SAND2014-17733C

Re-evaluating Network Onload vs. Offload for the Many-Core Era

***Abstract*—This paper explores the trade-offs between onloaded versus offloaded network stack processing for systems with CPU frequencies aligned with current trends in CPU multicore design and many-core architectures. While current hardware leads to similar performance between the onload and offload approaches, this study demonstrates that current offloaded networking solutions can provide equivalent or superior performance to onloading when operating at lower CPU frequencies. While the debate between network onload vs network offload proponents has been an active one for some time, this work is the first to provide experimental evidence of the impact on CPU technology trends on the future of networking stack processing.**

I. Introduction

Processor core frequency gains have declined significantly since the beginning of the multi-core era. With the transition to many-core architecture, frequencies are expected to remain essentially constant and may even decline. This trend has significant implications for network subsystem design, for example onloading versus offloading network protocol processing. Onload systems seek to leverage excess on-chip processors for protocol processing, while offload systems seek to leverage specialized NIC processing; As a result, the performance and power costs of host network protocol processing significantly impact this tradeoff.

The onloaded approach assumes that the performance gap between the available CPU cores and dedicated ASIC offloading hardware is minimal, and where it is not, it can be overcome by allocating more CPU cores. This assumes that the performance of the networking stack can be greatly improved through parallelization. While many networking stack functions can be done in parallel, the semantics of MPI enforce ordering that makes full parallelization difficult. Some approaches have been proposed [1] for methods of improving the parallel performance of MPI, but the current state of mutlithreaded MPI implementations illustrates the difficulties of such approaches. Other work has declared that multi-threaded MPI may not be a viable approach for future HPC [2].

The emerging trend of many-core architectures does not help address the networking issues arising from reduced core frequencies. Using these cores for network stack processing has further implications beyond lower core frequencies. Many-core compute units are, by design, more simplistic than modern x86 cores, with current generation many-core architectures like the Intel Xeon Phi lacking support for out-of-order processing. While future generations may include expanded core features, such as support for out-of-order execution, it is unlikely that the feature set supported by such cores will approach that of the full-featured state-of-the-art server class CPUs. Dedicating large power-hungry cores, such as a Xeon x86 or AMD equivalent core, to network processing has thus far been an acceptable compromise. However, with increasing concerns about the power consumption of world-class systems, the power efficiency of such an approach deserves to be examined. As world-class capability supercomputers begin to encounter a power wall for practical as well as administrative reasons, the ability to provide high performance networking, while reducing power consumption, is a key concern for next generation systems.

In this paper, we present an initial evaluation of the implications of host processing speed changes on onload vs. offload network protocol processing. We do so by examining the network performance and power consumption when running both onloaded and offloaded networking hardware alongside a consumer-class AMD CPU operating at different frequencies. By using identical systems where the only changes are in the high performance networking cards used, we isolate the differences between the two networking approaches and quantify the impact of processor frequency on networking performance.

The remainder of this paper is organized as follows. In Section II, we discuss the background of onload and offload protocol processing and architectural changes that motivate this study. We then discuss our experimental setup and methodology in Section III and present and analyze the results of these experiments in Section IV. We then discuss the implications of these results in Section V. Following this, we discuss other related work in Section VI. Finally, we present our conclusions and describe directions for future work in Section VII.

II. Background

Two competing models of high-performance networking, onloaded and offloaded networking, have been adopted by various vendors in systems over the years. InfiniBand, one of the primary HPC networking architectures, has both offload and onload network adapter (HCA) implementations. For example, Mellanox InfiniBand HCAs provide full featured offload engines, while vendors such as QLogic sell onload-based HCAs with simplified NIC hardware and a full-featured software stack providing the remainder of the functionality through the host CPU.

The high level differences between onload and offload are depicted in Figure 1. The major difference is that offloaded message processing is done by a dedicated chip on the NIC while onloaded is done on the system's CPU. The message processing can include all levels of message processing, from MPI to low level protocol processing. Offloading allows the manufactures to make hardware optimizations to the dedicated chip, to boost performance over onloaded NIC. It is possible to hybrid these two approaches; for instance, some offload infiniband cards do message matching onloaded on the CPU.

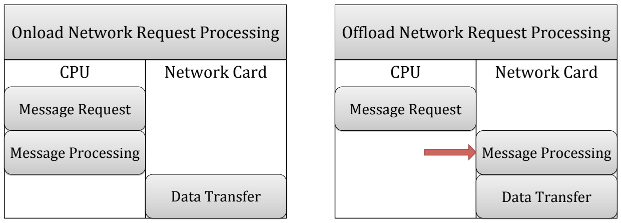


Fig. 1. A high-level depiction of the differences between onload and offload

During the time the onload model was developed, CPU capabilities and speeds were increasing with each successive generation. The end of Dennard scaling and the introduction of multi-core CPUs lead to further arguments in favor of the onload model. In particular, onload proponents have argued that such a model allows the use of other cores on a system that might not be able to be fully taken advantage of by an application, particularly during communication periods.

However, two recent trends are also potentially working counter to this approach:

- The gradual flattening and even regression in core speeds in traditional processors due to power and cooling issues
- The emergence of many-core architectures such as the Intel Xeon Phi with dramatically reduced single-core performance

These two trends potentially limit the ability of these host processors to keep up with the performance demands of onload HPC networking systems. For example, recent work has shown that many-core Xeon Phi processors limit MPI message processing rates in HPC systems [3].

In addition, large scale systems will be placed under power/energy constraints in the future [4]. This further complicates the offload/onload tradeoff space—general purpose processors potentially draw more power than specialized network-oriented offload processors, but may also be able to dynamically change power draw in response to changing system power caps. However, such changes could also cause significant fluctuations in networking performance, to the detriment of application performance.

As a result, it is important to understand the ramifications that power changes will have on network performance and explore the tradeoffs between onloaded and offloaded networks. It is similarly important to understand the potential for power and energy savings during communication phases. If such savings can be obtained without significant performance impact, or if the performance impact is tolerable, then leveraging the available savings will be important to future supercomputer efficiency.

III. Experimental Methodology and Setup

To evaluate the power and performance trade-offs between onload and offload networking approaches, we conducted experiments with both offload and onload InfiniBand cards. These cards were placed in systems instrumented for power collection, and host CPU power consumption was controlled to understand the impact of CPU speed on network performance and power consumption in different applications. In the remainder of this section, we provide additional details on our experimental setup; the hardware system on which these results were gathered and the micro and application benchmarks we used.

A. Hardware and Data Collection Setup

The evaluation of the onloaded vs. offloaded networking approaches was performed on 4 nodes of a cluster Nodes in this cluster each have a 3.8 GHz AMD Fusion APU, 16 GB of memory, and run Linux kernel version 2.6.32 (RHEL 6). For onload experiments, we installed a QLogic 4X QDR InfiniBand HCA in each node, while we used used a Mellanox ConnectX-3 4X QDR InfiniBand HCA for offload experiments. In both onload and offload cases, a Qlogic 12200 36-port InfiniBand switch connected the InfiniBand HCAs.

Power measurements for the experiments were collected using a power measurement devices installed in the cluster. This power measurement system is an out-of-band measurement device that collects fine grained samples for multiple system components through the use of a mother measurement board and risers on system components. They enable the inline reading of system power on a per component basis without impacting the performance or power consumption of the node. All power information output by the device used a separate out-of-band network to deliver the information to a central collection node that was not participating in the testing. Further detailed information on these devices can be found in [5]

B. Benchmarks and Applications

To further compare the onloaded vs. offloaded networking approaches, we analyzed the performance and power comparisons on benchmarks and applications. In particular, we compared onloaded and offloaded runs in the MILC application [6] and the LULESH [7] benchmark. Furthermore, we ran profiling runs, using MPIP, to determine why these applications react to the network cards differently.

The netpipe microbenchmark suite is a tool designed to test the bandwidth and latency of a network [8]. Of its tests, we ran the streaming, streaming without cache effects, and send-recv ping-pong tests over different message sizes ranging from two bytes to one megabyte. The MIMD Lattice Computation (MILC) application was the first application benchmark we used [9]. It was developed to study quantum chromodynamics and uses four dimensional lattice computation using a halo exchange communication pattern. We used an input deck based on the weak scaling NERSC 6 acceptance benchmarks [10]. In particular, each node has lattice of size 8x8x8x9. The Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) application is the second large benchmark we used [11]. LULESH is designed to be a representative application for larger hydrodynamics codes. For all of our tests, we ran a 120^3 problem for 130 iterations. There is a constraint on the number of MPI ranks used for this code; it has to equal a cube of an integer. Because of this, we fixed the number of MPI ranks at 8, adding an extra OMP thread to each rank in the four node case.

All three benchmarks were run three times for combinations of the following variables: InfiniBand card, number of nodes, and CPU frequency. We used the InfiniBand cards mentioned in section III-A to test both onload and offload. The number of nodes varied between 1, 2, and 4 for MILC, 2 and 4 for LULESH, and was not a variable for netpipe. The CPU frequency was modified using DVFS to the values 1.4GHz, 1.9GHz, 2.4GHz, 2.9GHz, 3.4GHz, and 3.8GHz. We collected the overall runtime and power statistics of these applications as well as MPI profiling information on a couple of separate runs.

IV. Experimental Results

A. Microbenchmark Evaluation

The netpipe microbenchmarks [8] were used to examine the impact CPU frequency has on both the power draw and performance of the different networking approaches. Figure 2 shows the stream bandwidths along with the power consumption of both onloaded and offloaded networks. Aside from the obvious protocol switching points (MPI eager to rendezvous) causing plateaus and in some cases dips in performance between messages sizes, the important observation to make from these figures is the spread in performance between the highest CPU frequencies and the lowest. The onloaded method expectantly loses some performance when CPU frequency is lowered, resulting in an near halving of bandwidth between the 3.8GHz and 1.4 GHz frequencies. For the offloaded network, the reduction in CPU frequency impacts network performance by a much smaller degree. The only noticeable difference in behavior occurs when the lowest CPU frequency is used. There is more variance in the bandwidth curve than the other scaling points, suggesting the microbenchmark may not be able to keep up with the network events at this speed. The performance gaps between the CPU frequencies remains relatively similar in terms of percentage of performance loss for all message sizes, including the smaller message size results.

Removing caching effects from the results as shown in Figure 3 has little impact on the offloaded case, except for a slightly reduced throughput. This results in less of a gap between the slowest speed (1.4GHz) and the other clock speeds. The impact on the onloaded case is similar for large messages. However, for small and medium sized messages, differences in throughput exist. The drop occurring after 8KiB message sizes is attributable to the eager-rendezvous protocol switch-over in MPI. The drop occurring at 64KiB message sizes is attributable to the virtual maximum transmission unit (VMTU) maximum of 64KiB, necessitating multiple calls to the onloaded networking stack.

Examining send-recv performance with bi-directional ping pong (as opposed to the previous unidirectional streams) in Figure 4, it can be observed that the results are similar to the stream results with cache effects. The increase in throughput is due to the bi-directional nature of the test, but generally aligns with the unidirectional results, in that they are reasonably within twice of the unidirectional throughput.

A key observation from these results is the relatively small trade-off in throughput performance from transitioning between CPU frequencies for both onload and offload at 2.9GHz and higher. We concentrate on the results including cache effects for the purpose of this analysis, but the percentages are similar for the case in which cache effects have been removed. For the onload case 36.4% of the power consumption can be saved while only losing 2.5% of throughput, while for the offloaded case 22.5% of power can be saved while only impacting performance by 0.5%. The offloaded network provides better results in scaling frequency back below the 2.9GHz level, providing power consumption savings of approximately 30.5% while impacting performance by only 1.5% when switching from a 3.8GHz clock rate to 1.9GHz. For the onloaded case, this is impractical, as using a lower frequency such at 1.9GHz would result in a performance loss of 35.1%. This emphasizes the potential issues that may arise when using many-core systems with slower and less powerful compute cores. It also highlights that if such network onload approaches are to be practical on future many-core systems, parallelism for communication will be a key component in achieving performant network throughput.

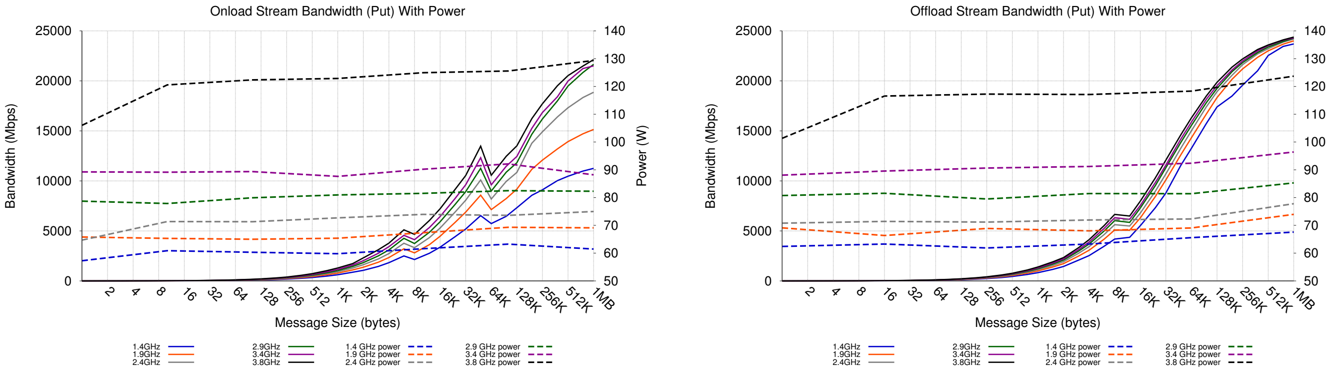
Finally, we examine the latency impacts from slower CPU frequencies on the onloading and offloading approaches. Figure 5 shows that the latency penalties associated with lower CPU frequencies occur for both onloaded and offloaded networking. However, the offloaded networking approach leads to convergence of latencies for successively lowering CPU frequencies at smaller message sizes, and all CPU frequencies eventually converge at 1MiB message sizes. For small messages under 512 Bytes, the offloaded networking approach has a flat latency curve, while the onloaded case has a upward slope at smaller message sizes.

B. Application Benchmark Evaluation

We used application benchmarks to examine the performance and power tradeoffs in realistic workloads. Using MILC and LULESH, we measured the runtime and power usage at different node counts and CPU speeds to compare onload and offload. Then we ran MPI profiling tests to compare the results of the two applications. It should be noted that there was not much variance in either the runtime or power of the application benchmarks; The standard deviations of 80% of the runs were below 1% of the mean with only 2.5% being greater than 2%.

1) Runtimes and Power: Figures 6a and 6b show the power and performance results of MILC. The tests show the runtime change over the CPU frequency for each of the different node counts. Because the problem size is scaled to the number of nodes, the runtime increases when adding nodes, for instance the four node, onloaded case at 1.4GHz takes over 17 minutes more than the one node, onloaded case. This can be attributed to a combination of the extra computation at the boundary and the communication time between the four nodes.

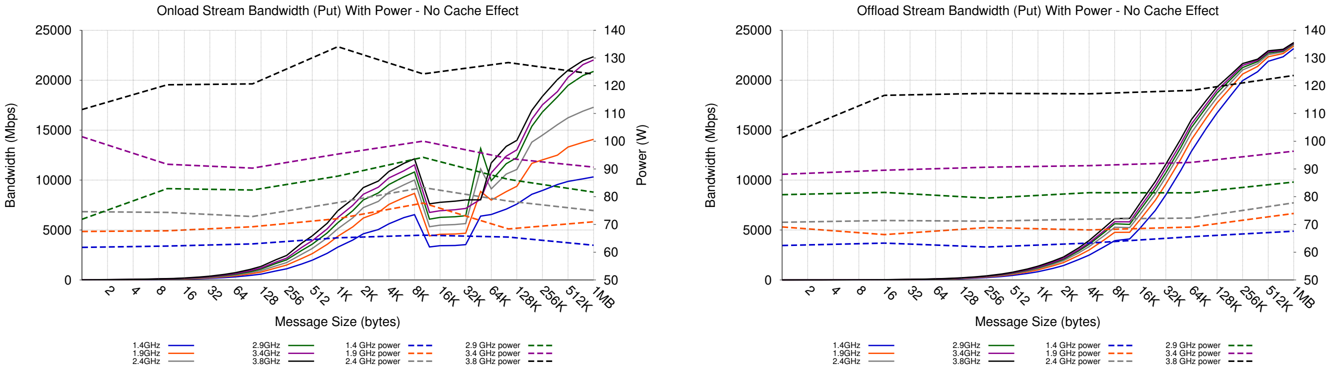
In all of the cases we measured, the offload version takes less time than its onload counterpart. For four nodes, it ranges from a 7.7% to a 10.6% difference between the two, for two nodes, the range is from 5.3% and 7.8%, and even the single node case had a small but consistent performance benefit, ranging from 0.9% to 3.1%. These differences all steadily decline when we increase the clock speed. This shows that while the offload cards have a significant effect on most of the test cases, they have a more significant performance impact on low frequency cores. The power usage of MILC has less



(a) Onload

(b) Offload

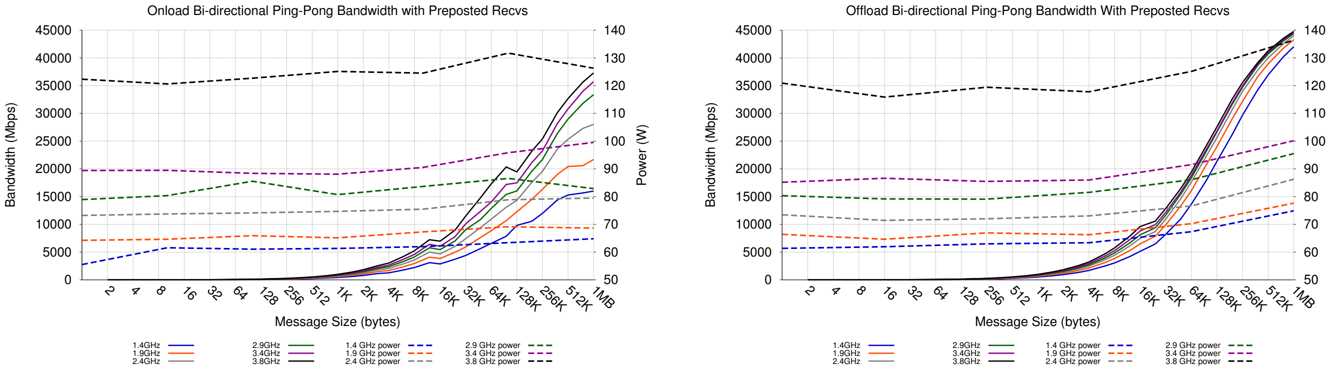
Fig. 2. Onload stream vs. offloaded stream with varying CPU frequencies



(a) Onload

(b) Offload

Fig. 3. Onload stream vs. offloaded stream with varying CPU frequencies without cache effects



(a) Onload

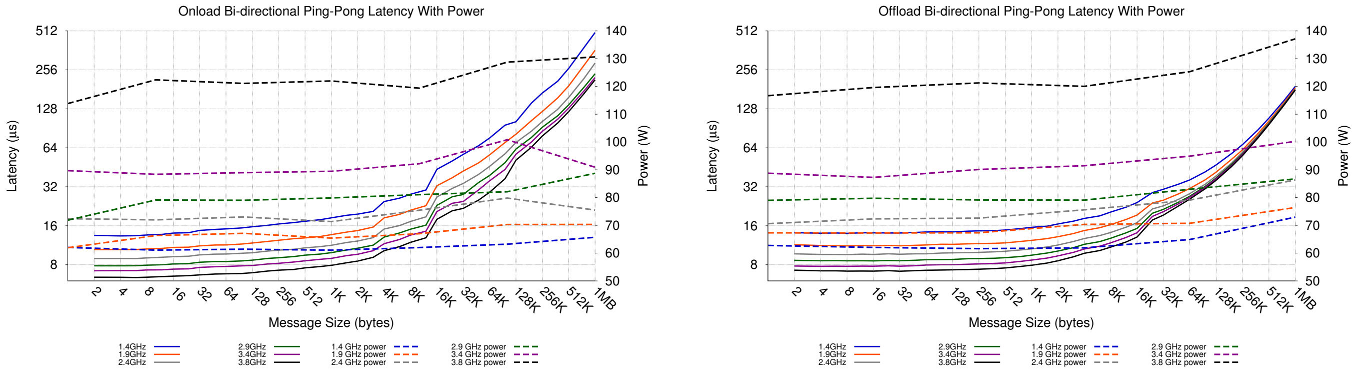
(b) Offload

Fig. 4. Onloaded vs. offloaded bi-directional ping-pong with send-recv and preposted recvs

significant differences. The offload ones used between 1.1% and 3.2% more power than their onload counterpart.

Figures 7a and 7b show the power and performance results of LULESH. The tradeoffs are less distinctive here. The difference between runtime and power usage fluctuate around 0%. The change in performance ranges from 0.6% in favor of offload and 0.6% in favor of onload. The power differences are similarly low. Since LULESH is not significantly affected by the InfiniBand card used, it interestingly contrasts with MILC.

It is important to note that the impact of decreasing CPU frequency is significant to the performance of the compute portion of the proxy-applications under study. The goal of these experiments is to examine systems known to have little process variation induced performance impact while limiting the differences between the systems to soley the networking hardware. By using the same switch for both onload and offload approaches, we have isolated other potential performance and power related impacts due to factors not of interest



(a) Onload

(b) Offload

Fig. 5. Onloaded vs. offloaded bi-directional ping-pong latency

to this study. In order to confirm the results, we conducted some MILC experiments on the exact same hardware with the network hardware swapped. These experiments confirmed that process variation between the servers used for the experiments were negligible.

2) Profiling The Applications: An interesting trend emerges when comparing these results. Offloading has a measurable performance benefit on MILC but LULESH is nearly indistinguishable from the onloaded environment. To understand the difference between these two applications, we profiled the two applications using MPIP [12]. These tests were run at 3.8GHz on four nodes with the QLogic onloaded InfiniBand cards. The pieces of information we gathered are percentage of time the application spent in MPI, the distribution of time within MPI, and the number and distribution of function calls.

Both applications spent a fair amount of time in MPI; MILC spent 15% of it's runtime in MPI, while LULESH spent 12%. However, the number of MPI calls was significantly different. MILC called MPI 4,011,216 times over its runtime averaging 1382.45 times per second. Comparatively, LULESH made substantially fewer calls, with 42,904 MPI calls over it's runtime, averaging 81.64 per second. This equates out to 2901.5 MPI calls per second for MILC and 525.5 MPI calls per second for LULESH. Table I shows the distribution of MPI calls to specific functions. The notable differences are MILC has larger percentage of wait calls and the lower percentage of Allreduce calls, compared to LULESH. Table II shows the distribution of time within MPI calls. The differences here are stark; MILC spends a reasonable amount of time in Allreduce, Isend, and Wait however, LULESH spends almost no time in Isend, mainly spending time in Allreduce and Wait. The time spent in Irecv and Isend in MILC illustrates that it is performing more significant point-to-point communication than LULESH. MILC is a memory bound code that can be sensitive to network performance, as such it is not surprising that the performance of MILC is impacted by lowering CPU frequency in an onloaded networking situation. LULESH is primarily dependent on the performance of Allreduce for good networking performance. These results indicate that the Allreduce performance is not significantly divergent in performance between the onloaded and offloaded networking approaches such that a clear winner emerges.

Call	MILC	LULESH
Allreduce	1.15%	2.22%
Irecv	24.71%	30.34%
Isend	24.71%	30.34%
Wait	49.42%	30.34%
Waitall	0.00%	6.73%
Other	0.00%	0.04%

TABLE I. Distribution of MPI Calls

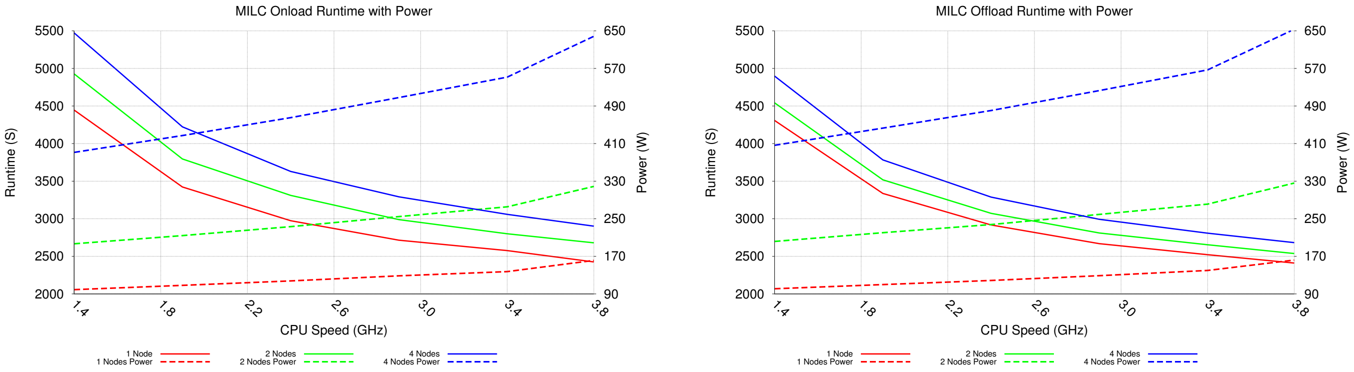
Call	MILC	LULESH
Allreduce	29.86%	42.68%
Irecv	1.71%	0 %
Isend	13.99%	0.2%
Wait	54.43%	54.02%
Other	0.01%	0.4%

TABLE II. Distribution of Time Within MPI on a Standard Run

These results indicate that the performance benefit of Offloading is seen primarily in codes that have a large number of small communication calls, rather than a few larger calls. MILC and LULESH spent similar percentages of their runtimes in MPI, but MILC relies on a large number of small point to point and collective operations and LULESH focuses on small number of large collectives that make up most of its time in MPI.

V. Discussion

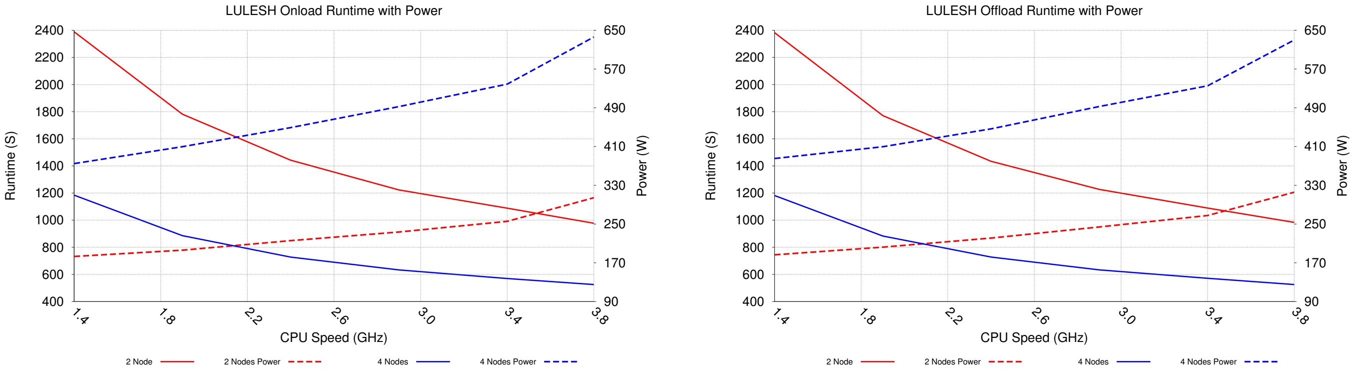
From observing the results of the microbenchmark and proxy application experiments, it can be observed that the impacts of lowered CPU frequencies on onloaded networking stack processing are inline with expectations. By using host CPUs to perform network stack processing, networking performance is susceptible to changes in CPU design, either through frequency reduction or through migration to many-core architectures. While it is expected that future multi-core server class CPUs will continue to improve their aggregate performance, single-thread performance is not expected to continue to improve in proportion to aggregate performance increases. As such, offloaded networking will provide a viable alternative for future generation systems as the networking ASIC approach can continue to provide good networking per-



(a) Onload

(b) Offload

Fig. 6. Onload vs. offloaded runs of the MILC application



(a) Onload

(b) Offload

Fig. 7. Onload vs. offloaded runs of the LULESH application

formance regardless of CPU changes as long as future CPUs can continue to provide low-latency networking requests.

Power consumption is also a concern for future capability-class systems. The experiments performed in this paper have demonstrated that networking performance for offloaded approaches can provide good network performance with major reductions in the range of 30% with less than 2% or networking performance loss. Further application studies showed that onload and offload networking approaches can diverge in performance at lowered CPU frequencies for some applications while others are less impacted by CPU frequencies changes.

VI. Related Work

The offload-versus-onload debate for high-performance interconnects has been ongoing since network interfaces moved from the memory bus to the I/O bus in early 1990's [13]. Early distributed memory on massively parallel processing machines where the network interface was on the memory bus, such as the Intel Paragon, had multiple processors per node and allowed one of these processors to be dedicated to network protocol processing. With the advent programmable network interface controllers (NICs), such as Myrinet [14] and Quadrics [15], offloading a significant portion of network protocol processing to a dedicated NIC processor became possible. For MPI-based HPC applications, these networks allowed offloading of latency-sensitive operations, such as collective communication operations. However, the benefit of offloading complex operations, such as tag matching and queue traversals required for MPI point-to-point communication operations, has continued to be debated. Proponents of onload have argued that the low performance of embedded processors in the NIC is prohibitive and that dedicating host processor cores is not only more efficient, but is also more cost effective, especially as the number of cores per node continues to grow.

Most interconnects used in large-scale HPC systems today incorporate some offload capability. IBM's Blue Gene/Q [16] and PERCS [17] networks both support offloading of MPI collective operations. Likewise, Cray's Gemini [18] and Aries [19] networks support MPI collective communication offload. With the ConnectX-2 [20] product, InfiniBand network adapters from Mellanox also began supporting MPI collective communication offload. However, these networks do not offload the more complex tag matching and queue traversal mechanisms needed to handle MPI point-to-point communication operations. These networks rely on the MPI process running on host processors for this capability. Techniques like Cray's Core Specialization [21] provide a mechanism for dedicating host processor cores to running an MPI progress thread. This technique has also been used to improve the performance of TCP/IP protocol stack processing [22].

More recently, power and energy efficiency of the interconnect has become an important consideration for large-scale data centers [23], [24] and HPC systems [25], [26], providing a new perspective on the offload-versus-onload debate. Other works have previously studied the impact that power-efficient cores have on MPI message rate [3].

VII. Conclusions and Future Work

In examining the differences between onloaded and offloaded networks for varying host CPU frequencies, it has been observed that the offloaded networking approach provides approximately equivalent or superior performance at lower frequencies. While this finding cannot be used to conclusively state that offloaded networking is key for future many-core systems networking performance, it provides key evidence to be used in future evaluation of networking approaches for future generation compute systems. The microbenchmark results clearly illustrate the potential benefits of network offloading, with power savings in the 20.5% range with only 0.5% performance loss and good performance down to 1.5% performance loss and power reductions of 30.5%. While onloading can reap greater power consumption drops, performance at the 1.4GHz level shows that onloading results in a loss of over half of the available throughput at higher CPU frequencies. This demonstrates the tradeoffs in single thread communications performance that would occur on systems with lowered CPU frequencies, and can reasonably be expected to be even lower if more simplified little-cores are used.

The conclusions reached from this study are somewhat intuitive and while the high-performance computing networking community has pre-supposed that such outcomes were likely, no study has yet addressed this issue. While the results are straightforward, they provide the foundation for discussions on the merits of onload versus offload for next generation systems. These results clearly demonstrate that single-thread performance of onloaded networking solutions can be restrictive in emerging many-core architectures. While multi-threaded approaches may alleviate some of the negative performance implications that this study exposes in single-threaded performance, the number of compute cores needed to close this gap is an open question that is currently being researched through studies into methods of providing parallelism in MPI. We can therefore conclude that at the current time, an offloaded networking approach can provide good networking performance for slower frequency processors, while an onloading approach will not be viable without further research and improvements to multi-core based network processing for high-performance computing.

In the future, we plan to expand this study by examining methods of improving multi-threading in MPI to explore if onloaded networking can provide similar performance to offloaded networks in many-core architectures. If it is possible to provide similar performance we will investigate the number of resources that need to be invested to provide offload equivalent network performance.

References

[1] J. Dinan, R. E. Grant, P. Balaji, D. Goodell, D. Miller, M. Snir, and R. Thakur, "Enabling communication concurrency through flexible message passing interface endpoints," vol. In press. Sage Publishing, 2014.

[2] A. Friedley, G. Bronevetsky, T. Hoefler, and A. Lumsdaine, "Hybrid mpi: efficient message passing for multi-core systems," in *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013, p. 18.

[3] B. W. Barrett, S. D. Hammond, R. Brightwell, and K. S. Hemmert, "The impact of hybrid-core processors on mpi message rate," in *Proceedings of the 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 67–71. [Online]. Available: http://doi.acm.org/10.1145/2488551.2488560

[4] U.S. Department of Energy's Office of Science, "The opportunities and challenges of exascale computing," 2010.

[5] "Omited for blind review."

[6] C. Aubin *et al.*, "Fermilab lattice, milc, and hpqcd collaborations," *Phys. Rev. Lett*, vol. 94, p. 011601, 2005.

[7] I. Karlin, J. Keasler, and R. Neely, "Lulesh 2.0 updates and changes," Tech. Rep. LLNL-TR-641973, August 2013.

[8] Q. O. Snell, A. Mikler, and J. L. Gustafson, "Netpipe: A network protocol independent performance evaluator," vol. 6, 1996.

[9] M. Collaboration *et al.*, "Mimd lattice computation (milc) collaboration home page," *Information available at http://physics. indiana. edu/sg/milc. html*.

[10] K. Antypas, "Nersc-6 workload analysis and benchmark selection process," *Lawrence Berkeley National Laboratory*, 2008.

[11] I. Karlin, A. Bhatele, B. L. Chamberlain, J. Cohen, Z. Devito, M. Gokhale, R. Haque, R. Hornung, J. Keasler, D. Laney *et al.*, "Lulesh programming model and performance ports overview," *Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-608824*, 2012.

[12] J. Vetter and C. Chambreau, "mpip: Lightweight, scalable mpi profiling," *URL: http://www. llnl. gov/CASC/mpiP*, 2005.

[13] K. Underwood, R. Brightwell, and S. Hemmert, "Network interfaces for high-performance computing," in *Attaining High-Performance Communication: A Vertical Approach*, A. Gavrilovska, Ed. CRC Press, 2009, pp. 149–168.

[14] N. J. Boden, D. Cohen, R. E. F. A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second local area network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, Feb. 1995.

[15] F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics network: High-performance clustering technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, January/February 2002.

[16] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The IBM Blue Gene/Q interconnection network and message unit," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 26:1–26:10. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063419

[17] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, N. Ni, and R. Rajamony, "The PERCS high-performance interconnect," in *IEEE Symposium on High-Performance Interconnects*, August 2010.

[18] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini system interconnect," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, Aug 2010, pp. 83–87.

[19] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: A scalable HPC system based on a Dragonfly network," in *Proceedings of the ACM/IEEE International Conference on High-Performance Computing, Networking, Storage, and Analysis (SC'12)*, November 2012.

[20] R. Graham, S. Poole, P. Shamis, G. Bloch, G. Bloch, H. Chapman, M. Kagan, A. Shahar, I. Rabinovitz, and G. Shainer, "Connectx-2 infiniband management queues: First investigation of the new support for network offloaded collective operations," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, May 2010, pp. 53–62.

[21] H. Pritchard, D. Roweth, D. Henseler, and P. Cassella, "Leveraging the Cray Linux Environment Core Specialization feature to realize MPI asynchronous progress on Cray XE systems," in *Proceedings of the Cray User Group Conference*, May 2012.

[22] L. Shalev, J. Satran, E. Borovik, and M. Ben-Yehuda, "Isostack: Highly efficient network processing on dedicated cores," in *Proceedings of the*

2010 USENIX conference on USENIX annual technical conference, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 5–5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855845

[23] G. Liao, X. Zhu, S. Larsen, L. Bhuyan, and R. Huggahalli, "Understanding power efficiency of TCP/IP packet processing over 10GbE," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, Aug 2010, pp. 32–39.

[24] J. Byrne, J. Chang, K. T. Lim, L. Ramirez, and P. Ranganathan, "Power-efficient networking for balanced system designs: Early experiences with PCIe," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, ser. HotPower '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:5. [Online]. Available: http://doi.acm.org/10.1145/2039252.2039255

[25] T. Hoefler, "Software and hardware techniques for power-efficient HPC networking," *Computing in Science Engineering*, vol. 12, no. 6, pp. 30–37, Nov 2010.

[26] E. Totoni, N. Jain, and L. Kale, "Toward runtime power management of exascale networks by on/off control of links," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 915–922.