SAND2016-0028C

NiMC: Characterizing and Eliminating Network-Induced Memory Contention

Taylor Groves*†, Ryan E. Grant
Sandia National Laboratories*
Center for Computing Research
P.O. Box 5800, MS-1110
Albuquerque, NM 87185-1110
email: {tgroves,regrant}@sandia.gov

Dorian Arnold
University of New Mexico†
Department of Computer Science
1 University of New Mexico
Albuquerque NM 87131
email:{tgroves,darnold}@cs.unm.edu

***Abstract*—Remote Direct Memory Access (RDMA) is expected to be an integral communication mechanism for future exascale systems – enabling asynchronous data transfers, so that applications may fully utilize all CPU resources while simultaneously sharing data amongst remote nodes. In this paper we examine network-induced memory contention (NiMC), the interactions between RDMA and the memory subsystem when applications and out-of-band services compete for memory resources and NiMC's resulting impact on application-level performance. For a range of hardware technologies and HPC workloads, we quantified NiMC and show that NiMC's impact grows with scale resulting in up to 3X performance degradation at scales as small as 8K processes even in applications that previously have been shown to be performance resilient in the presence of noise. We also evaluated three potential techniques to reduce NiMC's performance impact, namely hardware offloading, core reservation and software-based network throttling. While all three of these solutions show promise, we provide guidelines that help select the best solution for a given environment.**

***Keywords*-Measurement; Performance; Memory Contention; Network Contention; Networks;**

I. Introduction

On today's high-performance computing (HPC) systems, standard communication mechanisms are synchronous, requiring active sender and receiver participation in message transmission – expending resources and time on both ends. On emerging exascale-class systems, projected to comprise several orders of magnitude more processing elements than today's fastest systems, synchronous operation is expected to become a prohibitive bottleneck [9]. As systems and application researchers investigate novel approaches to reduce application synchronization, asynchronous communication widely is considered to be a part of the exascale system design solution. We expect that this emerging trend toward less synchronous computational paradigms and services, along with improvements to one-sided communications in MPI-3 [19] and new network technologies will lead to increased popularity for asynchronous communication.

*Sandia National Laboratories s a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Remote direct memory access (RDMA), also called one-sided communication, is a popular and useful mechanism for implementing efficient asynchronous communication. RDMA allows a node's local memory to be read or written by a remote node without involvement of the local operating system or CPU. Such *out-of-band* communication incurs no direct overhead on the target machine. There are many attractive use cases for RDMA, such as to overlap an application's communication and computation phases in non-bulk synchronous parallel (non-BSP) computational paradigms, for in-memory asynchronous checkpointing and for in-situ analytics. However, little is known about the potential indirect application interference of the additional memory contention caused by RDMA communication. Consider, for example, uncoordinated checkpoints that are staged in the memory of remote nodes before being moved to stable storage, as in SCR [17]. Memory traffic from a remote node writing a checkpoint will contend with the memory transactions of a local memory-bound application. Generally, memory contention between local operations and out-of-band network operations can cause significant decreases in memory and thus application performance. We refer to this phenomenon as network-induced memory contention (NiMC[1]).

Further exacerbating the situation, many-core technologies will be a fundamental part of the exascale system design solution. However, a greater number of hardware threads means a greater demand for shared resources such as the network and memory sub-system. As shown in Figure 1, while the total off-chip bandwidth has been increasing, per-core memory bandwidth has not been increasing as significantly. Coupling these trends with the increased interest in one-sided communication, *it becomes critical that we understand the potential application performance impact of NiMC*. While researchers have explored the memory subsystem as an area of concern for extreme scale systems [27], [3], the concept of NiMC has not been characterized previously.

In this work, we studied the application performance impact of NiMC on a variety of hardware architectures and evaluated three candidate solutions. The results show that **NiMC can**

[1]*nim* is an English form of the German word nehmen meaning "to take or steal". Using 'C' as "cycles", NiMC ($\backslash' nim -' s\bar{e}\backslash$) can also mean "*to steal cycles*" – the net result of network-induced memory contention.

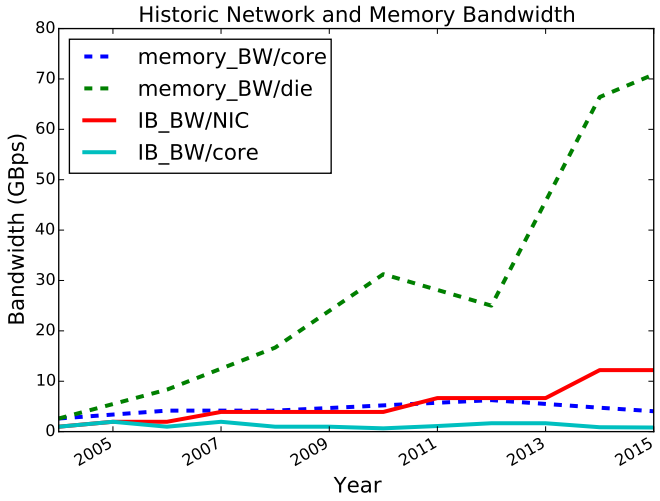

Fig. 1: A 10 year history of network and memory bandwidths. While total bandwidth has been increasing, per-core network and memory bandwidth have not significantly increased. The result is increased contention for the shared network and memory resources.

degrade single node memory bandwidth by 59.6%. Furthermore, for an **application that is performance-resilient to system noise, we show NiMC can increase runtime by up to a factor of three** (at our tested scales).

The specific contributions of this paper are:

- a comprehensive analysis of NiMC and its isolated effects;
- a quantification of the performance impact of NiMC for a range of systems and applications;
- a characterization of the system and application attributes that exacerbate NiMC; and
- evaluations of three candidate NiMC solutions;

NiMC is characterized on many different systems and found to exist on most of them in §IV. Further investigations in §V, show the impact from NiMC on applications and mini-apps for one of the evaluated systems, showing an approximately 2X runtime increase for 5 of 7 of the studied workloads. We then prove that NiMC is an issue for some systems at scale with a real application in §VI. Work in §VII then demonstrates the effectiveness of our three proposed solutions at scale with tests conducted on two large systems comprising a total of 160,000 core hours worth of runtime. Throughout these investigations, evidence is provided showing that the slowdowns attributed to NiMC are not due to contention for compute or network resources. Finally, our work concludes with related work and final conclusions in §VIII and §IX, respectively.

II. Network-Induced Memory Contention

Remote memory operations can induce memory contention in two ways: (1) RNICs (RDMA-enable network interface controllers) producing memory traffic in offloaded networks and (2) CPU-to-memory traffic when the CPU is used for onload network processing. For onloaded RDMA, not all traffic necessarily flows through CPU cores before being placed in memory. However, programming the DMA engines on the RNIC requires CPU intervention and causes some data to be transferred from the RNIC to the core to facilitate DMA requests. While there is ongoing debate for onloaded versus offloaded networking, this study reveals that NiMC should be an important consideration in this debate.

As Figure 1 illustrates, total network bandwidth is now much greater than per-core memory bandwidth. For future exascale systems, this relationship is expected to become more pronounced [14], compounding the memory contention caused by RDMA operations. Trends away from traditional BSP programming models toward finer-grained, asynchronous models that admit higher levels of concurrency also will lead to to greater demands on the memory subsystem and the network. Lastly, other exascale requirements, for example application resilience, in-situ data analysis and uncertainty quantification, will generate additional local and remote memory traffic associated with activities only indirectly related to the application. NiMC can be particularly troublesome when caused by traffic not directly related to the computation.

RDMA traffic that causes NiMC has the potential to also impact application performance through congestion on the network fabric [26] (rather than memory). Therefore, this study uses methods of introducing NiMC that are realistic usage scenarios, but also ones that do not create additional contention on the network. Specifically, (1) the examination of NiMC impact on single node jobs (§V) does not utilize the network for application communication. (2) in §VII-B we show that the observed slowdowns at scale are caused by NiMC on the node, rather than in the fabric.

III. Evaluation Methodology

Our approach to study NiMC is straightforward: for each application and hardware configuration under test, we injected remote memory operations into the compute node(s) and measured the resulting application perturbation by comparing application performance with and without RDMA traffic.

First, we used a memory-bandwidth benchmark to establish a baseline for application perturbation. These experiments were executed on multiple architectures to observe the NiMC impact for different hardware configurations. Our subsequent experiments then helped us to assess NiMC impact for real applications in single node and distributed contexts.

A. Injecting RDMA Operations

We used *ib_write_bw* from the Open Fabric Enterprise Distribution (OFED) Performance Tests [20] to generate network traffic streams. The OFED Performance Tests are a set of tests that use InfiniBand's user-level verbs API to measure IB performance. The *ib_write_bw* test uses RDMA writes to perform a series of operations between two connected nodes. As previously described, compared to traditional send/recv tests, the benefits of one-sided tests are that message delivery and synchronization are decoupled: after memory registration, writes do not significantly involve the CPU on the target node. We chose write operations as the overhead of performing read

operations on the nodes running the applications potentially would perturb the tests through use of compute cores to create and issue the read requests. Write requests do not have this issue, as the source node bears the burden of creating and issuing the network requests. An illustration of the flow of traffic to an individual node is presented in Figure 2.

B. The Workloads

1) STREAM: The STREAM memory benchmark [15] performs a small set of memory benchmarking kernels (copy, sum, scale, triad) that perform a small number of reads, arithmetic operations and a write back to memory. We used these operations to measure the sustainable memory bandwidth and corresponding computation rates by working with data sets significantly larger than the available cache. We used the STREAM triad test, $a(i) = b(i) + q * c(i)$, which is the most representative of a typical workload.

2) CNS: CNS [4] is a "simple stencil-based proxy-app for computing the hyperbolic component of a time-explicit advance for the Compressible Navier-Stokes equations using 8th order finite differences and a 3rd order, low-storage TVD RK algorithm in time." [8] CNS is intended to mimic the stencil operations of more realistic combustion applications, it does *not* mimic a typical problem found in combustion applications.

3) HPCCG: HPCCG [11] is an unstructured implicit finite element application, which calculates the conjugate gradient for a 3D chimney domain, running on an arbitrary number of processes. HPCCG creates a 27-point finite difference matrix, for which each MPI rank is designated a user-defined sub-block. This mini-app is generally considered to be memory-bandwidth bound, using from 25% to 75% of the total system memory. HPCCG is a designed to provide excellent weak scaling.

4) LAMMPS: LAMMPS [23], the Large-scale Atomic/Molecular Massively Parallel Simulator, is a molecular dynamics code modeling particles in different states. LAMMPS provides excellent weak scaling with the majority of communication occurring among nearest neighbors. In this work, we used a benchmark problem/data set to model the melt of a 3D Lennard-Jones system, using a weak scaled problem of a similar size to studies published by the authors of LAMMPS (32,000 atoms per core) [29].

5) LULESH: LULESH [13] represents shock hydrodynamics code solving a simple Sedov blast problem, illustrating the behavior of such solvers in ALE3D. This proxy-app distributes the spatial domain onto a set of volumetric elements defined by a mesh, where each intersection of mesh lines represents a node. Within the LULESH proxy-app, there are a variety of kernels, of which some subset are memory bound. One constraint of LULESH is that it must run with a cubic number of MPI Ranks. Therefore, for our experiments we extracted additional parallelism by leveraging OpenMP (OMP) on any unused cores.

6) SNAP: SNAP [32] models the performance of a modern discrete ordinates neutral particle transport application. This proxy-app does not employ any real physics in its calculations. Instead, SNAP produces the computational workload, memory requirements, and communication patterns that match the Los Alamos National Laboratories application, PARTISN. To distribute larger problem sizes, SNAP spatially decomposes its 3D mesh and maps it onto a 2D domain of MPI ranks. MPI ranks send and receive data following wave propagation which limits the weak scaling of the proxy-app.

7) XSbench: XSBench [28] is a proxy-app which represents the most significant kernel (85% of runtime) in a robust nuclear reactor core Monte Carlo particle (neutron) transport simulation. This variety of simulation can have significant data usage requirements and the proxy-app is considered to be memory-intensive. XSBench focuses on modeling intra-node performance characteristics of OpenMC and is not intended to be run at scale, as communication is limited to a single reduction at the end of a run.

C. The Platforms

Our study used nine different platforms from the Sandia National Laboratories and the Texas Advanced Computing Center consisting of a variety of architectures. We provide a concise description in Table I for the reader's convenience.

For a subset of the machines (Westmere, Lisbon, and Piledriver-1600/1866), we performed our experiments with varied memory frequencies, allowing us to see the impact available memory bandwidth has on NiMC. Westmere and Lisbon required BIOS option changes, whereas the Piledriver systems are separate nodes with different memory modules.

All of the systems, other than Haswell-X2 and Sandy-Bridge-X2-FDR-offload, utilize an InfiniBand QDR network with a maximum speed of 32 Gbit/s. Within the Table I in the network column *on* and *off* signify whether the NIC is an onload or offload NIC. The observed bandwidth of these systems varies with the physical network topology and the degree of contention. The general observation is that larger production clusters (e.g. Sandy Bridge-X2-FDR-offload) tend to have a larger variability in observed bandwidth, since network resources are shared among multiple jobs which may compete for bandwidth. However this topic is beyond the scope of this paper and we refer the reader to [2] for further discussion of performance degradation due to nearby jobs.

IV. A Memory-bound Benchmark

In our first NiMC experiments, we used the memory-bound, synthetic benchmark, STREAM (§III-B). These experiments were used to assess NiMC impact for worse-case memory intensive applications and to evaluate what architectural features can lead to increased NiMC impacts.

STREAM used one OMP thread per core in order to saturate the available memory bandwidth. A first-touch memory allocation policy was used to optimize memory utilization within the NUMA hierarchy. Each experiment comprised 10 STREAM executions with 300 iterations of the triad kernel per execution. The average sustained bandwidth was then calculated based on the the average time to complete a triad operation.

To measure the impact NiMC has on memory bandwidth, we re-executed the same set of experiments; this time, our *origin*

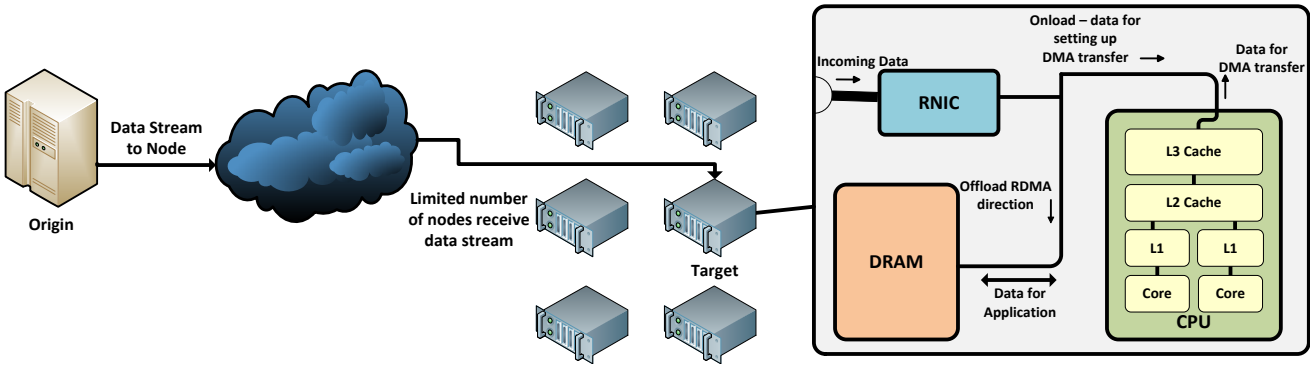

Fig. 2: An illustration of the data transfer path for NiMC, note that the path for a fully offloaded networking approach does not involve the CPU, while the on-loaded networking approach requires some CPU intervention to setup the data transfer.

TABLE I: Evaluated Architectures

machine	nodes	kernel	CPU	cores	channels	DRAM	DRAM GB/s	Network
Westmere@(800 MHz, 1066 MHz)	1	3.2.0 (Ubuntu12)	Intel E5620	4	2	16GB	12.8, 17.1	QDR IB off
Lisbon@(800 MHz, 1066 MHz, 1333 MHz)	1	3.13.6 (UN12)	AMD 4170 HE	6	2	16GB	12.8, 17.1, 21.3	QDR IB off
Piledriver-1600	70	2.6.32 (RHEL6)	AMD A10-5800K	4	2	16GB	25.6	QDR IB on
Piledriver-1866	2	2.6.32 (RHEL6)	AMD A10-5800K	4	2	64GB	29.9	QDR IB on
Sandy Bridge-X2-FDR-offload	6400	2.6.32 (Cent6.3)	2× Intel E5-2680	8	4	64GB	85.3	FDR IB off
Sandy Bridge-X2-onload	1196	2.6.32 (RHEL6.2)	2× Intel E5-2670	8	4	64GB	102.4	QDR IB on
Xeon-Phi (on-chip bandwidth)	49	2.6.38.8+mpss3.1.2	Xeon Phi 3120P	57	12	6GB	240	QDR IB off
Haswell-X2	33	3.14.23 (RHEL6.5)	Intel E5-2698	16	4	128GB	136	FDR IB off

node continuously injects 64 KiB of data (using *ib_write_bw* as described in §III) into a buffer allocated on the different *target* node on which the STREAM benchmark was running.

The results, shown in Table II, illustrate varied NiMC behavior, dependent on the underlying architecture. This performance degradation ranged from 0% to 60%. On all systems other than Xeon-Phi, Sandy Bridge-X2-FDR-offload and Haswell-X2, STREAM experienced significant (greater than 20%) memory bandwidth degradation due to NiMC. The Piledriver-1600 system and Sandy Bridge-X2-onload stand out by exhibiting a 60% and 51% performance degradation, respectively. The increased 9% degradation on Piledriver-1600 is expected due to a higher network bandwidth to memory bandwidth ratio compared to the Sandy Bridge system. We observe that all three of the systems with variable memory frequency see a decreased impact from NiMC as available memory bandwidth increases.

Of greater interest, our Sandy Bridge systems demonstrated how NiMC might impact onload versus offload networks differently. Both systems utilize Sandy Bridge CPUs, but Sandy Bridge-X2-FDR-offload utilize Mellanox offload network cards, whereas Sandy Bridge-X2-onload uses QLogic onload NICs. There are stark differences between the STREAM Triad results of these two machines. While the offload system sees no performance degradation, we see a 51% performance penalty to the onload system's Triad performance. We observe that the onload-NIC systems (Piledriver-1600/1866 and Sandy Bridge-X2-onload) are the most impacted by competing RDMA flows.

Examining the four offload NIC systems in isolation, we see a bimodal impact of NiMC. When the CPU fully utilizes close to the theoretical memory bandwidth (as is the case of Westmere and Lisbon), competing RDMA traffic can degrade the Triad performance by 16-25%. Westmere and Lisbon, show effective memory bandwidth of 98% and 93% the theoretical memory bandwidth, respectively. This compares to Sandy Bridge and Haswell, which only achieve 74% and 85% effective memory bandwidth, respectively. The decrease to percentage effective memory bandwidth in Sandy Bridge and Haswell leaves additional headroom for the RDMA traffic, so that we see no impact of NiMC. However, as increases to theoretical memory speeds slow down, chip designers must increase their effective utilization in future systems. Furthermore, network speeds are increasing rapidly, with 4X EDR InfiniBand reaching speeds of 12 GBps so that we will again see larger network to memory bandwidth ratios. For these reasons, we cannot rule out the resurgence of NiMC in future offload systems.

For the Intel Xeon Phi system, we observed a small (4%) memory bandwidth decrease. This is due partially to the Phi's unique memory architecture (as compared to traditional CPUs). For instance, all other systems under tests had a single memory controller. The Phi system has eight controllers that control 16 channels of GDDR5 memory. Each core has 64K of L1 cache and a 512k fully coherent L2 cache, which are connected over a bi-directional ring interconnect. Additionally, the Phi runs its own operating system, requiring a dedicated core for OS services. By leaving this reserved core open, we may be leaving additional memory-bandwidth-headroom into which the RDMA traffic can fit.

In summary, we saw that NiMC impacts a range of architectures spanning multiple vendors and hardware generations. Of importance, the NIC architecture (onload vs offload) appears to play a significant role determining the impact

TABLE II: STREAM Triad Bandwidth with and without RDMA-NiMC

machine	Triad no RDMA (GB/s)	Triad + RDMA (GB/s)	diff. (GB/s)	diff. %
Westmere @ 800 MHz, 1066 MHz, respectively	12.9, 16.8	9.7, 12.8	3.2, 4.0	-25%, -24%,
Lisbon @ 800 MHz, 1066 MHz, 1333 MHz, respectively	14.0, 17.9, 19.7	10.8, 14.3, 16.5	3.2, 3.6, 3.2	-23%, -20%, -16%
Piledriver-1600	12.4	7.4	5	-60%
Piledriver-1866	12.7	5.6	7.1	-44%
Sandy Bridge-X2-FDR-offload	77.8	77.6	-0.2	0%
Sandy Bridge-X2-onload	73.4	36.1	37.3	-51%
Xeon-Phi (on-chip bandwidth)	126.4	121.7	4.7	-4%
Haswell-X2	116.6	116.9	0.3	0%

of NiMC, as every system utilizing an onload NIC saw significant degradation. Additionally, as one might expect, the results suggest that increased available memory bandwidth reduces NiMC interference, while decreased available memory bandwidth increases NiMC interference.

V. Proxy-Apps on a Single Node

While STREAM illustrated NiMC effects for worst-case memory-bound applications, STREAM is not necessarily reflective of typical HPC applications. By mimicking the operations of a variety of important scientific problems, the DOE proxy applications (described in §III-B) are more accurate HPC workload representations. We now describe our use of the proxy apps to understand NiMC effects on a single node for realistic workloads with worst-case network traffic.

We used single node experiments to study NiMC effects in isolation from other potential network interference, for example due to application network communication. The applications use Open MPI v1.8 for inter-process communication. Additionally, LULESH and XSBench use OMP for increased on-node parallelism. For these latter hybrid applications, we used the highest performing combination of MPI processes and OMP threads[2]. Once again, we measured application execution times with and without injected RDMA traffic. Reported results are the averages of 10 runs with error bars displaying the standard deviation. For the experiments in §V and §VI, we used the Sandy Bridge-X2-onload system.

Table III shows the application time-to-solution results in the absence and presence of NiMC, and Figure 3 shows the application performance slow-down due to NiMC. These results illustrate several interesting behaviors. First, out of six proxy-apps, we observed significant performance penalties in five, but CNS exhibited almost no performance degradation. Second, three proxy-apps exhibited a performance degradation within 30% of that seen in STREAM. This was unexpected because we selected STREAM as a worst-case indicator of NiMC, due to its intensive memory usage. Furthermore, our pre-experiment hypothesis was that memory-intensive proxy-apps, like HPCCG, would experience the most interference among the proxy-apps; however HPCCG exhibited the second least relative interference. These results suggested that additional sources of contention, beyond the memory-channel bandwidth, may be influencing performance. Accordingly, our next set of experiments were aimed at uncovering these additional contention effects.

[2]For LULESH this was 8 MPI ranks (4 per socket) with 2 OMP threads per rank and for XSBench this was 16 Ranks with 1 OMP thread per rank.

TABLE III: Application Performance with and without RDMA

Application	Run time no RDMA (s)	Run time w. RDMA (s)
CNS	8.75	8.89
HPCCG	4.08	6.07
LAMMPS	177.21	372.04
LULESH	23.07	44.08
SNAP	3.17	5.97
XSbench	72.92	146.44

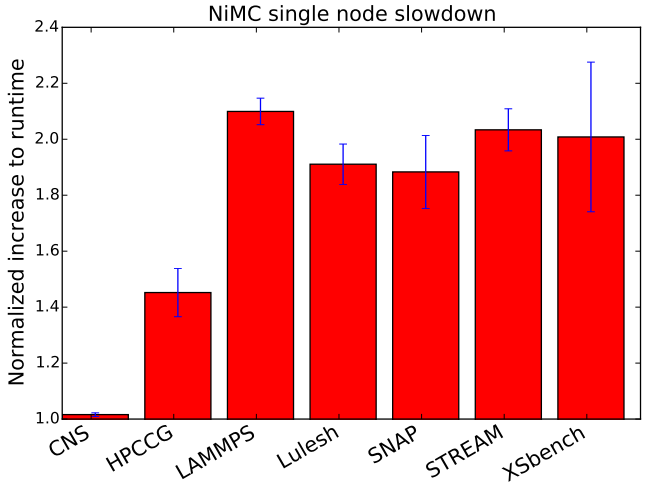

Fig. 3: Normalized impact of NiMC on single node runs.

A. In-Depth Memory Analysis

It can be hard to decipher precisely what is happening in a system with regards to NiMC. Often, to maintain competitive advantages, hardware vendors do not publicly share the details of components such as memory-controllers or network drivers. While it can be difficult to determine NiMC root causes, we glean insights by profiling application activity as measured by available hardware counters (PMCs). In this case, we use OpenSpeedShop (OSS) [25], which can provide PMC collection and analysis for large parallel applications. Using OSS, we found that for all but CNS (the sole application that did not exhibit a NiMC-based performance degradation), with RDMA traffic, NiMC impacted one core more than the other cores.

To understand why this process was performing much slower under RDMA activity, we used OSS to measure the performance of the L1, L2 and L3 caches as well as the total number of stalled cycles in the presence and absence of RDMA

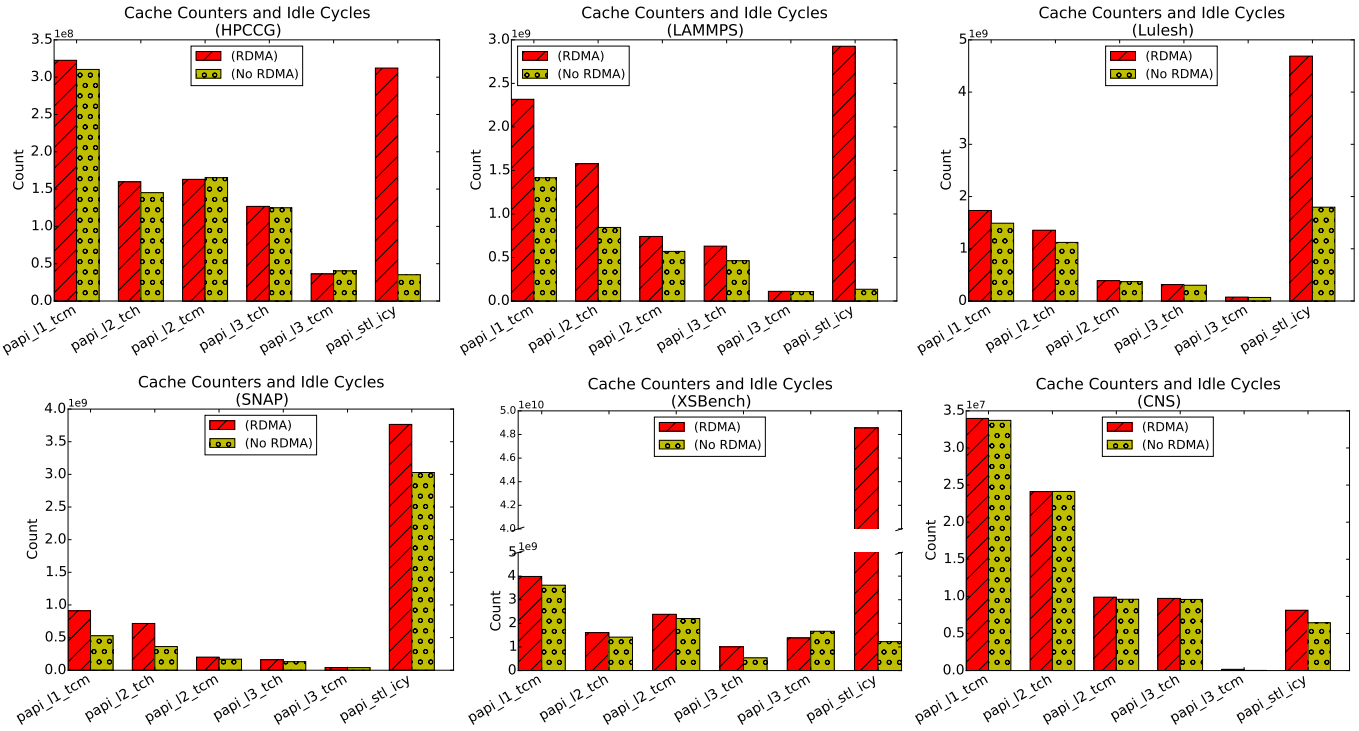

Fig. 4: Comparing performance counters for single node runs of benchmarks and proxy-apps with and without NiMC.

traffic. These results are shown in Figures 4(a) – 4(f)[3]. These results generally show increased L1 cache misses and, to a lesser extent, increased L2 cache misses. Ultimately, there is a significant increase in the number of stalled cycles in the presence of RDMA traffic. The relationship between cache misses and stalled cycles is consistent with Figure 3 in that proxy-apps that have reasonable cache efficiency experience the worst NiMC interference. At the same time, applications with poor cache utilization are less affected by additional cache misses since their cache efficiency is low to begin with. These results also help to explain why only one application process experienced significant performance degradation: the largest differences in cache behavior were observed at cache levels not shared by other cores.

Among the proxy-apps, CNS is an outlier in that it experiences almost no interference from NiMC. CNS demonstrates that the slowdown seen in other runs is not primarily due to CPU (core) perturbance, as CNS receives identical amounts of RDMA traffic and services it in the same manner as the rest of the benchmarks, but observes only a 1.6% slowdown from RDMA traffic processing overhead. In Figure 4(f) we can see that CNS achieves much better cache performance than the other proxy-apps, particularly when contending for resources with a separate flow of RDMA traffic. Specifically, we observe almost no difference in L1 or L2 cache utilization across runs and a negligible number of L3 misses. This is because in CNS, each MPI process utilizes an extremely small amount of memory (approximately 4 MB). Such a small working set of memory leaves space for both the one-sided RDMA and the proxy-app to effectively utilize cache. As a result, there is only a minor increase in the amount of idle cycles as we add interference from the network.

As Figure 2 illustrates, as a DMA transfer is serviced by an onload NIC, some amount of data is distributed throughout the cache hierarchy. This data takes up a larger proportion of space in L1 cache than L2 and L3, which is why we would see a greater impact on the performance at lower levels of the cache. When sending data synchronously, it makes sense that the application would want that information in cache so that the CPU may service communication events faster. However, when this data is sent asynchronously, we do not know when the application will require the written data (if it does at all). In the asynchronous case, loading the data into cache can be benign (as seen with CNS) or create significant bottlenecks (as seen in the other proxy-apps).

1) Correlation Analysis: Our discussion above, based on the performance data shown in Figures 4(a)–4(f), presumes some relationships between the slowdown seen with RDMA traffic and CPU stalled cycles. To determine the strength of these relationships, we performed a correlation analysis between runtime and stalled cycles and cache misses as well as between the number of stalled cycles and the cache related counters. We used Pearson's r for correlation,which measures linear correlation between two variables. The analysis results (Table IV) show that without additional RDMA traffic, the application runtimes are very strongly correlated to L1/L2/L3 misses but are not correlated with stalled cycles. Stalled

[3]total cache misses/hits/accesses (TCM/H/A), idle cycles (ICY)

TABLE IV: Performance Monitoring Counter Correlations Across All Applications

	Corr. Metric	Stalled Cycle	L1 Miss	L2 Miss	L3 Miss
No RDMA	Time	-0.04	0.941	0.946	0.930
	Stalled Cycles	N/A	0.086	0.030	0.068
RDMA	Time	0.912	0.959	0.978	0.925
	Stalled Cycles	N/A	0.870	0.973	0.997

cycles for the non-RDMA case do not meet required levels of significance to assert that a relationship exists. For the RDMA traffic case in Table IV a very strong correlation exists between runtime and stalled cycles with a 95% certainty. We see that there is a very strong correlation between the stalled cycle count and the cache misses, particularly L3 misses, showing that the stalled cycle increase is almost certainly due to misses throughout the cache hierarchy and requests to main memory. This correlation coupled with the large rise in stalled cycles that occurs when introducing RDMA traffic leads us to conclude that the increase in runtime observed is due to time waiting for the memory subsystem.

Though cache pollution from RDMA is correlated with an increased number of stalled cycles, it is not necessarily the only factor contributing to NiMC. Other contributing factors can include: the policy and scheduling of the memory controller(s), such as open-page row-buffer management, the degree of concurrent operations the memory controller(s) can handle, the number of memory channels, and how these memory channels are written to, for example, ganged or unganged. In order to fully model the impact of NiMC and offer mitigating solutions, these factors must also be considered.

VI. Large Scale Evaluation

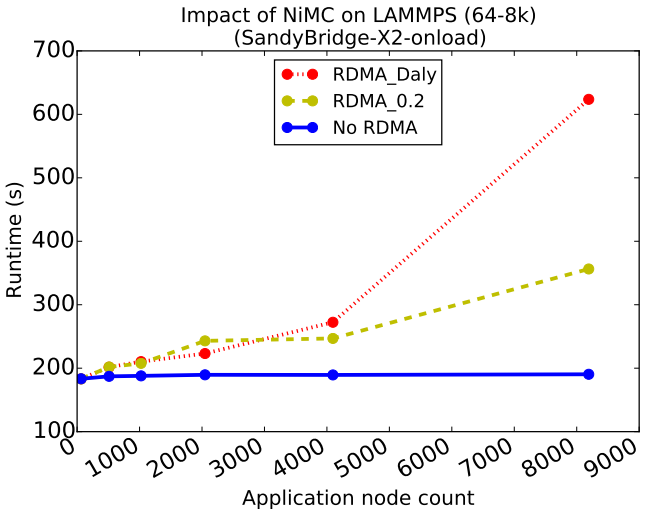

Fig. 5: Impact of NiMC on LAMMPS for an onload system.

From the previous studies, we gained a better understanding of the underlying causes of application interference in the context of a single, isolated node targeted with an unlikely high volume of RDMA traffic. Our final study examined NiMC for applications at scale with realistic RDMA traffic volumes.

TABLE V: Number of concurrent RDMA writes

Application node (rank) count	Writes/s (Daly) QDR-onload	Writes/s (Daly) FDR-offload	Writes/s (0.2%)
64	0	0	0
512	1	1	1
1024	2	2	2
2048	5	6	4
4096	15	17	8
8192	42	47	16

As with our single node experiments, we use the Sandy Bridge-X2-onload cluster executing a series of weak scaling LAMMPS experiments. We selected LAMMPS for the large scale runs for multiple reasons. First, of all our workloads, LAMMPS is the only real application: it is not a proxy app nor a contrived benchmark. Second, LAMMPS scales very well for the size of system under study. Finally and most importantly, LAMMPS is widely regarded as an application that is resistant to external interference [18]. Therefore, LAMMPS represents a good challenge when trying to realize performance degradation due to external perturbations like RDMA traffic.

As in our previous experiments, in addition to the target nodes running our application, we reserved an additional set of origin nodes that push our RDMA traffic to the target nodes. However, unlike in our previous experiments, we limit concurrent writes to a small subset of the total nodes. We also limit the duration of each write operation. We use a hypothetical uncoordinated, in-memory or disk-less checkpointing protocol[4] to shape our RDMA traffic pattern. Since there is no known optimal checkpoint interval for uncoordinated checkpointing, we use Daly's estimate [5] to derive optimal coordinated checkpoint intervals. We use Daly's estimate to compute the average number of processes simultaneously taking a checkpoint using a five year mean time to interrupt, and use this number as the number of concurrent writers, shown in Table V. We compute RDMA write duration by optimistically assuming that all checkpoints take 46 thousand message iterations (equivalent to ≈1s for 4X-QDR IB and ≈0.5s for 4X-FDR IB). Each data point in Fig 5 represents the minimum runtime of 5 runs. We chose the minimum because (1) the minimum is the hardest metric to overcome, when showing the existence of NiMC at scale. (2) it shows the impact of NiMC rather than the impact of contention on the network or I/O subsystem from other jobs that are outside of our control.

The results in Figures 5 show that as we scale up the number of application processes the impact of NiMC becomes significant. Despite the fact we decreased write duration to a single second, scaling up the number of application processes greatly amplified the magnitude of interference. This is similar to phenomena seen in the research of OS noise, where scale amplifies the magnitude of the overall perturbence [12], [22]. Even with a constant 0.2% of total nodes as simultaneous writers, time-to-solution nearly doubles

[4]Contemporary approaches for in-memory checkpointing use a coordinated protocol in which all processes take a checkpoint simultaneously. However, for next generation systems there is a concern that coordination at massive scale can become prohibitively costly.

at scale due to NiMC. While we've increased the pressure on the network by introducing additional RDMA writes, we will show in §VII-B that this additional traffic is not responsible for the increase to runtimes.

VII. Solutions for NiMC

We evaluated several approaches for reducing the impact of NiMC, specifically: (1) offload hardware, (2) core reservation, and (3) software based network throttling. All of these techniques have been applied in other areas of research [6], [10], [16], [1] but this work is the first to evaluate their effectiveness in mitigating NiMC. For the results in Figs 6(a) and 6(c) we present the best (min) baseline LAMMPS runtimes and the median runtimes for NiMC based on Daly's volume. For the platforms evaluated, the difference between the minimum and the median for Daly's volume runs was negligible.

A. Offload NICs as a solution

In § IV, it was shown that NiMC did not negatively impact the performance of the most recent offload systems for the benchmark STREAM. In this section we show that offload NIC's continue to provide a solution to NiMC at scale for real applications. In Fig 6(a) we ran LAMMPS on the Sandy Bridge-X2-FDR-offload system, weak scaling up to 8192 processes. Comparing the results of *No RDMA* and *Daly*, it is evident that NiMC does not have any observable impact on offload system performance at scales of up to 8192 processes. From these results, we conclude that offload NICs provide a solution for modern systems that would otherwise experience NiMC. The main drawback to offload NICs is their greater monetary cost compared to their onload equivalents. However, this is not a guarantee that future systems will be unaffected by NiMC as the disparity between network and memory bandwidth shrinks. Even though none of the most recent offload systems were impacted by NiMC – on slightly older systems (Westmere and Lisbon) we observed a 16-25% decrease to STREAM Triad performance due to NiMC. We believe the future viability of an offload solution is dependent on how fully CPUs utilize memory bandwidth and by future network bandwidth increases.

B. Core Reservation

Dedicating a core to service communication is another possible solution to mitigate NiMC. This reduces the memory throughput of the CPUs, and sets aside separate cache resources for handling network data. The downside to core reservation is that in the absence of RDMA communication the reserved core is wasted. To test the effectiveness of core reservation, we repeated the STREAM and LAMMPS tests of §IV and §VI while reserving one core per node to process communication, binding the QIB driver to the reserved core. Additionally, we evaluated core reservation for a modified version of STREAM which uses array sizes designed to fit entirely in last level cache (LLC). The modified STREAM allows us to evaluate LLC performance, with respect to NiMC. Both of these tests can be seen in Figs 6(b) and 6(c). Interestingly, Fig. 6(b) presents an intersection near 400-500 MBps on the x-axis, where a core reservation strategy begins to provide a performance benefit. As the RDMA bandwidth increases towards 3000 MBps we see performance gains by setting aside a core. This suggests that a dynamic strategy for reserving a core to service the network may be an attractive approach for future systems.

In Fig 6(c), we use identical input files as the previous section, however we only utilize N-1 cores per node for the application. These results show that core reservation continues to be an effective strategy to prevent NiMC at scale, independent of the volume of RDMA traffic. These results clearly demonstrate that contention on the network is not a factor in the increase to runtime seen in § VI, Fig 5. This can be observed in Fig 6(c), where the cases with and without RDMA traffic have only a 0.1% difference in runtime, despite contention on the fabric that would be present in the RDMA results. This allows for a quantification of the induced network contention due to the RDMA streams, which is much less than the observed contention due to NiMC. Of minor note, there is a 10% increase to the runtime of the runs that utilize 480 processes compared to the runs of 60 and 960 processes. After discussion with LAMMPS developers it was determined to be the result of increased communication due to a less efficient domain decomposition for 15 cores per node. Overall, the results suggest that core reservation incurs a runtime increase of 6.4% compared to 16-core to 15-core runs of LAMMPS without RDMA traffic. While a 6.4% increase to runtime is costly, it costs significantly less than the 330% increase without core reservation (Fig. 5).

C. Software-based Solutions

There are several methods for throttling or shaping the traffic sent over the network. One such method is to artificially throttle the throughput of the network so that the same total volume of traffic is sent over a longer time period. The practicality of throttling is partially dependent on the significance of the network data to the application. It is important to remember that traffic throttling increases the time to deliver the data, but makes more memory bandwidth available to the CPU. If the network data is part of the application's critical path and the application is executing faster (due to the increase in available memory bandwidth), throttling may leave the application stalled. In Fig. 6(b) we plot the impact that varying network speeds have on STREAM DRAM and LLC performance. Performing a least square linear regression shows that for every bit per second (bps) of RDMA bandwidth we add we slow the DRAM performance by 14 bps. When considering LLC, this tradeoff becomes even more expensive as 1 bps of RDMA bandwidth reduces LLC bandwidth by 22 bps. On modern HPC systems where memory performance is a highly valued commodity, the large disparity of this tradeoff makes network throttling less appealing as a solution compared to a hardware offloading. Additionally, for the system evaluated, software-based throttling is only effective if you can reduce the amount of network traffic to under 500 MBps. If an RDMA service requires more than 500 MBps network bandwidth, core reservation becomes a better solution.

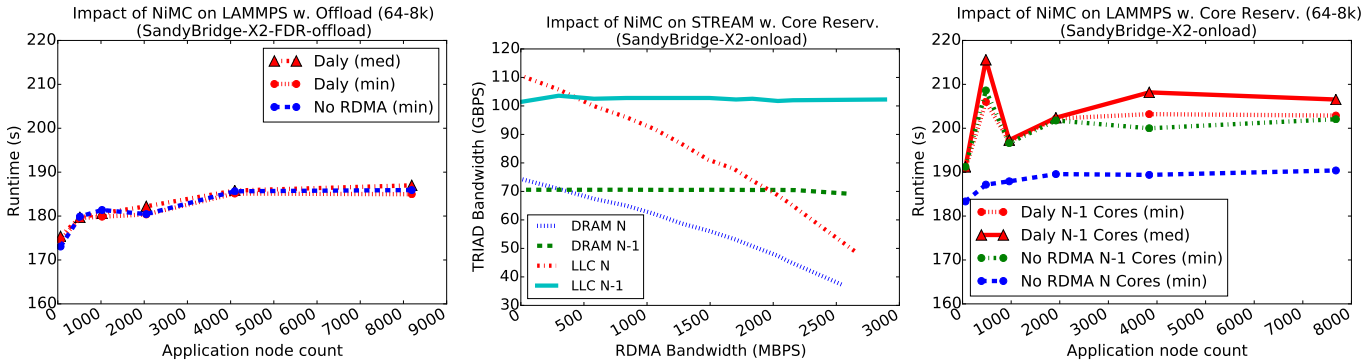

Fig. 6: Fig 6(a) highlights the performance of offload cards at scale for the application LAMMPS. Fig 6(b) shows the relationship between RDMA traffic and available memory bandwidth in DRAM and LLC using N and N-1 cores. Every bit per second of RDMA traffic reduces DRAM and LLC bandwidth by 14 and 22 bits per second, respectively. Fig 6(c) demonstrates a core-reservation solution at scale for the application LAMMPS.

In summary these results show that a hardware offload solution is ideal for modern systems, Though (as Westmere and Lisbon demonstrate) future effectiveness is dependent on trends in CPU memory bandwidth utilization and network speeds. Secondly, many systems utilize lower cost onload NICs. For these systems there are two approaches to mitigate NiMC. The first approach of throttling the network is limited in its effectiveness. Specifically there is a crossover point where the amount of network traffic becomes large enough that core reservation becomes a better solution. This crossover point will vary system to system but was 500 MBps for our evaluated platforms. Lastly, core reservation allows for unthrottled RDMA bandwidth but at a base-level increase to runtime of 6.4%. Though we provide general guidelines, determining the best solution for each system requires a knowledge of the application, services and underlying hardware.

VIII. Related Work

Memory contention has existed in many platforms over time, and tools have been developed to help understand its impacts [7]. Concerns about the ability of memory technologies to keep up with the bandwidth requirements of ever greater numbers of cores have been expressed [24]. However, code developers also deal with this issue through optimizing their code for cache use [21].This motivated investigations into the causes of off-chip memory contention [30] on modern systems for parallel applications, providing insight into the behavior of core heavily optimized for cache performance and those that are less tuned. Given that architectures are designed to balance memory bandwidth with the ability of the cores to saturate said bandwidth, the impacts of NiMC can push a system on the edge, degrading optimized code performance.

Concerns over memory subsystem performance at extreme scale, particularly with the expected growth in core counts, have prompted investigations into memory bandwidth reductions [27] and contention [3]. Tiwari et al. [27] proposed a model for studying the anticipated reduction in per-core bandwidths expected in the Exascale time frame by varying the memory frequency on a single node of the Gordon supercomputer as SDSC. While Tiwari et al.'s model was motivated by a desire to explore the per-core memory bandwidth reduction expected for future extreme-scale systems, it was developed and tested on a single compute node. As such, the model does not account for any source of memory traffic from the network.

Casas and Bronevetsky's work [3] is the closest to this work in terms of memory contention studies. Like the Memory Bandit tool [7], they seek to create "memory bandwidth interference" in order to observe the impact on application performance. Unlike the Memory Bandit [7] work, Casas and Bronevetsky can purposefully perturb different levels of cache while introducing threads that create memory traffic unrelated to the executing application. They present methodology to introduce main memory traffic with minimal cache impact for studying off-chip memory contention. Their observations of application slowdown in the worst case of 20%-35% is inline with was observed from the STREAM/RDMA single node tests in this paper. This is expected as their method of introducing interference for the application created main memory contention. However, the main difference between this work and ours is the source of the memory contention. While Casas and Bronevetsky introduce the contention from cores, they do not account for RDMA traffic.

To the best of the authors' knowledge, this the the first work to consider memory contention due to asynchronous communication. While these topics have been studied separately before, no prior studies have investigated the impact of NiMC.

IX. Conclusions

In this work, we introduced the concept of NiMC and demonstrated its impact for a variety of current HPC system architectures. We showed that NiMC is a concern for both onloaded and offloaded networking hardware, with the onloaded hardware observing the largest performance impact. For all but one of our applications, we observed significant performance impact on modern onload systems due to NiMC, and we showed that the observed NiMC impact was not significantly

attributable to CPU contention or network contention. Using real applications at large scale, we showed that NiMC can lead to significant performance degradations even in applications like LAMMPS that have previously demonstrated performance robustness in the presence of other types of system noise. While this work only examined InfiniBand based networks, such networks are relevant for HPC as evidenced by the recent $325 million CORAL procurement [31], a 150 petaFLOP IB-based system.

We explored several causes of NiMC – results which will inform future system design in both the existence of NiMC as well as the potential methods to reduce its performance impact. In addition, this study offers evidence to system software and application developers that NiMC should be taken into account when developing software that may be co-located with other applications or services that consume network bandwidth, even for small durations of time.

Lastly, we evaluated three strategies to mitigate NiMC, namely offload NICs, core reservation, and network throttling. Our results suggest that Offload NICs appear to provide the best, albeit most expensive, solution to NiMC on modern systems, provided there is sufficient headroom between theoretical and observed CPU memory bandwidth. In the event a cluster utilizes an onload NIC, setting aside a CPU core to service the network is a viable solution for onload systems, but incurs a runtime penalty proportionate to the processing power of the core (6.4% in our study). The current disparity in the way memory bandwidth is provisioned between RDMA and the CPU makes the third solution (network throttling) attractive only if the required RDMA bandwidth is below specified thresholds (Fig 6(b)).

REFERENCES

[1] H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, and S. Klasky. Extending I/O through high performance data services. In *IEEE Conf. on Cluster Computing*, pages 1–10, Aug 2009.

[2] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proc. of SC13: Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, page 41. ACM, 2013.

[3] M. Casas and G. Bronevetsky. Active measurement of memory resource consumption. In *Parallel and Distributed Processing Symp., IEEE 28th Intl.*, pages 995–1004. IEEE, 2014.

[4] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf. Software design space exploration for exascale combustion co-design. In *Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 196–212. Springer Berlin Heidelberg, 2013.

[5] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2006.

[6] M. G. F. Dosanjh, R. E. Grant, P. G. Bridges, and R. Brightwell. Re-evaluating network onload vs. offload for the many-core era. In *IEEE Intl. Conf. on Cluster Computing*. IEEE, 2015.

[7] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Bandwidth bandit: Quantitative characterization of memory contention. In *Intl. Symp. on Code Generation and Optimization (CGO)*, pages 1–10. IEEE, 2013.

[8] ExaCT Co-Design Center. https://ccse.lbl.gov/ExaCT/index.html 4/1/15.

[9] A. S. for the Top Ten Exascale Research Challenges. Top ten exascale research challenges. Technical report, U.S. DoE, ASCR, Feb 2014. http://science.energy.gov/ /media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.

[10] R. E. Grant and A. Afsahi. Improving energy efficiency of asymmetric chip multithreaded multiprocessors through reduced os noise scheduling. *Concurrency and Computation: Practice and Experience*, 21(18):2355–2376, 2009.

[11] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 2009.

[12] T. Hoefler, T. Schneider, and A. Lumsdaine. Characterizing the influence of system noise on large-scale applications by simulation. In *Proc. of the Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[13] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, et al. Exploring traditional and emerging parallel programming models using a proxy application. In *Proc. of the 27th IEEE Intl. Symp. on Parallel and Distributed Processing (IPDPS)*, pages 919–932. IEEE, 2013.

[14] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale computing study: Technology challenges in achieving exascale systems. 2008.

[15] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995.

[16] J. C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar. Using asymmetric single-isa cmps to save energy on operating systems. *IEEE micro*, (3):26–41, 2008.

[17] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2010.

[18] A. Morari, R. Gioiosa, R. W. Wisniewski, F. J. Cazorla, and M. Valero. A quantitative analysis of os noise. In *Proc. of the Parallel and Distributed Processing Symp. (IPDPS)*, pages 852–863. IEEE, 2011.

[19] MPI Forum. MPI: A message-passing interface standard version 3.1. Technical report, University of Tennessee, Knoxville, 2015.

[20] OpenFabrics. https://www.openfabrics.org/ (visited Oct. 2014).

[21] S. Perarnau, M. Tchiboukdjian, and G. Huard. Controlling cache utilization of HPC applications. In *Proc. of the Intl. Conf. on Supercomputing*, pages 295–304. ACM, 2011.

[22] F. Petrini, D. Kerbyson, and S. Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of ASCI Q. In *Proc. of the Intl. Conf. for High Performance Computing, Networking, Storage and Analysis*, pages 55–55. IEEE, 2003.

[23] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[24] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for cmp scaling. In *ACM SIGARCH*, volume 37, pages 371–382. ACM, 2009.

[25] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, and S. Cranford. Open— speedshop: An open source infrastructure for parallel performance analysis. *Scientific Programming*, 16(2-3):105–121, 2008.

[26] N. R. Tallent, A. Vishnu, H. Van Dam, J. Daily, D. J. Kerbyson, and A. Hoisie. Diagnosing the causes and severity of one-sided message contention. In *Proc. of the 20th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 130–139, New York, NY, USA, 2015. ACM.

[27] A. Tiwari, A. Gamst, M. A. Laurenzano, M. Schulz, and L. Carrington. Modeling the impact of reduced memory bandwidth on HPC applications. In *Euro-Par 2014 Parallel Processing*, pages 63–74. Springer, 2014.

[28] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz. XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. In *PHYSOR 2014*.

[29] C. Trott, C. Vaughan, S. Mashayak, M. Brown, C. Ponder, P. Crozier, and F. Reid. Lammps benchmarks. http://lammps.sandia.gov. 2015-10-12.

[30] B. M. Tudor, Y. M. Teo, and S. See. Understanding off-chip memory contention of parallel programs in multicore systems. In *Proc. of the Intl. Conf. on Parallel Processing (ICPP)*, pages 602–611. IEEE, 2011.

[31] U.S. DOE. Fact sheet: Collaboration of Oak Ridge, Argonne, and Livermore (CORAL). http://energy.gov/, December 2014.

[32] R. J. Zerr and R. S. Baker. SNAP: SN (Discrete Ordinates) Application Proxy: Description. *Los Alamos National Laboratories, Tech. Rep. LA-UR-13-21070*, 2013.