Nuclear Meltdown?
Assessing the impact of the Meltdown/Spectre bug at Los Alamos National Laboratory
LA-UR-18-24290

Joseph 'Joshi' Fullop
HPC-ENV
Los Alamos National Laboratory
Los Alamos, NM, USA
Email: fullop@lanl.gov

Jennifer Green
HPC-ENV
Los Alamos National Laboratory
Los Alamos, NM, USA
Email: jgreen@lanl.gov

***Abstract*—With the recent revelation of the Meltdown/Spectre bug, much speculation has been levied on the performance impact of the fixes to avoid potential compromises. Single node, single process codes have had very high impact estimates, but little is known on the extent to which the early patches will affect large-scale MPI jobs. Will the delays cascade into jitter vapor lock, or will the delays get lost in the waves on the ocean? Or is reality somewhere in the middle? With flagship codes and their performance driving future systems design and purchase, it is important to understand the new normal. In an investigative assessment, we research how the updates affect job performance of various benchmark codes as well as our mainstay workloads across different job sizes and architectures.**

***Keywords*-Meltdown; Spectre; LANL;**

I. Introduction

In the latter half of 2017, a major computer security vulnerability was unveiled that affected nearly all major modern processor architectures. The world of High Performance Computing (HPC) was no exception. The exploits for these vulnerabilities were published, with the best known referred to as Spectre and Meltdown (CVE-2017-5715; CVE-2017-5753; CVE-2017-5754). These attacks take advantage of various processors' speculative execution implementation. Meltdown can more easily be addressed by increasing memory isolation at the OS level. Spectre relies more on statistical odds and targeted application knowledge, which requires a much higher level of sophistication to implement. Given the nature and sensitivity of the work done on the machines at Los Alamos National Laboratory, we must operate under the assumption that we are under constant threat by infinitely sophisticated attackers.

The patches created to address these vulnerabilities have to be done at such a low level in the computational process that their impact is expected to be very broad and highly dependent on the propensity to switch to and from kernel mode. There have been ample reports and studies on performance impact of these patches on various codes and benchmarks. This paper will discuss some observed performance statistics, but will also go further into the aspects of managing these issues at a premier computing facility.

II. Testing

Some of the difficulties with inline testing is that the base system environment is in an almost constant state of flux. A run of a benchmark from a few months ago is generally not comparable to one today without caveat. Even though a machine's hardware and firmware can remain effectively the same over a given period, the environment on that machine generally changes frequently. This is a normal churn of applying bug fixes, installing new software releases and configuration modifications.

Since changes are not atomic and are generally done en-mass during a Dedicated System Time (DST), it is often difficult to attribute measured performance variations to any one individual change. It has been practice to separate multiple major system updates for many reasons. Some reasons being that there are only so many things that can be prepared and tested in a given period, and that of efficiently diagnosing fallout to the correct root cause. When more than one large change is implemented at the same time, the areas of problem investigation multiply. For example, when one system change is made, diagnosis can be fairy well focused around that change when problems arise. But when two large systems changes, A and B are simultaneously enacted on a machine. If there is a problem, investigations need to be conducted on both of the components individually, as well as potential conflicts between the two. Add another simultaneous change and the possibilities continue to expand further.

So there generally exists a balance between many individual changes and sets of multiple changes as the criticality, urgency and demand for various changes fluctuate. As much as systems administrators would like to make singular changes and test thoroughly each time, the necessity of

providing availability and high utilization rates force changes into compressed DSTs.

III. Restrictions on Disabling Patches for Testing

Since we in HPC at Los Alamos National Laboratory are in the business of providing compute cycles to the codes in support of our mission and are not purely exploratory computer scientists, there was little designated time to fully examine the phenomena presented by the Meltdown and Spectre vulnerabilities. Therefore, we tested as we could, when we could. Given the issues with parallel changes described above, the only way to have a clean experiment would be to isolate the machine and reboot into un-patched mode, run experiments, reboot patched and run the same set of experiments. This would require at least a day or two of concerted effort, that time taken away from the users.

One suggestion was to use only a dedicated set of nodes for this experimentation. Since the exploit is not a specific attack, but more of a feature that can be leveraged in many different ways, we have not opted to run in the unpatched mode. Even though we allocate nodes fully to one job at a time so there is no co-location of user codes, there could conceivably evolve an exploit for privileged escalation or accessing remnants of past jobs memory spaces. Yes, to some these attacks may seem far-fetched and even being concerned about them may seem paranoid, but when dealing with infinitely sophisticated attackers, exploits can be used to enable exploits and various attacks could be chained together to compromise a system. Allowing a known starting point or exposed feature was determined to be to high a risk.

One area of investigation that may prove useful in addressing the performance and security issues is the use of containers. The hope is that the containers may be used to isolate and insulate memory spaces in an appropriate manner. However, this is an ongoing evaluation whose results, if fruitful, will certainly be published and likely be a popular means of dealing with these vulnerabilities.

IV. Real Costs of Code Re-Writes

One of the reasons that the Meltdown and Spectre vulnerabilities get so much attention is due to its broad impact and the potential performance degradation of their fix. Many, if not most exploits are specific and directed at very targeted segments of code. Since these vulnerabilities target a CPU hardware feature, the opportunity for misuse is almost constant and therefore the mitigation has the possibility of massive performance degradation.

One of the unfortunate aspects of security exploits is that they can not be conveniently planned. Meltdown ans Spectre unlike most exploits degrade general performance of the system. So how does this impact the normal course of business at a secure computational facility? The first responsibility is to the mission. Analogous to any corporate scenario, there are deadlines and expectations. Also, codes have been designed, improved and refined over decades to match the ever evolving science behind them. So to rewrite and retune would be a Herculean effort. Just to put things into perspective, the code teams have been called upon to refactor our codes every time we decide that the bleeding edge of technology is the way of the future and we architect a massive new machine based on some emerging technology.

Every time a major change in computing architecture, technology or methodology comes about, a significant refactoring process takes place. So the question at the base of the Meltdown and Specter vulnerabilities is 'What degree of effort is necessary to address them?' So, let's examine from a large scale perspective. Let's pose the question of 'What level of performance impact would cause a significant derailment of the status quo?' Since the refactoring of codes to hardware changes happens on a fairly regular basis at Los Alamos National Laboratory, there exists a bit of a cycle refactor and refine.

Let's examine this cycle from an actuarial perspective. The lifespan of a leading edge computational resource appears to be anywhere from five to seven years. So if a performance impacting exploit were to surface 5 years from the planned retirement of a machine, a 20% impact would support up to a 1 year effort or commitment of resources at break even. So the initially estimated performance impact assessments of anywhere from 10 to 50% warranted evaluation. We had recently installed a 9000+ node segment of Intel's Knights Landing architecture, and our code teams were still in the settling-in phase of this machine.

It is important to understand the scientific simulation evolutionary processes. These include various tasks of validating that the new code produces the same mathematical results. Beyond that there is a verification process that the new codes are correct and representative and in alignment with experimental data from recent and past measured experiments. Which, in turn, requires additional computational hours on the same machines where we're trying to *alleviate* excess load. And further, a certification process must be undertaken to validate developer oversight. This process is hopefully similar to any other scientific computational process out there.

If the overall performance impact were measured to be persistently greater than around 25% to the end of the machine's life of 5 years, investing in 1 year of developing code that would fully eliminate the performance hit would be considered a break-even. In this scenario, the first year is spent developing the solution and recouping the performance gains over the remaining 4 years. Unfortunately, that scenario just is not a possible reality. There are many flaws in the logic. First, the performance degradation is not likely to ever be fully eliminated on the current hardware. This diminshed expectation of gains also shrinks the time those are willing to put forth in adressing it. Secondly, using the

machine to develop and test for a full year instead of making progress on core mission projects is not an acceptable tradeoff. In reality, progress must continue even if at a lower degraded performance. Since the code teams continuously refine and improve the efficiency of the simulations that run on our current hardware, there is a reasonable expectation that the impact of the Meltdown and Spectre vulnerabilities will continuously be diminished over time in the normal course of business.

With the upper limit of impact being bound and expected to be tuned over time, business continues as usual with yet another performance factor to consider. The real question comes down to how the code teams should spend their time. Should they focus on making up for the lost performance deficit that will be obsolete with the new the installation of the next machine? Or should they spend their time on refining the simulation accuracy that will endure through various machines. The simulation codes will continue to evolve down their own paths and the code teams will certainly be cognizant of the performance aspects of Meltdown and Spectre, but whole-sale re-engineering is not feasible. Even if all the code teams decided that they needed to rewrite, revalidate and re-certify their codes, that progress would be throttled by lack of free cycles on an already heavily subscribed set of supercomputers.

With the simulation codes having their hurdles, we can look to other areas for potential performance recuperation. I/O is certainly one of those areas. There are a number of I/O libraries available to use and with various strategies in parallel applications. I/O was also one of the areas singled out early in the Meltdown and Spectre vulnerability as a speculated area of high impact. This has in fact proven true, to various degrees.

V. Observations

A. XRAGE with Asteroid Benchmark

An examination of the XRAGE with asteroid code with 64 PE using the bulkio libraries showed to have a very significant impact at a nearly 40% reduction in I/O performance. The numbers reflected in Figure[1] are measurements from weekly regression tests of write bandwidth during the I/O phase of the benchmark. While this number can certainly be alarming, when put into perspective of the code only spending around 5% of its time in I/O, the overall impact ends up being about a 2% overall penalty attributed to I/O. Obviously, different applications at different scales will have different level of impact. This scenario appeared fairly typical.

B. IOR

I/O has become a focus area for Meltdown and Spectre performance issues due to its impact. Fortunately, I/O is rather compartmentalized in comparison to the simulation code. The libraries used for I/O can be targeted for performance tuning. There are various strategies implemented and currently being pursued in this realm. Depending on the application, various levels of performance can be achieved.

We somewhat expected the focused I/O impact to cause users to change their checkpoint timings, but that has not been a widespread occurrence. We also have not seen requested wall-clock times grow to compensate other than under the premise of 'just to be safe'.

In focusing on the more pure file system impact, Alfred Torrez performed a series of tests using IOR and MDTest on our Lustre installations. With well-formed and aligned writes, only marginal performance degradation of 5 to 7% was observed on average. (see Figure[2]) In fact is was characterized as the expected slip over the period of examination. Other things that occur in the normal course of a machine's lifespan take a toll on performance. Most notably in this case was the file system capacity. As a file system fills its performance has a tendency to decline. E.g.: Figure[3] shows a variation in performance of around 5% on average between scratch1 and scratch2. On one in particular test set (not shown), the performance actually improved by around 17% on peak numbers. (Up to 900GiB/sec from 770GiB/sec). But this was an aberration caused by a fresh reboot of the storage system and the disk actuator arms being in optimal positions. It is offered here only to give additional perspective to the range of normal performance variation.

C. Code Development Efforts

Since compiling and building codes involves a lot of small file operations, these processes have reported seeing around a nominal 20% performance hit with peaks up around 32% on our commodity systems. Again, here we see greater sensitivity with non-optimal I/O patterns.

D. DataWarp

It was also reported that DataWarp staging of data from Lustre (stage-in) was almost wholly unaffected. Staging out showed degradation on par with general Lustre write usage.

E. Jitter-like Expansion at Scale

One of the concerns is the potential for the micro delays to have a cascading affect on job performance as is potentially seen in the phemonena known as *jitter*. Fortunately, we have not experienced, nor have we had any reports of this behavior. Larger scale runs tend to have larger performance penalties, but that has been qualified as being in line with I/O at scale norms.

VI. Recommendations to Users

Currently we are recommending to users experiencing slow down in their runs to evaluate their I/O performance and strategies as that seems to be the area with the most sensitivity to the Meltdown and Spectre mitigations currently. Before these vulnerabilities became prevalent, we

were already recommending to users experienceing slow performance to evaluate their I/O performance and strategies.

VII. Conclusion

While interesting, pursuing fine experimentation beyond the results that we see from in-situ workload was considered academic. The observed degradation was not great enough in the general case to warrant any significant deviation from previously planned code team efforts. The expectation is that these vulnerabilities will expire with future chip architectures. Users have, however, called for this type of thing to be heavily considered in the design, evaluation and acceptance processes of future machines.

Even though there are specific cases of very high performance impact, given appropriate mitigation, the variation was generally within the bounds of common machine performance degradation over time. This is also evidenced by the very few reports and complaints from the user community. A very notable observation in support of the Meltdown and Spectre vulnerabilities having a low impact is the lack of user behavior change. But this journey is certainly not complete. Perhaps there will be larger outcry if future mitigations and patches create a greater performance deficit.

Acknowledgment

We would like to convey special thanks to fellow colleagues Jason Hick, Nathan Hjelm, Andy Nelson, Alex Parga and Alfred Torrez of Los Alamos National Laboratory for their contributions of testing, data and insight.

Publication Reference

LA-UR-18-24290

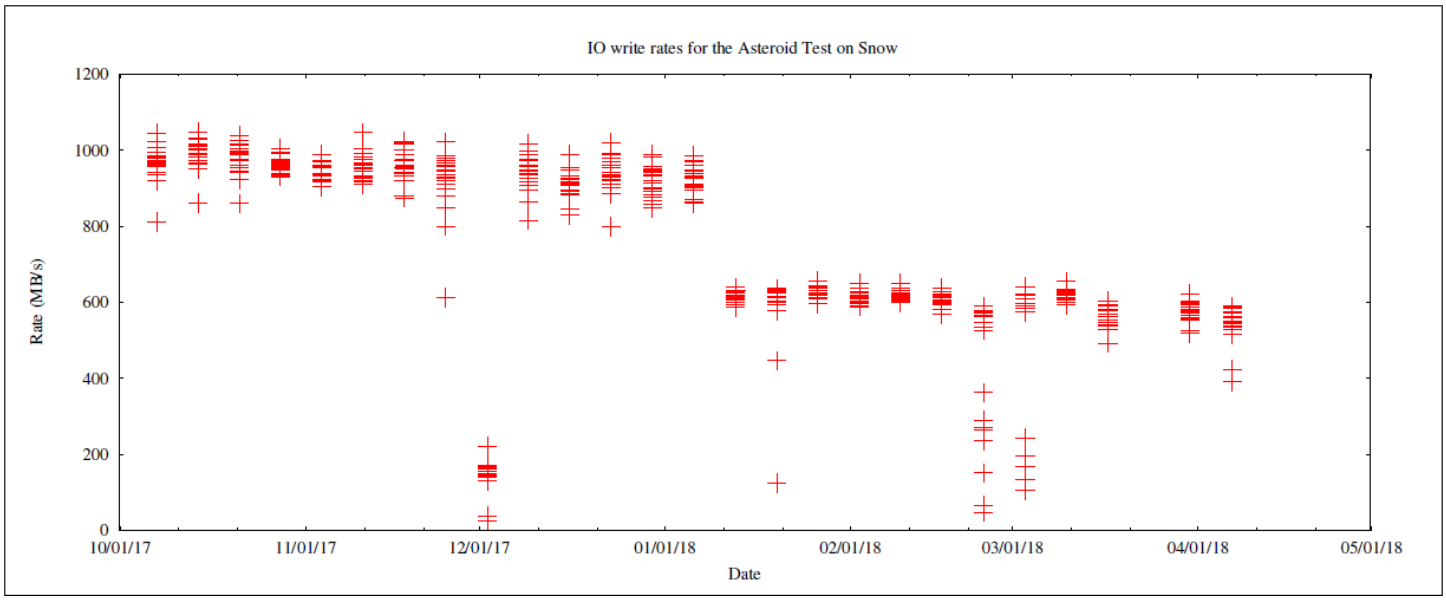
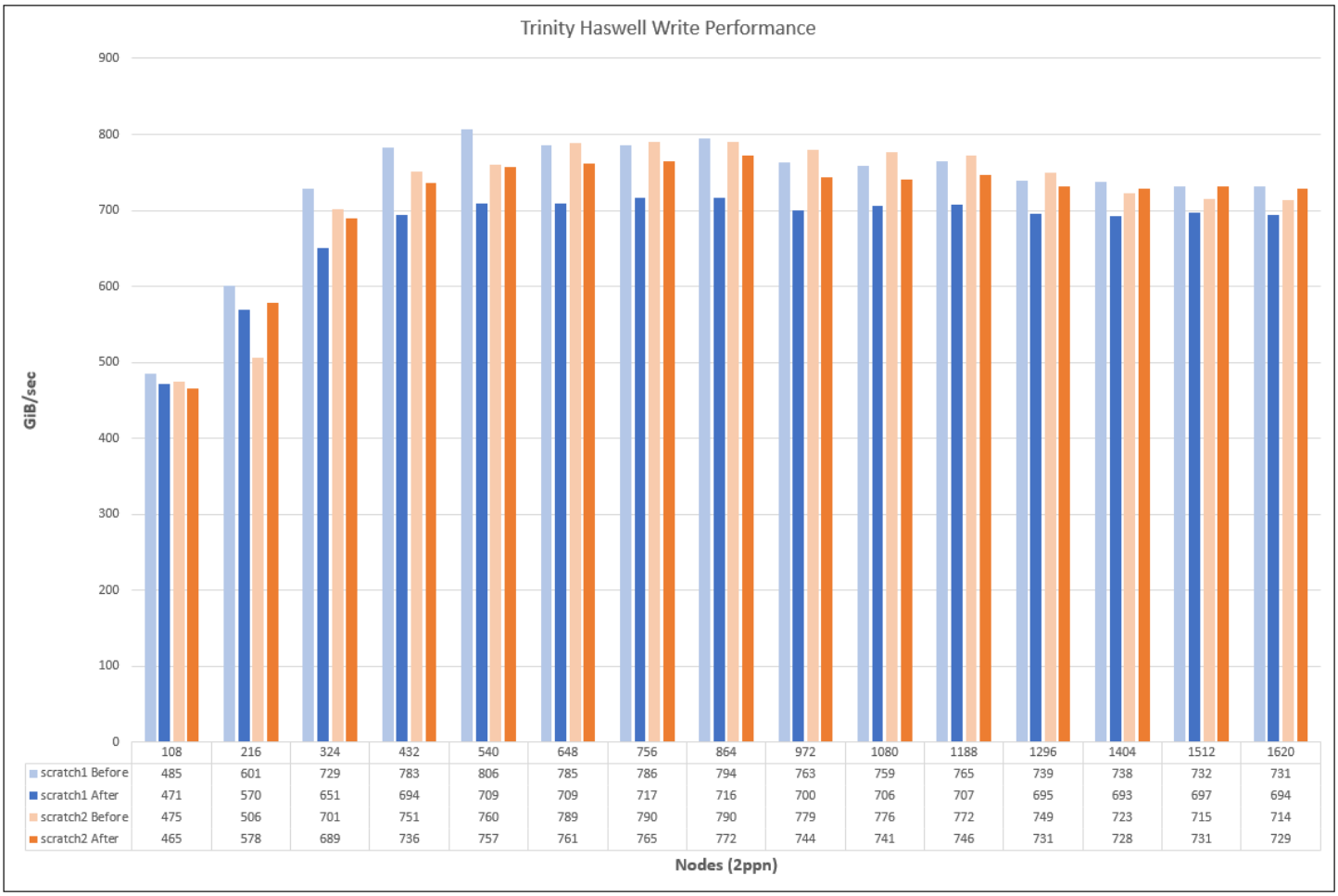
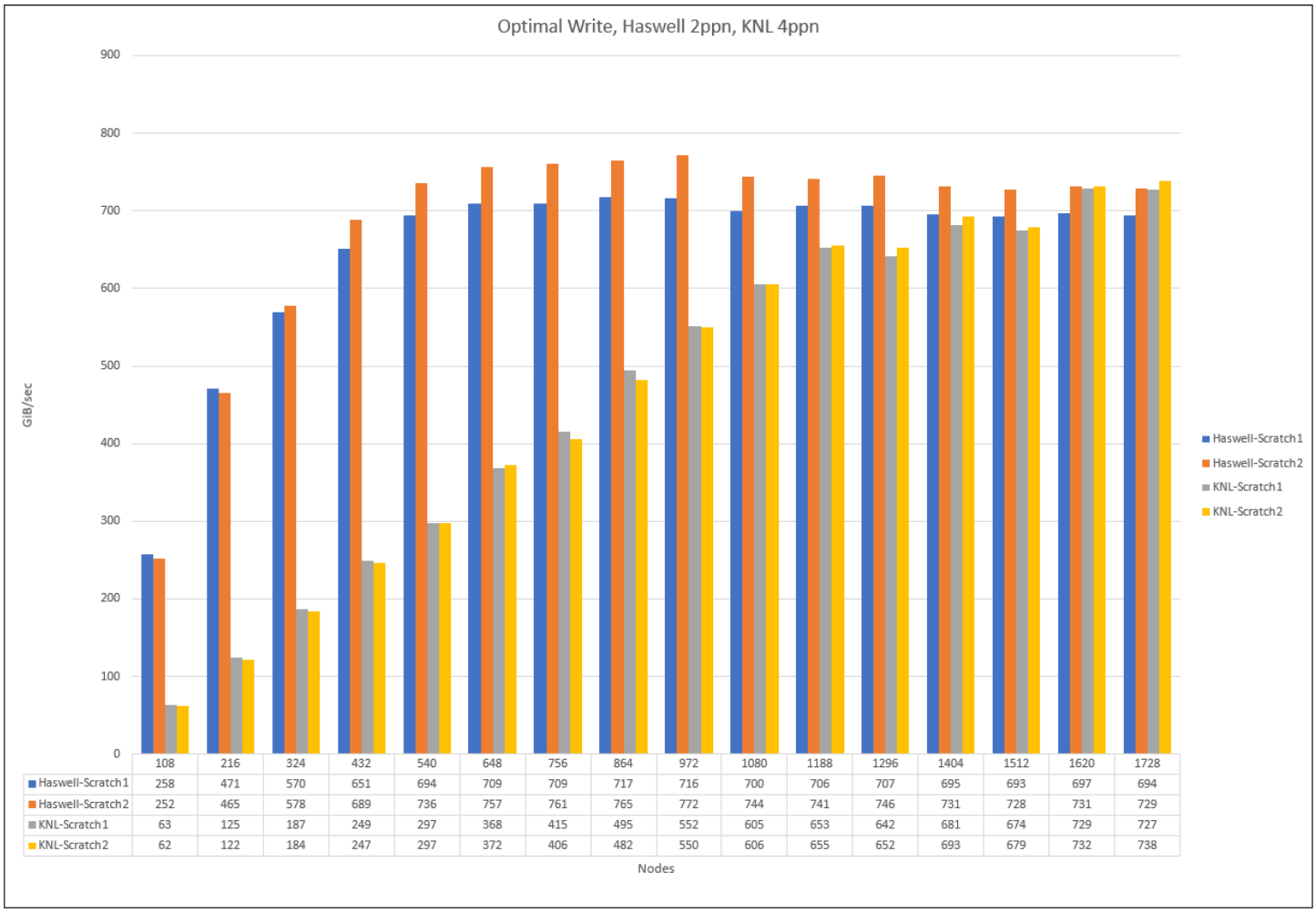


Figure 1. XRAGE with Asteroid 64 PE write performance over seven month period. Note application of patches in early January.



	108	216	324	432	540	648	756	864	972	1080	1188	1296	1404	1512	1620
scratch1 Before	485	601	729	783	806	785	786	794	763	759	765	739	738	732	731
scratch1 After	471	570	651	694	709	709	717	716	700	706	707	695	693	697	694
scratch2 Before	475	506	701	751	760	789	790	790	779	776	772	749	723	715	714
scratch2 After	465	578	689	736	757	761	765	772	744	741	746	731	728	731	729

Figure 2. Degradation of Trinity write performance on scratch1 and scratch2 from 9-22-17 (before) and 4-2-18 (after).



	108	216	324	432	540	648	756	864	972	1080	1188	1296	1404	1512	1620	1728
Haswell-Scratch1	258	471	570	651	694	709	709	717	716	700	706	707	695	693	697	694
Haswell-Scratch2	252	465	578	689	736	757	761	765	772	744	741	746	731	728	731	729
KNL-Scratch1	63	125	187	249	297	368	415	495	552	605	653	642	681	674	729	727
KNL-Scratch2	62	122	184	247	297	372	406	482	550	606	655	652	693	679	732	738

Figure 3. Performance variation of around 5% on average between scratch1 and scratch2. Provided as an example of performance difference due to usage.