Performance Variability on Xeon Phi

Brandon Cook(✉), Thorsten Kurth, Brian Austin, Samuel Williams, and Jack Deslippe

Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
{bgcook,tkurth}@lbl.gov

Abstract. An understanding of sources of performance variability is important for high performance application developers and users. In this paper we discuss non-I/O sources of application performance variability on Cori, a Cray XC40 at NERSC with 9600+ Xeon Phi nodes connecting to an Aries high speed network with a Dragonfly topology. Our survey covers variability due to on-node effects from MCDRAM configured as cache and clock frequency scaling as well as off-node effects due to the network. For each source of variability we quantify the variability through micro-benchmarks and mini-applications, discuss potential mitigation strategies and analyze the impact on applications.

Keywords: Cori · NERSC · Aries · Dragonfly · Performance variability

1 Introduction

High performance computing platforms harbor many potential sources of performance variability, including features of the on-node architecture, network, and I/O subsystem. Quantifying and understanding the various sources of variability is essential to application developers and performance engineers who want to analyze and measure application performance and for scientists to make efficient and informed use of resource allocations. In this paper we focus on non-I/O sources of variability on the Intel Xeon Phi many integrated core architecture.

In Sect. 2, we describe the target architecture. In Sect. 3, we describe applications that will be used to demonstrate the effects on variability on real-applications. Sections 4, 5 and 6 discuss different on-node and off-node sources of variability: MCDRAM configured as direct map cache, dynamic voltage-frequency scaling, and job placement. For each section, the mechanism of variability is introduced, mitigation and identification methodologies are discussed and illustrated with microbenchmarks. Finally, the impact on of variability on the performance of real applications is presented.

2 System Architecture

The Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC) is a Cray XC40 based supercomputer currently ranked 5th on

J.M. Kunkel et al. (Eds.): ISC High Performance Workshops 2017, LNCS 10524, pp. 419–429, 2017.
https://doi.org/10.1007/978-3-319-67630-2_31

the Top 500 list. Cori is unique in that it contains both Xeon and Xeon Phi nodes with a common scheduler, I/O subsystem and high speed network (HSN). Cori is configured with the following components; our analysis focuses on Cori's Xeon Phi partition.

- 2000+ compute nodes with 128 GB DDR4@2133 MHz per node and two 16-core 2.3 GHz Intel Haswell processors
- 9600+ compute nodes with 96 GB DDR4@2400 MHz per node, 16 GB on-package MCDRAM and one 68-core 1.4 GHz Intel Xeon Phi processor
- 1 PB aggregate memory
- 30 PB scratch filesystem with over 700 GB/s peak bandwidth
- Cray Aries network with Dragonfly topology and 45 TB/s bi-directional global bandwidth.

3 Applications

We use the following applications to illustrate several sources of variability and to discuss their impact on realistic workloads.

The HPGMG-FV benchmark [4,5] is highly instrumented, thus providing an immediate and detailed timing breakdown, and it is heavily optimized for threaded environments. Moreover, as it implements a variable-coefficient, fourth-order Laplacian on a structured grid, it is moderately compute intensive and thus sensitive to frequency variations.

The deep learning framework IntelCaffe [1] features highly optimized compute kernels for Intel® Xeon Phi as well as Intel Machine Learning Scalability Library (MLSL) [2] which provides communication primitives optimized for deep learning applications. The computational kernels are mostly SGEMM-like and are thus very compute intensive. The communication pattern is very well-defined as IntelCaffe uses only non-blocking allreduce operations. This communication pattern is very demanding because allreduces of large messages can put pressure on the network.

The lattice QCD application Chroma [3], together with the QPhiX Wilson operator and solver package [6,7], uses a hybrid OpenMP and MPI approach to parallel programming. It makes heavy use of JIT compilation, AVX512 intrinsics and employs pool allocators in order to improve memory allocation performance on Xeon Phi. Chroma's performance is limited by memory bandwidth and is sensitive to communication latency at large scale. The primary communication pattern nearest neighbor boundary exchange in the Wilson operator (4D stencil) and the BiCGStab solver additionally issues small allreduce operations.

MiniFE mimics the key operations of many finite-element applications that rely on implicit solvers. MiniFE uses a simple, non-preconditioned conjugate-gradient solver that consists mainly of sparse matrix-vector products and vector-vector operations that are memory bandwidth sensitive.

4 MCDRAM Cache

4.1 Introduction

Cache conflict misses are a well-known phenomenon arising from memory references aliasing to the same cache line in a limited associativity cache and resulting in superfluous data movement. Users often mitigate this by padding array sizes so that the (virtual) addresses of concurrently accessed array elements no longer alias. Such techniques work well on virtually-addressed caches or in cases where the padding is smaller than a page. Unfortunately, for cache architectures with very large numbers of sets (e.g. the MCDRAM cache on KNL), it is possible for multiple pages within or across processes to map to the same cache set despite having distinct (virtual) address bits (the TLB hides the true aliasing from users). This is an artifact of, upon a `malloc/sbrk`, the system (perhaps agnostic of the cache architecture) allocating two physical (and aliased pages) from the free page list to disjoint virtual addresses. On a system like KNL where the cache is direct mapped (no associativity), such accesses can result in MCDRAM cache thrashing and degraded performance.

4.2 Mitigations/Solutions

For a freshly booted node, the page list is initially "well ordered" in the sense that the first pages allocated from the heap do not conflict with each other. Over time, as various processes allocate and free memory, the entropy of the free page list increases, the probability of being allocated conflicting pages increases, and performance decreases.

One might consider rebooting nodes before each job to eliminate this source of performance variation, but lengthy reboot times make this suggestion untenable. To help avoid cache conflict misses without rebooting, Intel developed the so-called "Zone-Sort" kernel module, which performs on-demand sorting of Linux's free page list. The overhead for running Zone-Sort is small enough (<1 s) that it can be run before each job. On Cori, we have modified the Slurm prologue to call Zone-Sort immediately before each application launch. (This can be optionally disabled by adding `--zonesort=off` option to either of the sbatch or srun commands.)

4.3 Microbenchmarks

KNL's vulnerability to cache thrashing is illustrated in Fig. 1(left), which shows the average performance of 128 single-node runs of the MiniFE application in KNL's *quad,cache* mode. The first iteration scans a series of problem sizes, ranging from 4 to 50 GB, and achieves good performance up to the MCDRAM cache size of 16 GB, but decreases rapidly for larger problems as the number of capacity cache misses increases. The second iteration scans the same series, but has significantly worse performance for problems that fit into MCDRAM cache. Figure 1(right) repeats the scan of MiniFE problems, but calls Zone-Sort

before each run. The two scans now generate the same performance for all problem sizes, which confirms that the poor performance observed in Fig. 1(left)-2nd run was due to cache thrashing, and demonstrates the effectiveness of Zone-Sort for MiniFE. However, the extent of the performance degradation depends on the state of the free page list, which varies across nodes, but also on the susceptibility of an application's memory access pattern to cache conflicts; other applications might not show the same sensitivity or improvement.

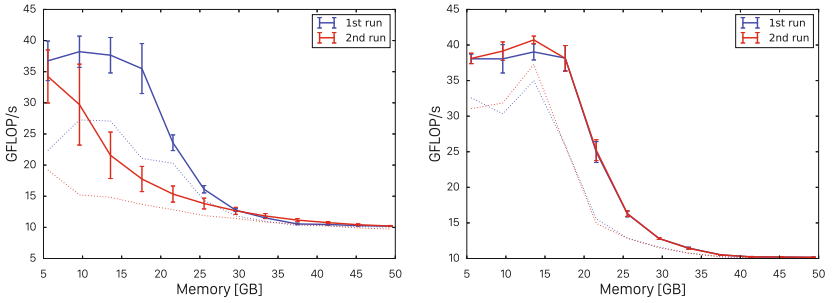

Fig. 1. MiniFE performance without (left) and with (right) the Zone-Sort kernel module. Each curve shows the average, standard deviation and minimum performance for 128 single-node runs.

4.4 Impact on Applications

Figure 2 presents the run time for HPGMG's smooth operator on the finest multigrid level for a problem size of 256^3 per process under either cache mode (*quad,cache*) or flat mode (*quad,flat* with `numactl -m 1`). Without Zone-Sort or rebooting for several weeks, the time per process for this ideally load balanced computation is far from uniform in cache mode (the Linux kernel on each node behaves independently based on its unique history). Conversely, in flat mode, there is only a 5% variation in performance. For bulk synchronous or frequently synchronizing applications like HPGMG, this node-to-node variability induced load imbalance manifests as increased `MPI_Wait` times, reduced performance (limited by slowest node), and reduced scalability (as one scales to higher concurrencies, the probability of being allocated a 'slow' node increases). With `#SBATCH --zonesort=on` and a more frequent reboot schedule, the discrepancy between flat and cache mode is diminished. As a result, overall performance and scalability are improved.

5 Dynamic Voltage and Frequency Scaling (DVFS)

5.1 Introduction

In response to thermal limits, power limitations and instruction modern architectures can dynamically adjust processor clock frequency. On Cori the default

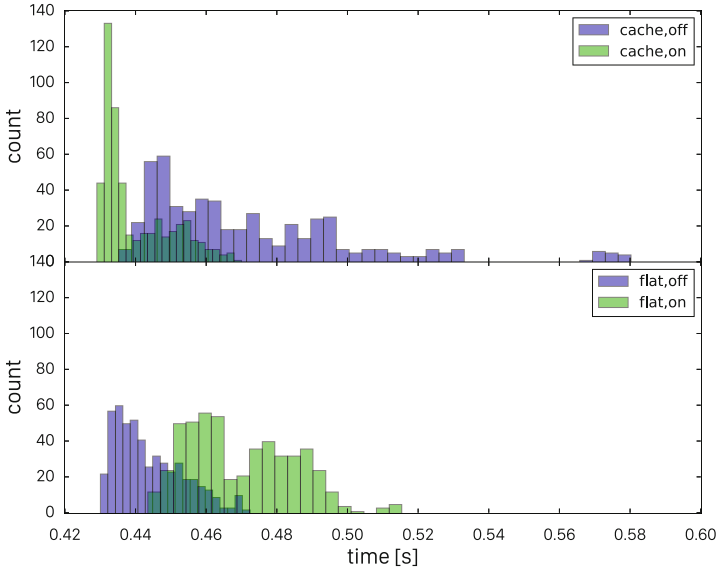

Fig. 2. MCDRAM Cache aliasing induced load imbalance on HPGMG. There are 512 processes spread over 64 nodes. Each count represents the nominally load-balanced smooth time for that process averaged over 60 s.

cpufreq driver is "acpi-cpufreq" and the minimum and maximum scaling frequencies are 1000000 kHz and 1401000 kHz. The default governor is "performance" which will attempt to run the CPU at the maximum possible frequency. However, this is an upper bound and the processor can always run at a lower frequency than that requested by the operating system. For example, when executing AVX-intensive blocks of code, the cores on a node will downclock to 1.2 GHz.

Cori users can select non-default frequency scaling through SLURM's `--cpu-freq=P1` option, which will set the target(upper) frequency (in KHz) a job can nominally run at. These can be used either on a job-wide (#SBATCH) or run step (srun) basis.

5.2 Microbenchmarks

One of the most widely used BLAS operations is `DGEMM`. Figure 3 shows histograms of DGEMM performance collected with $n = m = k = 2048$, $\alpha = 1$ and $\beta = 1$ with Intel MKL distributed with Intel compilers 2017.1.132. The processor was in *quad,flat* mode and the benchmark was run using pure OpenMP with `numactl -m 1` to bind memory to the MCDRAM. For each frequency, 500 calls to `DGEMM` were made and the first call was excluded.

Increasing the number of threads greatly increases the variability, as shown by the wide spread with 64 threads in Fig. 3. However, reducing the clock frequency increases the sharpness of the variability at the cost of reducing the peak

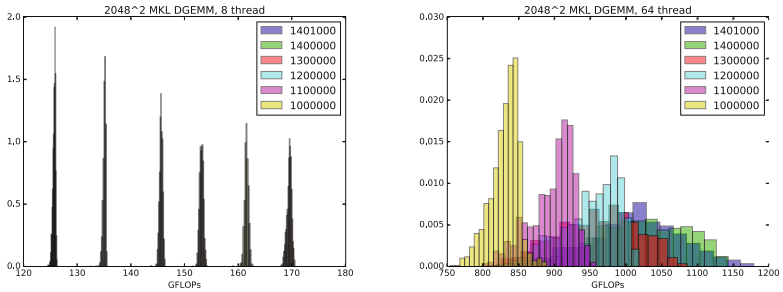

Fig. 3. Normalized histograms of 1000 DGEMM's with $n = m = k = 2048$ with 8 and 64 threads for different frequencies (kHz). All measurements were done on the same node using a single process at a time.

performance. In some cases this may be an energy efficiency optimization or just a tool for reducing the impact of variability when analyzing other potential code modifications.

5.3 Impact on Applications

Figure 4 presents on-node variability for HPGMG as a function of frequency (e.g. `--cpu-freq=1200000`). In affect, reducing the frequency makes the application more compute-limited and sensitive to slightly different code execution paths. In all cases, we used 64 nodes running in *quad,flat* with either 8 threads per process (512 processes) or 64 threads per process (64 processes). Figure 4(left) shows substantial variation in smoother (on-node computation) performance at low frequencies with 8 processes per node. This is quite surprising as all processes perform exactly the same computation (perfect load balance). As one increases frequency, one approaches the lower bound imposed by the memory limit and thus reduces variability. Close examination shows the 8 processes on each node deliver similar performance while processes on different nodes can deliver substantially different performance. Conversely, Fig. 4(right), which is configured to use one 64-thread process per node, shows neither variability between processes nor variability between nodes. At high frequencies, 64 threads deliver nearly the same on-node performance as 8 processes of 8 threads (memory bandwidth wall). Unfortunately, at low frequency run time is now consistently at the upper bound of variability.

Although this routine is perfectly load balanced on-node computation, the prior routines (filling ghost zones with MPI data or with a boundary condition) will vary from process to process. This suggests that these routines can artificially warm up the caches or pollute the TLBs such that the effects are seen in the subsequent (plotted) smooth routine. At high thread concurrencies, threads walk all over memory to affect the boundary condition or to fill in ghost zones, and are thus unlikely to have warmed up the caches for the smoother stencil.

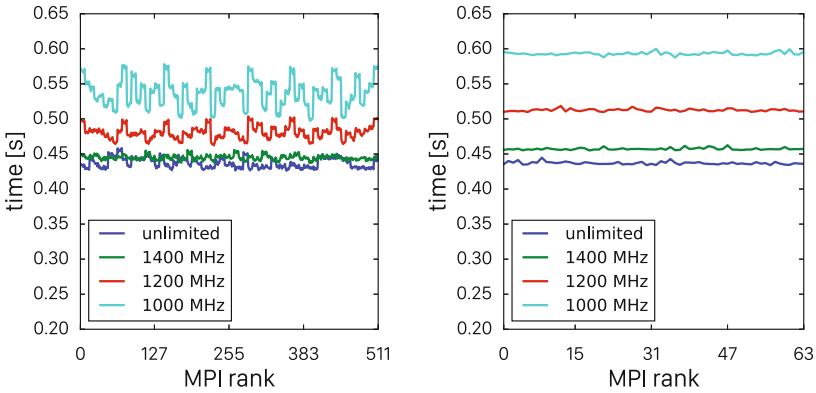

Fig. 4. Effects of frequency scaling on HPGMG's on-node smoother (averaged over 60s) at 64 nodes in *quad,flat* mode. Observe, there is far more variability in the 512 process $\times 8$ thread configuration (left) while the 64×64 configuration (right) generally runs slower at low frequencies.

6 Job Placement

6.1 Introduction

A "Dragonfly" topology consists of groups of locally connected routers with groups connected to each other by high-speed global links. Within the Cray's Aries implementation of the dragonfly topology, local groups consists of a two-cabinet pair and contains a total of six chassis. A chassis houses sixteen blades, and each blade has four slots (or nodes) and a (shared) Aries router. This leads to a total of 384 slots in each Aries two-cabinet group. However, burst buffer nodes and other system service nodes may displace compute nodes, reducing the number of compute nodes per group to as few as ∼355. All blades in a chassis are wired in an all-to-all fashion called rank-1 network or *green* links. The rank-2 network (*black* links) connect each blade in a chassis with each of its peers in other chassis of the same group. The latency and all-to-all bandwidths for rank-1 and rank-2 connections are similar. Finally, the rank-3 network (optical *blue* links) connects each Aries group with every other group. Adaptive routing is enabled on Cori, which allows packages to take routes which are not considered as shortest routes in terms of number of hops. This feature mitigates network congestion problems by rerouting packages through different chassis and cabinets. However, this can also lead to interference with applications running in other cabinets. Applications that span multiple nodes are expected to be more vulnerable than applications which only run on few nodes. Applications that span multiple chassis and cabinets are likely be more affected than jobs that are placed in a compact manner. In this section, we will investigate the performance variability dependent on job placement.

6.2 Mitigations/Solutions

SLURM provides control over job placement topology through the switches option to sbatch. The syntax is `--switches=N@HH:MM:SS`, where N is number of Dragonfly groups and HH:MM:SS is the maximum time to delay the start of the job until an allocation satisfying the requested number of switches is found. If the time elapses then the job will take the next available allocation which may not satisfy the requested number of switches. From the application side, hugepages can be used to mitigate Aries TLB thrashes. For that purpose, the code has to be recompiled with cray-hugepages module enabled and at runtime a page size has to be set.

6.3 Impact on Applications

We first tested if adaptive routing has the potential to interfere with applications running at very small scale and compact job placement. For that purpose, we ran IntelCaffe on 4 and 8 nodes on Cori ensuring that the 4 node jobs are contained within a single blade and the 8 node jobs span two adjacent blades. In both cases, we employed 66 OpenMP threads where each thread is bound to one physical core and dedicated additional 2 cores to the operating system by using the core specialization feature of SLURM (i.e. by specifying `-S 2` in the submit script). We are used 2MiB hugepages and RDMA accelerated collectives through DMAPP by enabling `MPICH_NETWORK_BUFFER_COLL_OPT`, `MPICH_RMA_OVER_DMAPP` and `MPICH_USE_DMAPP_COLL`. Our experiments were carried out on *quad,cache* configured nodes. Since all node-local data fits fully into MCDRAM, we did not see any difference from *quad,flat* mode. For comparison, we also ran the eight node job on our TDS system Gerty, using the same binary, same setup and system configuration. Gerty is a small-scale, lightly used test and development system with a comparably quiet network and thus much less prone to network noise. The neural network used was a 5 layer convolutional neural network (CNN) with small fully connected layer and $224 \times 224 \times 3$-sized input images that could be fully cached in memory. We measured the execution speed of forward, backward and update steps averaged over 10 iterations. The backward pass issues non-blocking allreduce operations that are concluded by a wait command in the subsequent update step of the corresponding layer. Therefore, only the latter should be affected by communication variability. The results from this test are displayed in Fig. 5, which shows that the computationally heavy forward and backward passes do not show any variability for any system and node count. The update step however shows severe performance variations. We further notice that variability on Cori depends on job compactness, i.e. jobs contained within a blade show much less variability than jobs spanning multiple blades. The same code executed on eight Gerty nodes shows very little performance variation with a somewhat periodic pattern[1]. The performance variation on Cori is less regular

[1] We observed spikes in the execution time about once every 20 iterations; averaging over 10 iterations gave rise to the figure's zig-zag pattern.

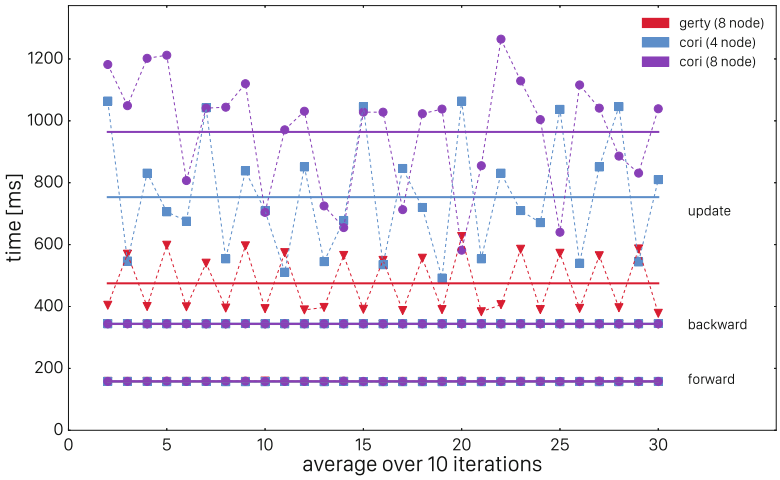

Fig. 5. Timings for forward, backward and update steps of a 5 layer CNN in IntelCaffe on different systems and for different node counts. The timings are averaged over 10 subsequent iterations. Solid lines represent mean timings for the respective steps.

and suggests that those runs were potentially impacted by other applications running on the same machine.

To further investigate the job placement dependence of performance variability, we performed medium scale runs with Chroma. Since this code is mostly doing nearest neighbor message exchanges, it is not as sensitive to network congestion from concurrently running applications as IntelCaffe. For this experiment, we employ 4 MPI ranks per node and 256 nodes in total. We further utilize 32 threads per rank and allocate 3.5 GiB of memory pool data and use 2 MiB hugepages. We run the Chroma Hybrid Molecular Dynamics (HMC) trajectory benchmark over several trajectories on a global $64^3 \cdot 128$ lattice (local size $16^2 \cdot 8 \cdot 16$) and measure the performance of each call to the BiCGStab solver. Comparing performance variation with in a job to its distribution of nodes across the network can provide a qualitative understanding of the impact of node placement on Chroma's performance. Figure 6(a) shows the performance histories for three selected jobs as well as their placements Fig. 6(b), (c) and (d). The later three depict the compactness of the job: they each show 4 cabinets of the Cori system along with worker nodes (open rectangles), service nodes (solid black rectangles) and the nodes occupied by the respective job (solid red rectangles). Rows and columns of the images can be directly mapped to physical rows and columns of the system. We observed that job 3724518 showed low performance variability and was also placed inside one cabinet in a contiguous fashion. Job 3731220 is contiguous to a certain extent, but spans two cabinets and has "holes" in between; the performance of this job is significantly less reliable. Finally, job 3828179 was scattered across four cabinets and shows huge variation over the duration of the run. Job 3731220 features some severe downward spikes at the

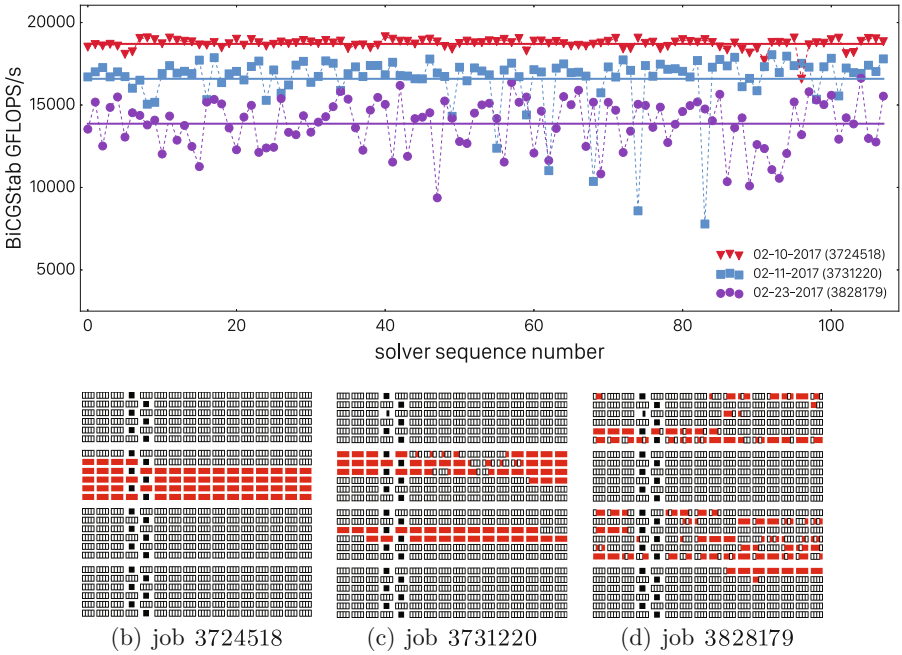

Fig. 6. Performance histories of Chroma HMC runs (a) with three different job placements: compact and contiguous (b), contiguous with holes (c) and scattered (d). The solid lines in (a) represent the corresponding mean flop rates. Solid red rectangles in (b, c, d) represent nodes occupied by the given job, black rectangles are service nodes and open rectangles are other worker nodes which might run other jobs at the same time (not shown). (Color figure online)

end of its lifetime that have not yet been fully explained. Given the approximate periodicity of the spikes, they may be related to heavy I/O operations by another job on the system. The root cause is still under active investigation.

7 Conclusions

In this paper we have highlighted three non-IO sources of performance variability on the Cori machine at NERSC. Effects of cache conflicts when MCDRAM is configured as a cache are one of the most prominent potential sources of performance variability on Xeon Phi, and the use of the Zone-Sort kernel module was found to be very beneficial in recovering lost performance. Variability stemming from DVFS does not show the same clear, well-defined signal and is highly application dependent, particularly when MKL is used. However, by running applications and benchmarks at different nominal clock frequencies this source of variability can be probed potentially at the cost of absolute performance, but of benefit to developers benchmarking code changes. Job placement in a large-scale production environment is a difficult problem, particularly when adaptive

routing is in use and applications which are communication intensive should take job placement and interference effects into consideration when analyzing performance. The identification of sources of performance variability will continue to be important as processor architecture becomes more complex and system size increases.

Acknowledgments. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

1. Intel® distribution of Caffe* (2017). https://github.com/intel/caffe
2. Intel® Machine Learning Scaling Library for Linux* OS (2017). https://github.com/01org/MLSL
3. Edwards, R.G., Joo, B.: The Chroma software system for lattice QCD. Nucl. Phys. Proc. Suppl. **140**, 832 (2005)
4. http://crd.lbl.gov/departments/computer-science/PAR/research/hpgmg
5. https://bitbucket.org/hpgmg/hpgmg
6. Joó, B.: `qphix` package web page. http://jeffersonlab.github.io/qphix
7. Joó, B.: `qphix-codegen` package web page. http://jeffersonlab.github.io/qphix-codegen