Are We Witnessing the Spectre of an HPC Meltdown?

Verónica G. Vergara Larrea, Michael J. Brim, Wayne Joubert, Swen Boehm,
Matthew Baker, Oscar Hernandez, Sarp Oral, James Simmons, and Don Maxwell
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory, Oak Ridge, TN 37830
Email: {vergaravg,brimmj,joubert,boehms,bakermb,oscar,oralhs,simmonsja,maxwellde}@ornl.gov

***Abstract*—We measure and analyze the performance observed when running applications and benchmarks before and after the Meltdown and Spectre fixes have been applied to the Cray supercomputers and supporting systems at the Oak Ridge Leadership Computing Facility (OLCF). Of particular interest is the effect of these fixes on applications selected from the OLCF portfolio when running at scale. This comprehensive study presents results from experiments run on Titan, Eos, Cumulus, and Percival supercomputers at the OLCF. The results from this study are useful for HPC users running on Cray supercomputers, and serve to better understand the impact that these two vulnerabilities have on diverse HPC workloads at scale.**

***Keywords*-high performance computing; performance analysis; security vulnerabilities; Spectre; Meltdown;**

I. Introduction

In early January 2018, two major security vulnerabilities present in the majority of modern processors were announced. The Spectre [1] and Meltdown [2] vulnerabilities take advantage of hardware speculative execution to expose kernel or application data stored in memory or caches to other applications running on the same system, bypassing security privileges and permissions. Fixes have started to be released from the different vendors as microcode and firmware updates. Mitigations have also been released in the form of operating system kernel patches.

Current reports indicate that processors from Intel, ARM, IBM, Qualcomm, and AMD are impacted to various degrees. Beyond posing a serious security risk, initial reports show that available fixes for Meltdown can impact performance anywhere from just a few percent to nearly 50% depending on the application. The highest impacts are expected in applications that perform a large number of system calls (e.g., I/O operations). Furthermore, the performance impacts are expected to vary depending on the parallel file system used (e.g., GPFS vs Lustre).

In this work, performance results obtained from user facing systems, including Cray supercomputers at the Oak Ridge Leadership Computing Facility (OLCF) are presented. This study includes experiments conducted on Titan (Cray XK7), Eos (Intel Sandy Bridge Cray XC30), Cumulus (Intel Broadwell Cray XC40), and Percival (Intel Knights Landing Cray XC40) supercomputers, as well as supporting systems. The applications and benchmarks used for these experiments are selected from the acceptance test plans for each system. In order to provide a comprehensive view of the performance impact on HPC workloads, the test set was expanded to include applications representative of the current OLCF portfolio. We measure and analyze the performance observed when running applications and benchmarks before and after the Meltdown and Spectre mitigations have been applied. Of particular interest to this work is the effect of these fixes on applications running at scale.

If confirmed, a significant performance hit would have enormous impact to the high performance computing (HPC) community. Since the majority of OLCF's users are focused on scientific simulation and modeling, a performance degradation equates to reduced scientific output for a given allocation of computing time. As a result, users may not reach their planned research milestones. Furthermore, a widespread performance impact may justify reexamining community measures of performance, such as the well known Top500 list [3] which ranks supercomputers all over the world, after patches are applied.

Notice of Copyright. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

CCPE Special Issue. This paper has been accepted and will be published as an article in a special issue of Concurrency and Computation Practice and Experience on the Cray User Group 2018.

II. Background

A. Understanding Spectre and Meltdown

Processors can at times be natural procrastinators. Over the years techniques were developed to limit the impact when the processor stalls. All processors are governed by a clock which dictates how much of an instruction can be completed. Since most instructions cannot be completed in a single clock cycle, the instruction is broken into different

independent stages that can each be executed in one clock cycle. That is referred to as instruction pipelining.

To maximize the processor's performance, several microinstructions are handled in parallel. The depth of the pipeline tells us how many microinstructions are handled in parallel. Typically x86 processor can have pipelines that are 32 deep. This approach has a great performance benefit for the processor as long as the instruction is completed before the next one. When this condition is not met, it is referred to as a hazard. Speculative execution is one class of hazard. The exploits covered in this paper take advantage of overlooked design details of this optimization.

Spectre and Meltdown are different variants of a speculative execution vulnerability that impacts the majority of modern computer processors on the market [4]. Speculation occurs when the information needed for the next instruction is not known, but the processor makes an educated guess. For example, branch speculation occurs when the processor guesses the result of a branch condition and speculatively executes future instructions at the predicted branch target. Another form of branch speculation is when the processor guesses the target address of an indirect branch. If the target is correctly predicted, performance can be greatly improved by speculative execution. When an incorrect prediction (i.e., mispredict) happens, the processor needs to roll back the execution state. The processor discards the speculated program state to prevent user-visible effects of the speculative execution. Vulnerabilities occur when speculated program state is not properly discarded. Exploit vectors include any processor hardware that is not fully and securely restored to its preprediction state, with typical targets being for example L3 caches (cache pollution) and I/O ports.

There are three documented vulnerability variants, each having been assigned a separate Common Vulnerabilities and Exposures (CVEs) number. CVE-2017-5753 (V1) and CVE-2017-5715 (V2) are the two variants known as Spectre, and CVE-2017-5754 (V3) is the variant known as Meltdown. In January 2018, Kocher et al. described Spectre [1], a new type of microarchitecture attack that exploits speculative execution by forcing the processor to execute instructions outside the correct path of execution of a program. Meltdown [2] does not rely on branch prediction, but exploits some modern processors that perform multiple memory references without page table permission while executing speculatively. As a result, Meltdown permits a user process to read physical memory, including protected kernel memory, by exploiting side-channel information. Lipp et al. describe this novel attack as a vulnerability that allows an attacker to bypass memory isolation, thus granting access to information that should only be accessible by the operating system [5]. While Spectre impacts most processors with out-of-order execution engines, including those manufactured by AMD, ARM, IBM, and Intel, Meltdown primarily impacts Intel processors.

Meltdown and Spectre exploit the fact that L3 cache is not correctly cleaned up after misprediction. Using L3 cache as an exploit vector was popularized by Yuval Yarom and Katrina Falkner [6]. Spectre and Meltdown do not rely on misbehavior of the L3 cache. Rather, they use it as a side channel to exfiltrate data. They leverage the fact that data may be loaded into L3 cache based on bad predictions and not cleared from the cache upon misprediction. Thus, a separate process may use processor-specific instructions to access the cache contents directly to find which memory values are present and in turn deduce the values used in the bad speculation.

B. Spectre V1: Condition misprediction

Spectre V1 relies on mispredicting a conditional branch. The example given in the Spectre paper uses an array boundary check. In normal execution, the array boundary will not be exceeded, thus the processor will speculate that the branch doing the boundary checking will be true.

```
/* In this example array_a is an array that is cache hot.
 * array_a does not contain the data targeted for leaking, but
 * index+array_a will point to the memory the attacker wishes to exfiltrate.
 * The variable index and array_b are attacker controlled;
 * array_a_size and all indexes of array_b are flushed from cache;
 */

 if (index < array_a_size) {
  array_value = array_b[array_a[index]*256];
}
```

When executing the above code with the cache in the state as described, the processor will stall, waiting for array_a_size. The processor will then speculatively fetch array_a[index] on the assumption that index $<$ array_a_size evaluates to true. Index is an offset large enough to point to another array, one with secret data (e.g., an encryption private key). The speculative execution engine will then load into L3 cache a value from array_b based on a calculation using values from array_a.

After speculative results are discarded, an attacker cannot directly inspect results of access to array_b. However, array_b had a memory load performed, making its value present in L3 cache. The attacker can then scan array_b, time the access of each index, and reconstruct the value from the out-of-bounds memory access to array_a based on whether the access to array_b was from main memory (slow) or L3 cache (fast). Example code is available in Spectre paper [1].

C. Spectre V2: Indirect branch misprediction

Spectre V2 is an exploit based on mispredicting the value of a branch from a function pointer. The use of function pointers built at runtime is common in C++ applications and modular frameworks such as OpenMPI. The function pointers are resolved by the time the program is run and don't change through the life of the program. Modern processors have an indirect branch predictor based on this assumption, but no processor vendor documents the operation of the predictor. The Spectre authors have tested how the predictor behaves, looking for ways to exploit it. Of particular note,

the predictor seems to care about destination address and little else. To exploit this behavior, the processor is trained to believe that a particular memory address is overwhelmingly the target of indirect branches. The predicted targets are not shared between cores, so all processor cores must be trained individually.

Useful exploitation of Spectre V2 has much in common with an exploit technique called Return Oriented Programming (ROP). This technique is used when new executable code cannot be loaded into a running process for the purposes of the exploit. Instead, an attacker will identify blocks of assembly code in the target application that can be chained together to construct a new program using existing executable code. It relies heavily on return instructions, a corrupted process stack and return pointers being used to guide the process on an unintended execution path.

In Spectre V2 the attack depends on constructing an attack program from the large amount of executable code that will be loaded with a process in dependent shared libraries. Before the victim program is run, the attacker launches a process that will train the processor's branch predictor. This program does nothing more than run an indirect branch in a loop to make the processor believe that all indirect branches go to the attacker's chosen memory location. The victim program will have the value of the indirect branch flushed from cache, causing the processor to execute speculatively based on the predictor conditioned by the attacker. The speculative program will then have an objective similar to what was documented for Spectre V1, attempting to load data from an attacker controlled array into L3 cache in a way that may be inspected by a separate program. The program that is run speculatively has a large advantage over ROP attacks, since its state will be discarded when the indirect branch is resolved, leaving the attacker with more options for exploit code that will not cause the victim program to crash. After the speculative branch is discarded, the attacker can scan L3 cache similarly to Spectre V1 to obtain leaked information.

D. Spectre V3 / Meltdown: Illegal cache load

The most serious of the speculative execution exploits is Meltdown. Meltdown exploits the fact that some processors do not check the memory access privilege of program code run speculatively. Affected processors will wait until the access instruction is retired from the pipeline to do this check. In the Meltdown paper, the attack program spawns two threads. The first thread is a spy program that allocates an array to be shared between the two threads, flushes the contents of that array from cache, and creates the second thread. The second thread executes an illegal operation (such as a segmentation fault). However, that instruction is not in cache, so the processor will continue to speculatively run instructions after the illegal instruction until that stall is resolved. In the speculative code, access to a protected memory region may be performed. For example, in the Linux kernel, all physical memory is mapped into the kernel address space. On x86 processors, access to these memory addresses by user programs is prohibited by the page table. However, with Meltdown the access to kernel memory is only executed speculatively, since the thread doing the memory access will be terminated by the illegal instruction before the memory access instruction is retired. In the speculative part of the thread, the illegally read memory will be used to compute an index and access to the attacker-controlled array, same as in Spectre V1. After the second thread is terminated, the first thread will scan the array, looking for which index is loaded from L3 cache rather than from memory. In this way, any process may scan the entire memory address space regardless of privileges. This exploit works even when the program is hosted in a virtual machine, breaking the isolation of the virtual machine to scan all of the host system's memory.

E. Spectre and Meltdown Mitigations

Shortly after Spectre and Meltdown were made public, Cray released Field Notice #6219f [7] detailing how their system offerings were impacted by each of the variants. Table I summarizes how each system is affected.

Table I
SPECTRE AND MELTDOWN IMPACT ON CRAY ARCHITECTURES

Cray System	Processor	Spectre		Meltdown
		CVE-2017-5753	CVE-2017-5715	CVE-2017-5754
XC	Intel x86	Yes	Yes	Yes
XC	Intel KNL	Yes	Yes	Yes
XE, XK	AMD x86	Yes	Yes (1)	No
XE, XC	NVIDIA GPU	Yes (2)	Yes (2)	No
Envoy	ARM Cavium TX	No	No	No
XC	Intel KNC	No	No	No

III. EXPERIMENTAL DESIGN

A. OLCF Systems

The OLCF has several supercomputers and supporting systems available to the user community. This study focuses only on the Cray systems available at the OLCF as well as their associated file systems. The following systems were used to determine the performance impact of Spectre and Meltdown patches:

1) Titan: Titan is currently the OLCF's flagship supercomputer. Titan is a Cray XK7 system with 18,688 compute nodes, each equipped with a 16-core AMD Opteron, one NVIDIA K20x GPU, and 32 GB of RAM. Titan debuted as the #1 system in the November 2012 Top500 list [8].

2) Eos: Eos is a Cray XC30 system with 736 compute nodes, each equipped with two 8-core Intel Xeon E5-2670 processors and 64 GB of RAM. Eos uses Cray's proprietary Aries interconnect in a Dragonfly topology [9]. At the OLCF, Eos is mainly utilized for post-processing and analysis of data produced on Titan.

3) Percival: Percival is a Cray XC40 system with 168 compute nodes, each equipped with a 64-core Intel Xeon Phi 7230 processor, also known as "Knights Landing" (KNL), and 110 GB of RAM. Like Eos, Percival uses Cray's proprietary Aries network configured in a Dragonfly topology [10]. The Percival system is primarily used for performance portability efforts as it offers OLCF users access to Intel's KNL processors. Percival was selected for this study to assess the performance impact of the vulnerabilities on the Intel Xeon Phi family of architectures.

4) Cumulus: Cumulus is a 112-node Cray XC40 system with two 18-core Intel Broadwell processors and 128 GB of RAM per node. Cumulus also uses the Aries interconnect [11]. The Cumulus system is primarily dedicated to Biological and Environment Research (BER) projects. Cumulus was selected for this study because it contains a more modern Intel processor than Eos.

B. Applications and Benchmarks

In order to provide a comprehensive view of the impact of the Spectre and Meltdown patches on OLCF systems, a wide range of applications and benchmarks were utilized. This set of tests include applications that are compute-, I/O- and network-bound. The following codes were used for this study and are representative of OLCF workloads:

1) HPL: High-performance Linpack (HPL) [12] is a weak-scaling compute-intensive benchmark for evaluating HPC systems using dense linear algebra. HPL gives a measure of peak floating-point operation performance, uses memory access patterns that are efficient for modern cache hierarchies, and has relatively little distributed communication. As such, it serves as a reference for the impact to compute-bound computations.

2) HPCG: High-Performance Conjugate Gradient (HPCG) [13] is a weak-scaling benchmark for evaluating HPC systems using multigrid preconditioned conjugate gradients. It mimics the full behavior of scientific simulations based on sparse linear algebra, nested grids, and communication patterns using both local halo exchanges and global reductions. Compared to HPL, HPCG is more sensitive to memory accesses and MPI communication latencies, and its peak performance correlates more closely with applications using sparse data.

3) OSU Microbenchmarks: The OSU Microbenchmarks (OMB) [14] are a comprehensive suite of benchmarks measuring latency and bandwidth for MPI point-to-point, one-sided, and collective operations across a wide range of message sizes, as well as MPI application setup and teardown times. OMB is thus useful for studying the impact of the patches to interprocess communication in isolation, which may help in understanding the results of other benchmarks and applications.

4) SPEC OMP2012: SPEComp is a benchmark suite for parallel computers. The suite consists of 14 applications covering science and engineering codes. The benchmarks are based on the OpenMP 3.1 standard.

5) GTC: GTC [15] is a gyrokinetic toroidal plasma fusion code used to model fusion reactors. It has been used for science simulations for some time at the OLCF. Its well-understood performance characteristics have made it an effective candidate for benchmarking and system testing for OLCF systems. It scales to large node counts and has a range of computational motifs used by OLCF codes, including particles, unstructured grids and sparse linear algebra. The version used here uses MPI, CUDA, OpenACC and the PETSc solver package.

6) S3D: S3D [16] is a massively parallel direct numerical simulation (DNS) solver developed at Sandia National Laboratories [16] used for combustion modeling. S3D uses MPI and is primarily written in FORTRAN. Different versions of S3D are used at the OLCF to exercise different parts of the software stack. In this study, compute and I/O intensive cases were used.

7) LAMMPS: The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) code is a classical molecular dynamics package developed at Sandia National Laboratories. It is a widely used application, particularly in the fields of material science and biophysics. LAMMPS is open source, uses MPI, and has support for several types of architectures and accelerators including GPUs, Intel KNC, and Intel KNL [17], [18].

8) NWCHEM: NWCHEM is a widely used open-source application used for large-scale simulations of chemical and biological systems. NWCHEM was developed at Pacific Northwest National Laboratory [19] and uses the Global Array (GA) toolkit [20] to provide support for distributed data algorithms which are commonplace in computational chemistry. NWCHEM supports different architectures and accelerators including GPUs. This application was chosen not only due to the fact that it is a prominent code run at the OLCF, but also because it is network intensive.

9) LSMS: The Locally Self-consistent Multiple Scattering (LSMS) code was introduced by Wang et al. as a novel linear order method to calculate the electronic structure of large systems [21]. LSMS is highly parallel, uses MPI, and was primarily written in Fortran. Newer versions of LSMS have added support for the Wang-Landau/LSMS hybrid method [22] and provide support GPU support. LSMS was chosen because it is a highly scalable, compute intensive code.

10) MILC: The MIMD Lattice Computation (MILC) code is a quantum chromodynamics package developed by the MIMD Lattice Computation collaboration [23]. MILC was used as one of the benchmarks in the Trinity / NERSC-8 procurement [24]. This application was chosen because it is optimized for the Intel Knights Landing processor and was used for Percival's acceptance. MILC was only used for experiments on Percival.

C. I/O intensive Applications and Benchmarks

Typically the MPI applications deployed by users to generate data for their research use checkpoints. These checkpoints tend to perform I/O in burst, but in order to better understand the impact of the Spectre and Meltdown mitigations, applications designed to generate synthetic I/O workloads were included in our evaluation. The impact due to I/O is of particular interest due to the high levels of syscalls such as write() and read() being performed. A very measurable impact of the meltdown fix can be observed due the context switch between kernel space and user space due to the syscalls performed. I/O-lightweight scientific application can give a false impression of the impact of the Meltdown fix.

1) IOR: The benchmark chosen to simulate bulk I/O for read and write calls with generated data was IOR [25]. This utility was created to measure file system bulk I/O performance at both the POSIX and MPI-IO level. IOR is flexible in that one can control whether the I/O is sequential or random, file per process or single shared file, as well as various other settings. For the case of our testing we attempted to mimick the typical user's I/O pattern. This is typically either a single shared file or file per process, so both conditions were tested.

2) mdtest: Metadata operations can have an even larger impact on users than bulk I/O. The application chosen for profiling the metadata operations behavior for before and after the Spectre and Meltdown fixes was mdtest. For both files and directories, standard metadata operations of creation, deletion, and collecting stats were performed. These are the most common operations performed by our typical user.

3) simul: Besides performance, general POSIX compliance is important. While simul is used mainly to demonstrate that recent changes in Lustre or a kernel don't break the behavior expected of a POSIX compliant file system, it also reports how long performing the standard POSIX operation took.

IV. Results

In this section, we report the results of our experiments on each OLCF system. We include results from single node runs up to full-system jobs. Where relevant, we include two node results to help observe effects when applications first require internode communication. Because specific applications use slightly different node counts, we group them based on the approximate percentage of the full system size.

A. Titan - Cray XK7

Figure 1 summarizes our results on Titan for various job sizes, from a single compute node up to full-system jobs. Performance after patches is reported normalized to the before patch measurement. Values less than 1.0 indicate a performance degradation.

1) HPCG: On Titan, since no Intel MKL optimized version of HPCG was available, we used the 3.0 release of the code from the website [13], compiled with the Cray compiler v8.6.4. The build configuration targeted only the Opteron CPUs. HPCG results were gathered using ten runs in a single job allocation per job size, and the arithmetic mean of the reported floating point performance was calculated. As shown in Figure 1, HPCG showed no degradation on Titan after patching. In fact, performance up to 9,000 nodes showed modest improvements of up to 4%.

2) OSU Microbenchmarks: We ran OMB on Titan before and after patches from two nodes up to 9,000 nodes (i.e., roughly 50%). Point-to-point and one-sided tests were run on two nodes, with one MPI rank per node. Collectives and multiple pairs of point-to-point tests were run on job sizes using more than two nodes. For collectives, four MPI ranks were placed on each node. All-to-all and non-blocking send-receive tests were not run. All tests showed unpredictable variability, both positive and negative, between before and after results across all message sizes. We attribute the variability to network contention and job placement effects on Titan's Gemini mesh network. Our 25% and 50% job sizes also experienced active throttling of the network, as reported by aprun messages. Due to this high variance, no general performance or scaling trends were observed.

3) SPEC OMP 2012: Figure 2 shows the runtime of SPEC OMP 2012 tests normalized to the results on Titan before patching. Numbers below 1.0 indicate performance improvement, whereas a number above 1.0 indicates a regression. Results are reported for the Cray and PGI compilers. The impact of the patches to SPEC OMP appears to be minimal. We observe runtime variations in the range of up to 7%.

4) GTC: For GTC on Titan, the performance difference between before-patching and after-patching was less than 2% with no consistent winner. Except for an outlier, this figure is roughly on a par with standard deviation within a single series of runs. We conclude there is little to no significant impact on GTC performance between the two run environments.

5) LSMS: LSMS was built on Titan using the GCC 4.9.3 programming environment and with GPU support enabled. Test cases were run up to a quarter of the total size of the system. The arithmetic mean of the self-consistent field (SF) loop time was used to measure the performance of LSMS over multiple iterations within the same job allocation. Results show minimal performance variation across small node counts and up to 4% performance improvement when running on 4,500 nodes. It is worth noting that the results obtained with LSMS were highly sentitive to the current workload on the system. The impact of the Spectre and Meltdown patches appears to be minimal.

6) LAMMPS: LAMMPS was built on Titan using the GCC 4.9.3 programming environment and the GPU

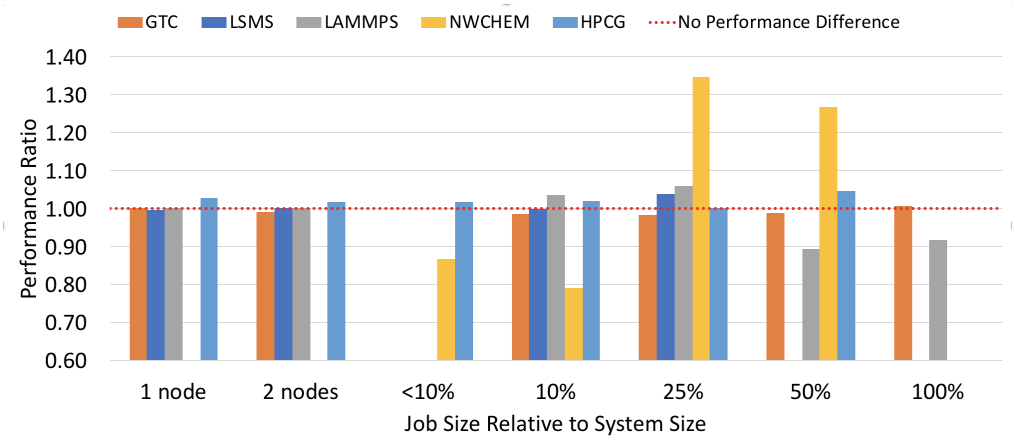

Figure 1. Normalized Application Results on Titan.

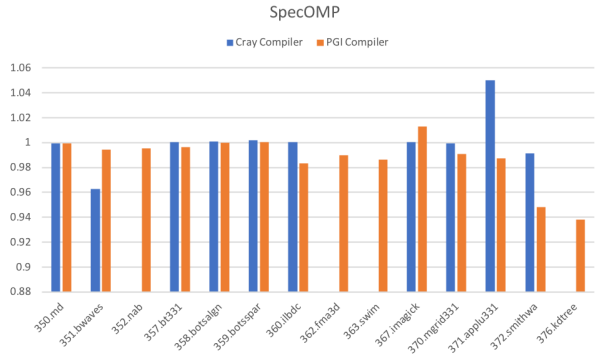

Figure 2. Normalized SPEC OMP 2012 Results on Titan.

LAMMPS package with CUDA 7.5. Test cases were run up to 18,000 nodes on Titan. Performance was measured using nanoseconds per day (ns/day) from the Rhodopsin protein benchmark simulation. While very little impact was observed for the small node count tests, results show approximately 9% of performance degradation for this particular simulation in the 9,000 and 18,000 node jobs. For smaller jobs, a slight improvement in performance is observed. However, given that at larger job sizes there is higher variability overall, the impact cannot be exclusively attributed to the patches.

7) NWCHEM: NWCHEM was built on Titan using the PGI 17.9.0 programming environment and CUDA 7.5. Test cases were executed up to half of the system size. Performance was measured using the total time as reported by the application. Due to the sensitivity of the application to the current workload on the system and activity in the network, the results measured show a high variance. For instance, in the 64 and 1,800 node tests a standard deviation of approximately 20% was observed. For the larger node counts, better performance was observed after the patches were applied. Due to the high variability of the data, further investigation would be required to identify the source of the performance difference observed.

8) I/O Harness: In addition to the set of applications selected, an isolated test was run on Titan using the OLCF's I/O test harness [26] which includes I/O intensive problems for S3D and GTC, as well as three commonly used I/O benchmarks: mdtest to measure metadata, simul and IOR to simulate specific I/O patterns. The jobs in this test suite were run up to the full system size using Spider 2, the OLCF's center-wide parallel file system running Lustre version 2.8. Overall, no significant performance difference was observed for the full system jobs.

B. Eos - Cray XC30

Figure 3 summarizes our results on Eos for various job sizes, from a single compute node up to full-system jobs. Performance after patches is reported normalized to the before patch measurement. Values less than 1.0 indicate a performance degradation.

1) HPL and HPCG: The Intel MKL AVX-optimized versions of HPL and HPCG, included with v17.0.0.98 of the Intel compilers, were used for Eos. Results were gathered using ten runs in a single job allocation per job size, and the arithmetic mean of the reported floating point performance was calculated. HPL showed no significant difference in performance after patches, with results within 2% of the before patch results. In contrast, HPCG performance showed a noticeable decline as job size was increased. However, the decline was not consistent across all ten runs, and the peak performance achieved before and after patches was nearly identical, suggesting that the slowdown may have been due to ephemeral job conditions such as network contention.

2) OSU Microbenchmarks: We ran OMB on Eos before and after patches from two nodes up to 640 nodes (i.e., roughly 90%). Point-to-point and one-sided tests were run on two nodes, with one MPI rank per node. Collectives

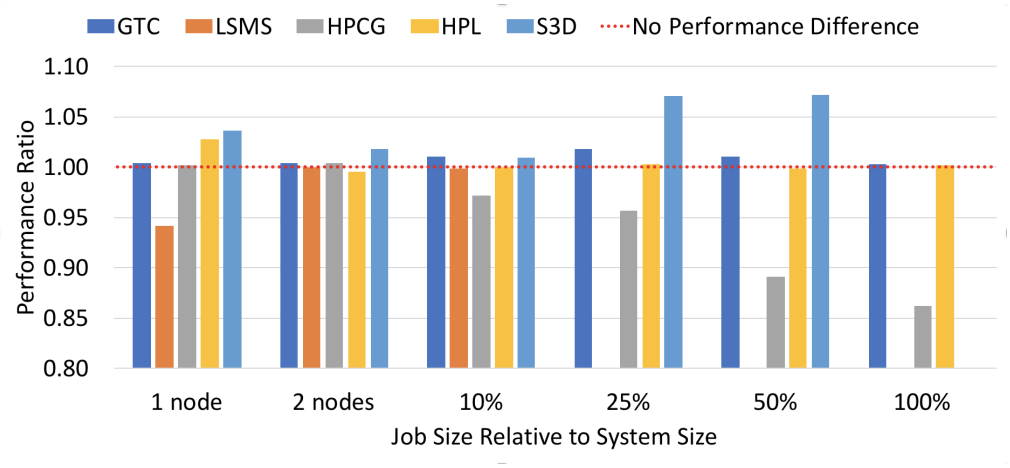

Figure 3. Normalized Application Results on Eos.

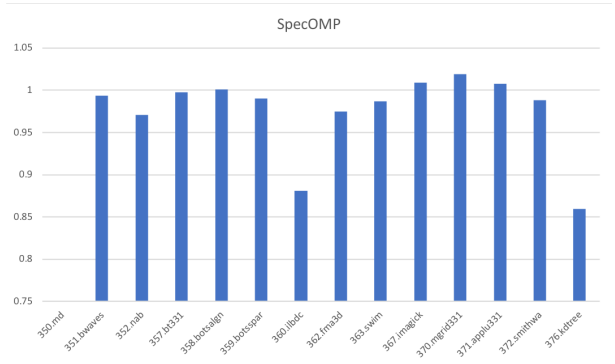

Figure 4. Normalized SPEC OMP 2012 Results on Eos.

and multiple pairs of point-to-point tests were run on job sizes using more than two nodes. For collectives, sixteen MPI ranks were placed on each node, one per physical core. All-to-all and non-blocking send-receive tests were not run. All tests showed unpredictable variability between before and after results, both positive and negative, so no general performance or scaling trends were observed. Again, we attribute this variability to network contention and job placement effects on Eos' Aries dragonfly network. Compared to Titan, however, the observed variability was much less extreme.

3) SPEC OMP 2012: Figure 4 shows the runtimes for SPEC OMP 2012 tests on Eos normalized to the results before patching. We are showing results for the PGI compiler. Numbers below 1.0 are a performance improvement, whereas numbers above 1.0 indicate a performance degradation. The patches appear to have resulted in a positive impact on the SPEC OMP results. We observe runtime reductions of up to 14%.

4) GTC: On Eos, GTC runtime is systematically slightly faster after patches compared to before patches, by at most about 2%. This is significantly higher than indicated by run-to-run variation. However, the size of impact is minor.

5) LSMS: LSMS on Eos was built using the GCC 6.3.0 programming environment. Only jobs up to approximately 10% of the system size were executed before the system was patched. At small node counts, no significant performance difference was observed.

6) S3D: S3D was built using the Intel compiler version 17.0.0.98. The arithmetic mean of time reported by the application was used to measure performance. Across job sizes up to half of the system size, performance improved from only a few percent in the smaller jobs to approximately 7% in the quarter- and half-system jobs.

C. Cumulus - Cray XC40 (Xeon)

At the time of paper submission, Cray had not yet provided functional patches for the Cumulus Cray XC40 system. An initial set of patches resulted in frequent node crashes, and had to be reverted.

D. Percival - Cray XC40 (Phi)

Figure 5 summarizes our results on Percival for various job sizes, from a single compute node up to full-system jobs. Performance after patches is reported normalized to the before patch measurement. Values less than 1.0 indicate a performance degradation.

1) HPL and HPCG: The Intel MKL Xeon Phi-optimized versions of HPL and HPCG, included with v17.0.0.98 of the Intel compilers, were used on Percival. Results were gathered using ten runs in a single job allocation per job size, and the arithmetic mean of the reported floating point performance was calculated. For both HPL and HPCG, performance was within 2% of baseline after patching, regardless of job size.

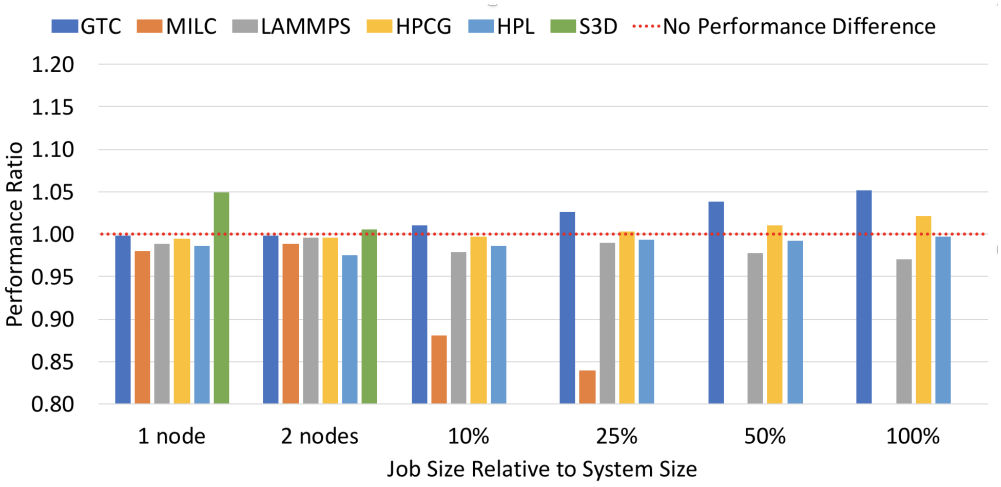

Figure 5. Normalized Application Results on Percival.

2) OSU Microbenchmarks: We ran OMB on Percival before and after patches from two nodes up to 160 nodes (i.e., roughly 100%). Point-to-point and one-sided tests were run on two nodes, with one MPI rank per node. Collectives and multiple pairs of point-to-point tests were run on job sizes using more than two nodes. For collectives, eight MPI ranks were placed on each node. All-to-all and non-blocking send-receive tests were not run. All tests showed unpredictable variability between before and after results, both positive and negative, so no general performance or scaling trends were observed. However, Percival did allow for running full-system jobs, which should mitigate effects due to contention or job placement. At full scale, latency for send-receive varied by no more than 10%, and bandwidth differences were negligible. For collectives, we observed more substantial latency variations up to 50% for small message sizes (i.e., less than 512 bytes), while larger messages reduced the latency differences to less than 20%.

3) GTC: GTC on Percival for significant node counts runs up to 5% faster after patches compared to before patches. This is significantly higher than measured run-to-run timing variation of a single series of runs. In practice this would be considered a small but significant performance improvement.

4) MILC: MILC on Percival was built using the Intel compiler version 17.0.0.98. MILC jobs were executed up to a quarter of the system size. The time reported by the application was used to measure performance. Up to the job sizes measured, the patches seem to have had a modest negative impact from 2% to approximately 16% when using 40 compute nodes.

5) LAMMPS: LAMMPS on Percival was built using the Intel compiler version 17.0.0.98 with Intel Xeon Phi support enabled. For this case, iteration time was measured in each run. Jobs executed varied from a single node to up to the full system size. Overall, all cases show 1-2% performance degradation after the patches were applied.

6) S3D: S3D on Percival was also built using the Intel compiler version 17.0.0.98. For this particular test, only single node and two node jobs were executed. Both job sizes show a minor improvement in performance after the patches were applied. In the multinode case, the improvement is less than 1%.

V. Conclusion

This study summarizes the performance impacts observed on four different Cray architectures available at the OLCF. The applications and benchmarks chosen were selected because they are commonly used at the OLCF. In addition, applications with different characteristics were chosen to get a comprehensive view of the potential impacts of Spectre and Meltdown patches.

Due to the significant security risk that the vulnerabilities present, systems had to be patched immediately following the release of the patches from Cray. In some cases, this limited the number of tests that could be conducted before a system was patched.

The results show that up to the time of submission, the overall impact of the patches available is minimal. This is an encouraging result for OLCF's user community, which can be assured that the deployed vulnerability mitigations will not decrease their scientific productivity.

Given that additional patches may be released to fully address the vulnerability on all the architectures, this study will be expanded as additional patches become available and are installed.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.

The authors would also like to acknowledge Cathy Willis (Cray) and Rich Ray (OLCF) for their assistance with this study.

REFERENCES

[1] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *CoRR*, vol. abs/1801.01203, 2018. [Online]. Available: http://arxiv.org/abs/1801.01203

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *CoRR*, vol. abs/1801.01207, 2018. [Online]. Available: http://arxiv.org/abs/1801.01207

[3] "TOP500 Supercomputer Sites," https://www.top500.org.

[4] J. Fruhlinger. Spectre and meltdown explained: What they are, how they work, what's at risk. [Online]. Available: https://www.csoonline.com/article/3247868/vulnerabilities/spectre-and-meltdown-explained-what-they-are-how-they-work-whats-at-risk.html

[5] Ellexus, "How the meltdown and spectre bugs work and what you can do to prevent a performance plummet," Ellexus Ltd., Tech. Rep., 2018.

[6] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*. San Diego, CA: USENIX Association, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom

[7] Cray, "Fn#6219f: Side-channel security vulnerability - cve-2017-5753, cve-2017-5715, cve-2017-5754," January 2018.

[8] (2012, November) TITAN: OAK RIDGE NATIONAL LABORATORY - No. 1 system in November 2012. [Online]. Available: https://www.top500.org/resources/top-systems/titan-oak-ridge-national-laboratory/

[9] (2018) Eos: System Overview. [Online]. Available: https://www.olcf.ornl.gov/for-users/system-user-guides/eos/system-overview/

[10] Percival Quickstart Guide. [Online]. Available: https://www.olcf.ornl.gov/percival-quickstart-guide/

[11] (2018) Cumulus Quickstart Guide. [Online]. Available: https://www.olcf.ornl.gov/cumulus-quickstart-guide/

[12] "HPL Benchmark," http://icl.cs.utk.edu/hpl/index.html.

[13] "HPCG Benchmark," http://www.hpcg-benchmark.org.

[14] "OSU Microbenchmarks," http://mvapich.cse.ohio-state.edu/benchmarks.

[15] W. Tang, B. Wang, S. Ethier, and Z. Lin, "Performance portability of hpc discovery science software: Fusion energy turbulence simulations at extreme scale," *Supercomputing Frontiers and Innovations*, vol. 4, no. 1, 2017. [Online]. Available: http://superfri.org/superfri/article/view/128

[16] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo, "Terascale direct numerical simulations of turbulent combustion using s3d," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, 2009. [Online]. Available: http://stacks.iop.org/1749-4699/2/i=1/a=015001

[17] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199918571039X

[18] (2018) LAMMPS. [Online]. Available: http://lammps.sandia.gov

[19] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. V. Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, "Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477 – 1489, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0010465510001438

[20] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: A nonuniform memory access programming model for high-performance computers," *J. Supercomput.*, vol. 10, no. 2, pp. 169–189, Jun. 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=243179.243182

[21] Y. Wang, G. M. Stocks, W. A. Shelton, D. M. C. Nicholson, Z. Szotek, and W. M. Temmerman, "Order-n multiple scattering approach to electronic structure calculations," *Phys. Rev. Lett.*, vol. 75, pp. 2867–2870, Oct 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.75.2867

[22] M. Eisenbach, C.-G. Zhou, D. Nicholson, G. Brown, J. Larkin, and T. C Schulthess, "Thermodynamics of magnetic systems from first principles: Wl-lsms," in *Proceedings of the 2010 SciDAC conference*, 04 2010.

[23] MIMD Lattice Computation (MILC) Collaboration. [Online]. Available: http://physics.indiana.edu/ sg/milc.html

[24] Trinity / NERSC-8 RFP. [Online]. Available: http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/

[25] "Parallel filesystem I/O benchmark," https://github.com/LLNL/ior.

[26] V. G. V. Larrea, H. S. Oral, D. B. Leverman, H. A. Nam, F. Wang, and J. A. Simmons, "A more realistic way of stressing the end-to-end i/o system," in *Proceedings of the Cray User Group 2015 conference*, 2015.