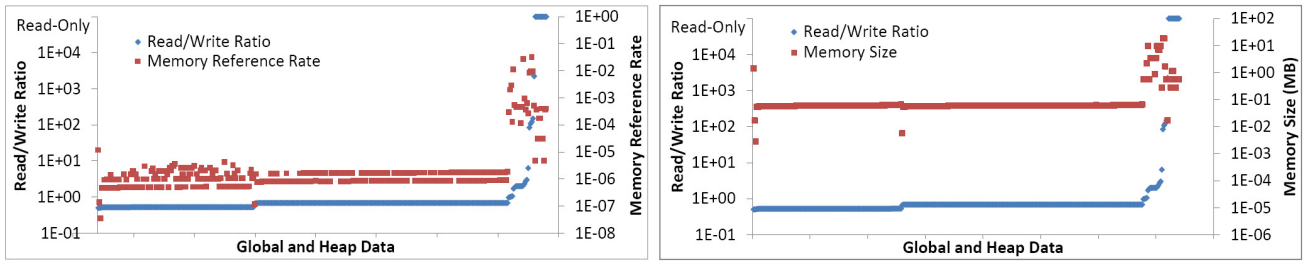

Figure 5: Read/write ratios, memory reference rates and memory object sizes for memory objects in GTC

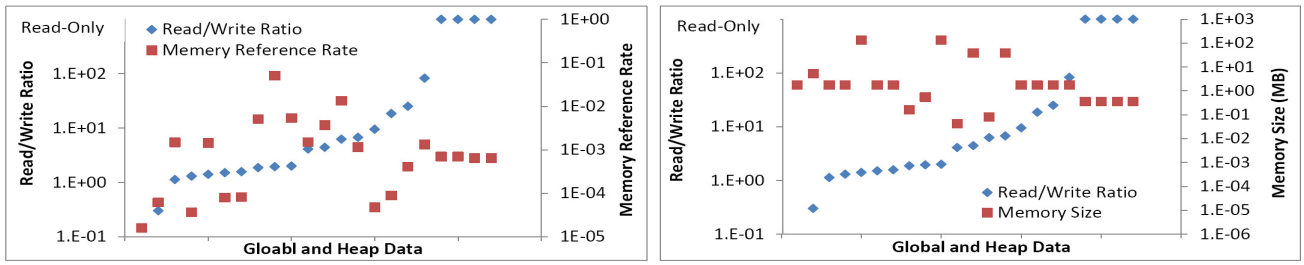

Figure 6: Read/write ratios, memory reference rates and memory object sizes for memory objects in S3D

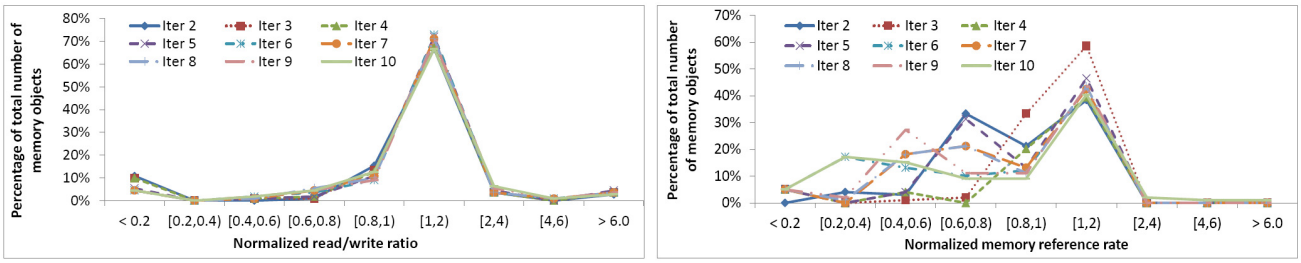

Figure 8: Read/write ratio and memory reference rate variances across the computation iterations for Nek5000

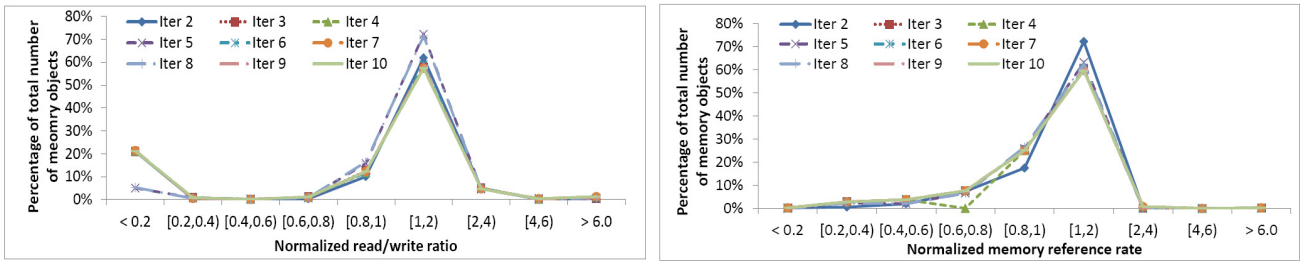

Figure 9: Read/write ratio and memory reference rate variances across the computation iterations for CAM

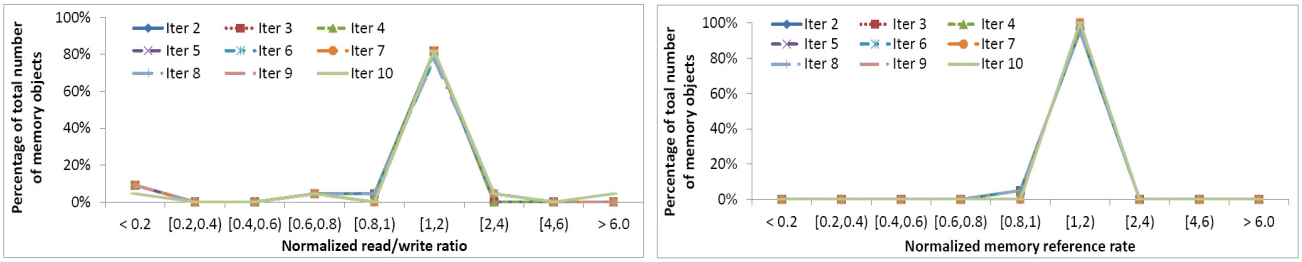

Figure 10: Read/write ratio and memory reference rate variances across the computation iterations for S3D

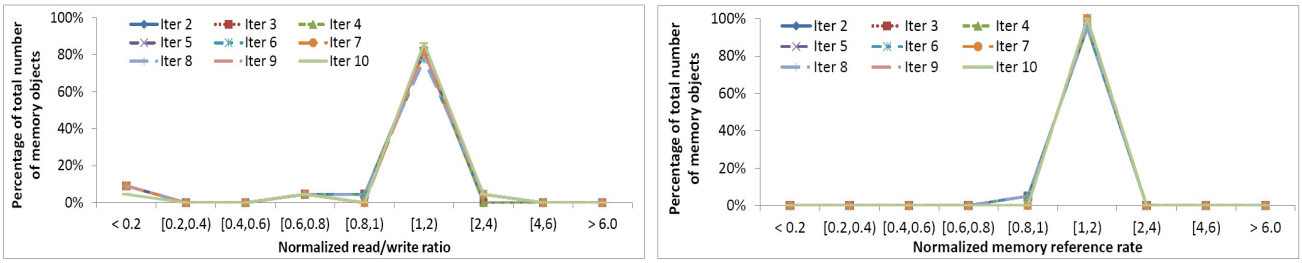

Figure 11: Read/write ratio and memory reference rate variances across the computation iterations for GTC

Application	DDR3	PCRAM	STTRAM	MRAM
Nek5000	1	0.688	0.706	0.711
CAM	1	0.686	0.699	0.701
GTC	1	0.687	0.708	0.718
S3D	1	0.686	0.711	0.730

Table VI: Normalized average power consumption

we observe at least 27% power saving. Since our simulation gives a power consumption upper bound, more power saving should be expected with a real device. Furthermore, we notice that STTRAM and MRAM have slightly larger power consumption than PCRAM. Intuitively they should have smaller power consumption. However, given the fact that the two memories have smaller access latencies, their memory bandwidth usage are larger than PCRAM. Their memory systems are more loaded on average, which leads to their higher power consumption. We expect their energy consumption will be smaller than PCRAM, given the better performance on these two memories.

E. Performance Sensitivity

In this section, we present our preliminary simulation results for two applications. To save simulation time for the time-consuming full system simulation, only one iteration of the main computation loop (or one time step) for one task is simulated. We use the memory latencies (Table IV) corresponding to DRAM and other three representative NVRAMs for our simulation. The results are shown in Figure 12. We find that the applications tolerate well the longer memory latencies. When the memory latency is increased by 20% (i.e., MRAM with 12ns), the performance loss is negligible. When the memory latency is doubled (i.e., STTRAM with 20ns), the performance loss is less than 5% for both applications. But if the memory latency is really large (i.e., PCRAM with 100ns), the performance loss can be as high as 25%.

These results are interesting. STTRAM memory, although its access latencies may not be able to compete with traditional DRAM, does not affect significantly the performance of the simulated applications. For NVRAMs with significantly longer access latencies than DRAM, a hybrid memory design or other architectural innovation must be applied to avoid performance loss.

VIII. Related Work

Using NVRAM for HPC. Emerging NVRAM provides new opportunities for high performance computing. Caulfield et al. [22] evaluate several approaches to integrate NVRAM into HPC systems. They explore several options for connecting solid-state storage to the host system and

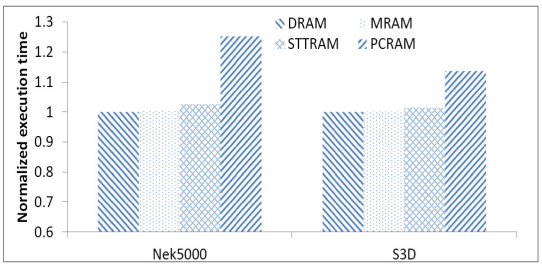

Figure 12: Time simulation results

find different system organizations lead to different performance trade-off. prototype data intensive supercomputer named DASH with SSD. They show that DASH is highly competitive by performance, cost, and power. With scientific applications from graph theory, biology and astronomy, they show that as much as two orders-of-magnitude speedup can be achieved with this special NVRAM. Stan and Shen [23] evaluate scientific I/O workloads on SSD. They find that SSD only provides modest performance gains due to write-intensive nature of many scientific workloads. They also observe that the concurrent I/O may significantly affect SSD performance, depending on specific implementation features.

Unlike the above work that evaluates the system at the granularity of the whole application, our work studies the applications characters at very fine granularity. Scientific applications can have diverse access patterns at different phases. Investigating them at fine granularity exposes more opportunities for NVRAM.

Design hybrid memory system using NVRAM. Because of existing problems in NVRAM, having a hybrid memory system by combining NVRAM and DRAM has been studied to leverage benefits of both memory systems. Ramos et al. [3] propose a hardware-driven page placement policy for hybrid memory systems. In particular, they rely on the memory controller (MC) to monitor popularity and write intensity of memory pages. MC then migrate pages between DRAM and PCM such that performance-critical pages and frequently written pages are placed in DRAM, while non-critical pages and rarely written pages in PCM. They also ask OS to periodically update its mapping of virtual pages to physical frames based on the MC translation table to guarantee correctness of the system. Qureshi et al. [2] proposed a hierarchical hybrid memory system. They use DRAM as a buffer logically placed between processor and PCM main memory. The DRAM buffer is implemented as a set-associative cache managed by MC. All data blocks take space in PCM. PCM is accessed only when the DRAM buffer eviction or buffer miss happens. Zhang et al. [24] places PCM and DRAM in a flat address space and store all pages in PCM initially. Like [3], they rely on MC to monitor access patterns. Unlike [3], they rely on OS to manage pages and treat DRAM as an OS-managed write partition. Over time, the frequently written pages are placed into DRAM to avoid long access latency of PCM.

Our work also targets at a hybrid memory system. Our work provides important supplementary to the above work. We demonstrate that using a hybrid memory system is a feasible solution for HPC, given a large amount of opportunities we found from scientific applications.

Re-design system interfaces and OS to leverage NVRAM. The unique characters of NVRAM call for rethinking system designs to make them suitable for NVRAM. Condit et al. [8] design a file system that uses a technology called short-circuit shadow paging to allow copy-on-write at fine granularity and atomically commit small changes at any level of the file system tree. This technology improves system reliability and performance by leveraging persistency and byte-addressable of NVRAM. Ouyang et al. [7] propose a new storage primitive called atomic-write to support NVRAM. They batch multiple I/O operations into a single logical group that will be persisted as a whole or rolled back upon failure. They move write-atomicity primitive out of user space libraries and operating system implementations into a NVRAM device specific file system layer, such that the amount of traditional operations required at the applications and file system layers are significantly reduced. Coburn et al. [6] propose a fast persistent data structure implemented on top of NVRAM. They point out several new types of programming errors that arise only in the persistent NVRAM data structure. They also propose corresponding solutions to provide safe access to persistent objects.

In contrast to the above work, our work focuses on the applications instead of low level file systems or system libraries. We describe how application data structures might easily exploit a hybrid NVRAM memory architecture to improve energy efficiency. This evidence provides the motivation to leverage a hybrid NVRAM memory system design in future HPC architectures, and to begin exploring how to co-design applications to best use this new capability.

IX. Conclusions

This paper performs a detailed analysis of memory access patterns for several real large-scale scientific applications. By performing the analysis, we intend to investigate opportunities for a hybrid DRAM-NVRAM system from the application view, which is not explored in previous NVRAM research. To carry out the analysis, we develop a binary instrumentation tool, NV-SCAVENGER. It monitors the access patterns of memory objects in stack, heap, and global and statistically present results. We also use our NVRAM power simulation tool based on DRAMsim to specifically study the implication of PCRAM on power consumption. We investigate performance sensitivity of those applications to different memory latencies of NVRAM using a full system simulator. Our results reveal a lot of opportunities for using NVRAM in HPC.

X. Acknowledgments

The paper has been authored by Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract #DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy. We are especially grateful to Paul Fischer (Argonne National Laboratory) and Ihor Holod (University of California, Irvine) for their insightful suggestions to DOE applications.

References

[1] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA Information Processing Techniques Office, Tech. Rep., 2008.

[2] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *Proc. of the Int. Symp. Computer Architecture*, 2009.

[3] L. Ramos, E. Gorbatov, and R. Bianchini, "Page Placement in Hybrid Memory Systems," in *Proc. Int. Conf. Supercomputing*, 2011.

[4] W. Zhang and T. Li, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architecture," in *International Conference on Parallel Architecture and Compilation Techniques*, 2009.

[5] B. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecturing Phase Change Mmemorhy as a Scalable DRAM Architecture," in *Int. Symp. Computer Architecture*, 2009.

[6] J. Coburn *et al.*, "NV-Heaps: Making Persistent Objects Fast and Safe With Next-Generation, Non-Volatile Memories," in *Proc. Int. Conf. Architectural Support for programming languages and operating systems*, 2011.

[7] X. Ouyang *et al.*, "Beyond Block I/O: Rethinking Traditional Storage Primitives," in *Proc. Int. Symp. High Performance Computer Architecture*, 2011.

[8] J. Condt *et al.*, "Better I/O Through Byte-Addressable, Persistent Memory," in *ACM Symp. Operating System Principles*, 2009.

[9] M. H. Kryder and K. Chang Soo, "After hard drives: What comes next?" *IEEE Trans. Magnetics*, vol. 45, no. 10, pp. 3406–3413, 2009.

[10] S. Yu and H. Wong, "A Phenomenological Model for the Reset Mechanism for Metal Oxide RRAM," *IEEE Electron Device Letters*, vol. 32, no. 12, 2010.

[11] H. Jeong, Y. Kim, J. Lee, and S. Choi, "A low-temperaturegrown TiO2 based devcie for the flexible stacked RRAM application," *Nanotechnology*, no. 11, 2010.

[12] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architecture," in *Int. Symp. Computer Architecture*, 2009.

[13] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying locality in the memory access patterns of hpc applications," in *Proc. SC 2005*, 2005, pp. 50–50.

[14] C. Luk, R. Cohn, R. Muth, and et. al, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *ACM SIGPLAN Conf. Programming Language Design and Implementation*, 2005.

[15] T. Sherwood and E. P. et al., "Automatically Characterizing Large Scale Program Behavior," in *Proc. Int. Conf. Architectural Support for programming languages and operating systems*, 2002.

[16] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," *Computer Architecture Letters*, no. 1, 2011.

[17] M. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *Int. Symp. Performance Analysis of Systems and Software*, 2007.

[18] P. Fischer, J. Lottes, and S. Kerkemeier, "nek5000 Web page," 2008, http://nek5000.mcs.anl.gov.

[19] N. C. for Atmospheric Research, "CAM Web page," 2011, http://www.cesm.ucar.edu.

[20] Z. Lin, "GTC Web page," 2010, http://phoenix.ps.uci.edu/GTC.

[21] E. Hawkes and R. Sankaran, "Direct numberical simulation of turbulent combustion: fundamental insights towards predictive models," *Journal of Physics: Conference Series*, no. 15, 2005.

[22] A. M. Caulfield, J. Coburn, and et. al, "Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing," in *Proc. SC 2010*, 2010.

[23] S. Park and K. Shen, "A Performance Evaluation of Scientific I/O Workloads on Flash-Based SSDs," in *Workshop on Interfaces and Architectures for Scientific Data Storage*, 2009.

[24] W. Zhang and T. Li, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architecture," in *International Conference on Parallel Architecture and Compilation Techniques*, 2009.