Analyzing the Energy and Power Consumption of Remote Memory Accesses in the OpenSHMEM Model

Siddhartha Jana[1], Oscar Hernandez[2], Stephen Poole[2], Chung-Hsing Hsu[2], and Barbara Chapman[1]

[1] HPCTools, Computer Science department,
University of Houston,
Houston, Texas
sidjana,chapman@cs.uh.edu
[2] Computer Science and Mathematics Division
Oak Ridge National Laboratory,
Oak Ridge, Tennessee
oscar,spoole,hsuc@ornl.gov

Abstract. PGAS models like OpenSHMEM provide interfaces to explicitly initiate one-sided remote memory accesses among processes. In addition, the model also provides synchronizing barriers to ensure a consistent view of the distributed memory at different phases of an application. The incorrect use of such interfaces affects the scalability achievable while using a parallel programming model. This study aims at understanding the effects of these constructs on the energy and power consumption behavior of OpenSHMEM applications. Our experiments show that cost incurred in terms of the total energy and power consumed depends on multiple factors across the software and hardware stack. We conclude that there is a significant impact on the power consumed by the CPU and DRAM due to multiple factors including the design of the data transfer patterns within an application, the design of the communication protocols within a middleware, the architectural constraints laid by the interconnect solutions, and also the levels of memory hierarchy within a compute node. This work motivates treating energy and power consumption as important factors while designing compute solutions for current and future distributed systems.

1 Introduction

Recent studies on the challenges facing the Exascale era express a need for understanding the energy profile of applications that depend on inter-process communication on large-scale systems. The amount of energy consumed due to data movement poses a serious threat to the usability of distributed memory models on future systems. One-sided communication in PGAS models are analogous to memory accesses in shared-memory models. However, its impact on the performance and power consumption is different.

Shared memory models are characterized by implicit data transfers that are bounded by the distance between the CPU and the different levels of the memory hierarchy. Such data transfers include intra-node cache and memory accesses that consume very low energy, typically of the order of 800-1000 pico Joules [7]. In contrast, inter-process communication patterns in PGAS models are initiated by the programmer and bounded by a number of factors internal and external to a single compute node. In this paper, we present our study of the factors affecting the energy and power consumption behavior of parallel programming models. These can be categorized as either internal or external based on the associated layers of the hardware and the software stack. Benedict [6] and Hoefler [12] list some of the factors that have the potential to affect the energy consumption of interconnect solutions. They include the router/switch organization, flow control, congestion control, routing and deadlock handling, network topology, load balancing, reliability, and QoS support.

The scope of this paper is to discuss the energy consumption by the CPU and the memory hierarchy while servicing remote data transfers and synchronization constructs provided by the OpenSHMEM model.

PGAS implementations like OpenSHMEM stand out with respect to their memory consistency model. To maintain a consistent view of the progress of execution and the globally-shared memory among multiple processes (or *processing elements*), OpenSHMEM provides synchronizing constructs like *shmem_barrier_all()*, *shmem_fence()*, and *shmem_quiet()*. The impact of such barriers on the performance and scalability of distributed applications is well known [17]. In Section 3, we discuss our findings on the factors affecting the power consumption of applications that use such barriers.

The OpenSHMEM memory model permits RDMA operations. Our studies indicate that during the progress of such operations, there is a significant impact on the power consumed by the CPU and DRAM due to multiple factors including the design of the data transfer patterns within an application, the design of the communication protocols within a middleware, the architectural constraints laid by the interconnect solutions, and also the levels of memory hierarchy within a compute node. We present a study of the parameters that affect the power consumption behavior of such interfaces in Section 4.

Our empirical study was carried out at the granularity of various OpenSHMEM constructs. Because of the fine level of granularity, it was essential to reduce the impact on the power readings by external noise like the host Operating system and background processes. In Section 2, we describe our experimental setup for collecting the energy and power readings under such conditions.

2 Notes on Experimental Setup

The details of the test environment used to obtain the empirical results in this paper are listed in Table 1.

Table 1: Test machine and environment details

Processor	Intel Xeon CPU E5-2670
Microarchitecture	Intel's Sandy Bridge
Maximum Thermal Design Power (TDP)	115 Watts
Hyperthreading support	Disabled
Sockets	2
Cores/socket	8
L1 cache size (per core)	32KB
L2 cache size (per core)	256KB
L3 cache size (shared - 1/socket)	20MB
Infiniband card	Mellanox MT26428 [ConnectX VPI PCIe 2.0 5GT/s]
Infiniband switch	InfiniScale IV 36-Port QSFP 40 Gb/s, MTS 3600
Compiler	gcc version 4.4.6
Compiler flags used	-O3
OpenSHMEM version	Mellanox OpenSHMEM ver. 2.2-23513

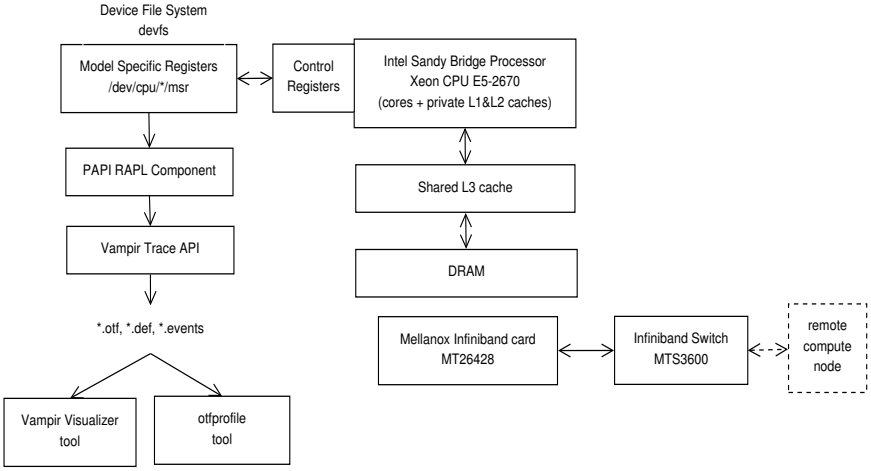


Fig. 1: Experimental Setup incorporating Intel's RAPL interface for fine-grained power monitoring

2.1 Setup for Monitoring Energy and Power consumption

In order to monitor energy consumption by different components of a compute node (cores, socket, memory), we used Intel's Running Average Power Limiting (RAPL) interface [1]. Fig. 1 illustrates our experimental setup which incorporates this interface by monitoring the thermal and power management values of the model-specific registers (MSRs) exposed by the Intel Sandy Bridge processor, E5-2670. Readings provided by these interfaces have certain shortcomings due to its model-based approach for predicting results [11]. However, verifications by David et al. [8], Hackenberg et al. [11], and Dongarra et al. [9] provide empirical evidence of a high correlation between the energy consumption readings provided by the RAPL interface and direct power measurements. Statistical evidence [8] indicate that the estimated power as reported by RAPL is within 1% of direct power readings[3] with a standard deviation of 1.1% [8]. On our system the time window for the RAPL interface was 0.046 seconds. This proved sufficient for studying the behavior of the energy and power consumption patterns by different OpenSHMEM interfaces.

In order to read the RAPL counters in MSRs from the device file system (/dev/cpu/*/msr on *devfs*), we used the RAPL component provided by PAPI v5.1 [18]. In addition, we used Vampir Trace [14] for fine-grained instrumentation of our synthetic microbenchmarks.

2.2 Reducing Noise in Readings due to the OS and Background Processes

To reduce OS noise and avoid other processes from being scheduled on the monitored socket, we used Linux *CPU shielding* [13]. This ensured that all unrelated processes/threads (including most OS service threads) were scheduled on the extra unmonitored socket on the compute node (refer to Table 1 for the machine details). We verified this approach by observing a steady power consumption of 3.786 Watts when none of our experimental processes were scheduled on the monitored socket.

3 Effects of Synchronization Barriers

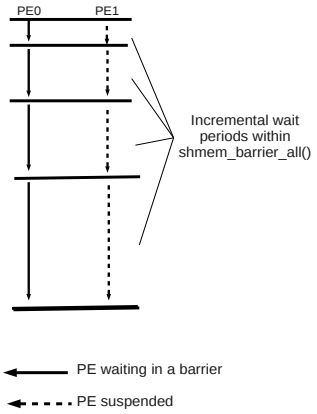
For applications in which the work distribution among multiple processes is non-uniform, using synchronizing constructs[4] result in a subset of processes waiting for varying intervals of time without making any progress. Thus, applications become bounded by the speed of the slowest process, thereby significantly impacting both its performance and scalability [17]. This impact worsens with the rise in the number of processes executing the application. In this section, we

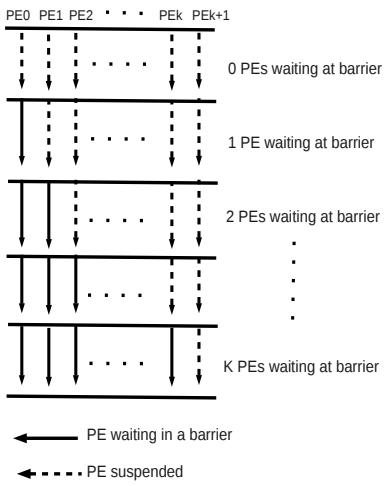
[3] 'Direct power reading' implies power measurement obtained by AC instrumentation that use power meters with high accuracy and calibration.

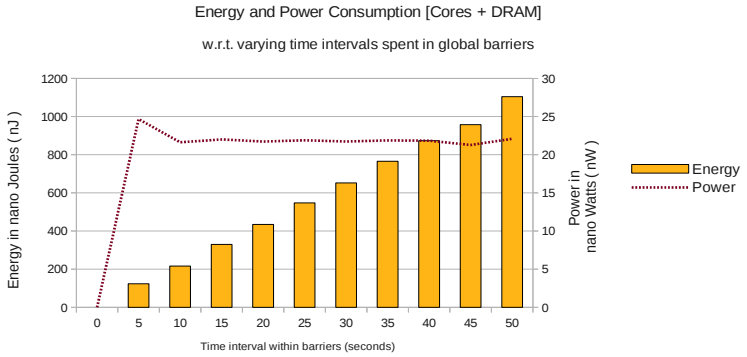
[4] In the rest of the text, we use the words 'synchronizing construct' and 'barriers' interchangeably.

Table 2: Microbenchmark and line charts for studying the impact of barrier on energy and power cost
(i) varying wait periods within a barrier (ii) varying number of processes participating in a barrier

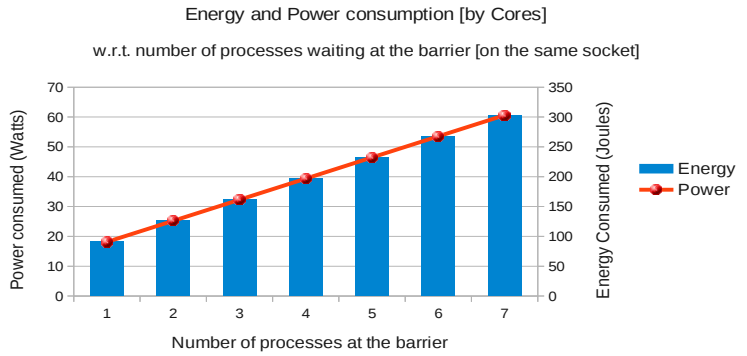
Line charts	Code snippets

	`for(sleep_cnt=0; sleep_cnt>=MAX_SLEEP; sleep_cnt +=5)`
	`{`
	`shmem_barrier_all();`
	`if (me == 0)`
	`sleep(sleep_cnt);`
	`else`
	`sleep(MAX_SLEEP);`
	`// START monitoring`
	`shmem_barrier_all();`
	`// STOP monitoring`
	`}`

	`for(cnt=num_pes()-1; cnt>=0; cnt--)`
	`{`
	`shmem_barrier_all();`
	`if (me <= cnt)`
	`sleep(CONST_SLEEP);`
	`// START monitoring`
	`shmem_barrier_all();`
	`// STOP monitoring`
	`}`



(a) Impact of wait period within a barrier



(b) Impact of number of processes participating in a barrier

Fig. 2: Empirical results illustrating the impact of barriers on the energy and power cost of the system

underscore the notion that such a lack of progress by processes lead to significant waste of computational resources. This in turn implies a rise in the energy consumption of applications. We study this impact on the energy cost in terms of two factors - the cost incurred by processes waiting for different time periods within a barrier, and the cost incurred by the entire system with a rise in the number of processes participating in a barrier.

PGAS implementations like OpenSHMEM decouple communication and synchronization operations [10]. A process may progress in its execution of code segments while being oblivious to communication operations initiated by other processes. In other words, processes are permitted to have an inconsistent view of the globally shared memory during a phase of an application. To ensure sequential consistency and an ordering of remote data transfer operations, OpenSHMEM applications may use synchronizing constructs like *shmem_barrier_all()*, *shmem_fence()*,

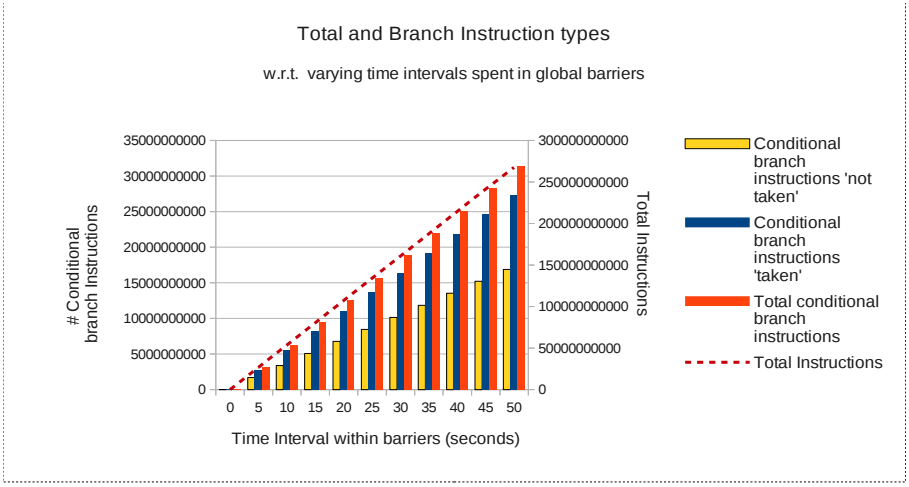


Fig. 3: Comparing the types of instructions executed by the CPU while waiting at a barrier. The count includes (i) Total number of instructions (ii) Number of conditional branch instructions (iii) Number of conditional branch instructions that are 'taken' (iv) The number of conditional branch instructions that are 'not taken'

and *shmem_quiet()*. We discuss the impact of using the global barrier - *shmem_barrier_all()* below:

- *Energy and power consumption with respect to time spent within a barrier*:
  The line chart and the code snippet of the microbenchmark used to verify this is presented in the first row of Table 2.
  Fig.2a illustrates that a linear growth in the time spent by a process within a barrier leads to a linear rise in total energy consumed by the system (cores and the DRAM)[5]. In addition, we also observe that the power consumption or the rate of change in energy, is independent of the time spent by a process waiting at a barriers. We discuss this observation in Section 3.1
- *Energy and power consumption with respect to the number of processes waiting at a barrier*:
  The line chart and the code snippet of the microbenchmark used to verify this is presented in the second row of Table 2
  The results depicted in Fig. 2b verify the claim that an increase in the number of processes waiting at a barrier leads to a linear rise in the energy consumed over the entire system which, in turn implies a linear rise in the average power consumption.

[5] For our experiments, the linear relationship between the energy (E) consumed and the time (T) spent within a barrier was: E = (33.1446*T)-1.88467. As expected, the model was characterized with a high Coefficient of determination (r^2=0.999027)

3.1 A Note on Implementation of Barriers

Common implementations of a barrier incorporate the use of shared semaphores which are subjected to repeated atomic polling by each process. The purpose of this polling is to keep track of the state of the semaphore objects. These are typically *globally shared* so that they remain accessible by other processes[6]. The polling is always *atomic* in nature to ensure that only one process can test or set it at any point in time. Furthermore, this polling is typically performed directly over the copy of the semaphore object within the remotely accessible memory, thus avoiding accesses to stale cached versions. This in turn increases the pressure on the memory. Additionally, the polling is *repeatedly iterated*, to ensure that there is no significant delay between the time each process signals entering the barrier and the time this event is detected.

Such *software-based* implementations of barriers result in the CPU repeatedly executing the same set of instructions without making any progress in the application. It is only when a semaphore signals the end of the barrier, that the CPU executes a code fragment that prepares the process to exit the barrier region. In accordance with this design, Fig. 3 depicts the change in the energy and power consumption pattern with respect to the types of instructions executed by the CPU. The waste in CPU cycles can be observed by the linear rise in the difference between the number of conditional-branch instructions that are 'taken' and 'not taken'.

Also, the high correlation between the *total* number of instructions executed and the total number of *conditional-branch* instructions hint at the execution of the same set of instructions, irrespective of the time spent in the barrier. This *homogeneity* in the instruction types result in a constant power consumption by the system (Fig. 2a)

4 Effects of Remote Data Transfers

This section discusses the impact of the use of explicit data transfer routines on the energy cost of OpenSHMEM applications. While using these routines, a programmer may decide to transfer the program data in multiple fragments based on the design of an application. While this practice makes it easier to align the semantics of an algorithm to an implementing program, our studies indicate that such practices come at a significant cost.

Thus, we identify two application characteristics to analyze the communication patterns in OpenSHMEM programs:

- *Total size of the data to be transferred*
  This factor is governed by the problem size of the application and granularity of parallelism chosen[7].

[6] If RDMA is supported by the interconnect, the overhead of the management of semaphores reduces when they are remotely accessible

[7] The granularity of parallelism is typically determined by the number of processes participating in the data/task distribution.

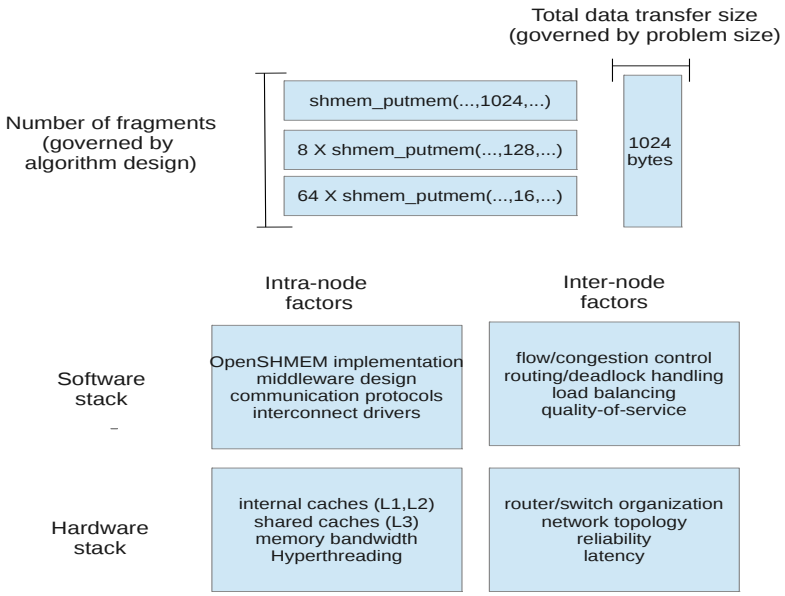


Fig. 4: Top: Parameters that define communication patterns from an OpenSHMEM programmer's point of view. Bottom: Underlying factors within the software and hardware stack that impact the power and energy cost of interfaces for remote data transfer

– *Number of explicit calls (or fragments) used to transfer the data*
  This factor is dependent on the nature of the design of the application by the programmer.

However, it must be noted that the actual progress of the data movement depends on a number of factors related to the design and the capabilities of the underlying software and hardware stack. Fig. 4 categorizes these factors depending on whether they impact the energy and power profiles of internal system-components like the CPU and DRAM i.e. the *intra-node* factors, or external components like the interconnect solutions i.e. *inter-node* factors.

The study of the effects of inter-node factors on the energy profile of remote data transfers is outside the scope of this paper. Nevertheless, in order to account for their impact, we abstract their effects in terms of the net achievable bandwidth. Fig. 5 illustrates this constraint with respect to the two communication-based parameters discussed above. We observe that for any given data transfer size, maximum bandwidth is achievable with minimum amount of fragmentation.

The impact of the intra-node factors on the energy and power cost incurred in Sections 4.1 and 4.2, respectively.

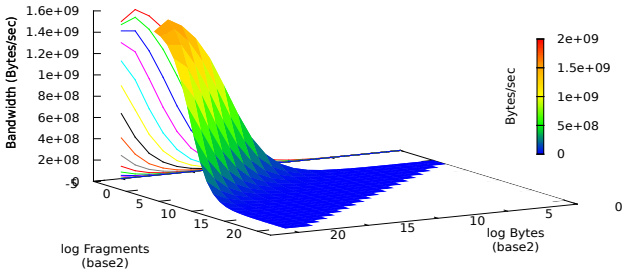


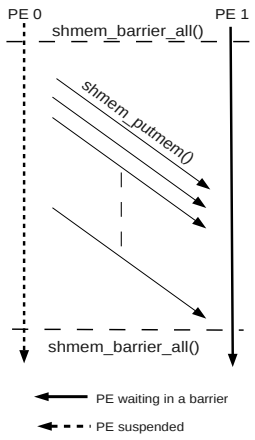
Fig. 5: The impact of the interconnect solution can be summarized by the achievable bandwidth with respect to:
(i) size of the total data to be transferred
(ii) number of fragments into which the transfer is divided into.

4.1 Energy Consumption Observations

Fig. 6 illustrates the energy consumption by the CPU and the DRAM with respect to the different message sizes of data transferred (in bytes along X-axis) and the number of fragments used to transfer the total data (along Y-axis). The noteworthy observations are:

- Energy consumed holds a correlation to the number of instructions executed. Since an increase in the number of data transfers initiated implies a rise in the number of instructions executed, the energy consumption increases with rise in fragmentation.
- For large bulk transfers with a fixed message size, the energy consumed remains independent of the *initial* rise in fragmentation.
- Using a constant number of fragments, the energy consumed in servicing the transfer of small to medium sized messages (2 to 65536 bytes) is independent of the total size of the data transferred. This behavior can also be observed in terms of the spectrum of the achievable bandwidth shown in Fig.5. This behavior can be explained by the fact that for such small-sized messages, the cost in managing the data buffers for remote transfers overshadows the actual movement of the data. This cost is independent of the message size and hence leads to a steady bandwidth and energy consumption.
- For large bulk transfers (>65536 bytes), the energy consumed increases with the size of the data to be transferred. This can be attributed to cost incurred in handling data buffers. For large messages, this becomes dependent on the actual size of data that is being transferred.

Table 3: Microbenchmark for evaluating energy and power consumption by varying the total size of data payload and the number of fragments

Line chart	Code snippet
	see below


```
me = my_pe();
for(j=1 ; j<=MAX_WRK_SIZE ; j*=2)
{
  for(frag_cnt=MIN_MSG_NUM;
      frag_cnt<=j; frag_cnt*=2)
  {
      bytes_per_frag = j / frag_cnt;
      shmem_barrier_all();
      // START monitoring
      if(me == 0)
        for (it=0; it<frag_cnt;
             it++)
          shmem_putmem(   ....,
              bytes_per_frag, 1);
      shmem_barrier_all();
      // STOP monitoring
  }
}
```

4.2 Power Consumption Observations

Fig.s 7 depicts the power consumption by the CPU cores and the DRAM for different message sizes and number of fragments.

- For small data transfer sizes, the power consumed by the CPU (16 Watts) and the DRAM (7 Watts) is low.
- The power consumed by the CPU during transfer of large bulk data payloads (16.2 Watts) is marginally more (1.25%) than that consumed during small data transfers.
- The power consumed by the DRAM during transfer of large bulk data payloads (9 Watts) is significantly more (22%) than that consumed during small data transfers.
- With very low fragmentation, the CPU consumes more power than with fragmented data payload.
- As the message size is increased (along x-axis), the transition of the change in the power consumption behavior by the CPU appears to hold a correlation to the sizes of the intermediate levels of the cache hierarchy. The transition levels correspond to the sizes of the L1 and L2 caches - 32KB and 256KB

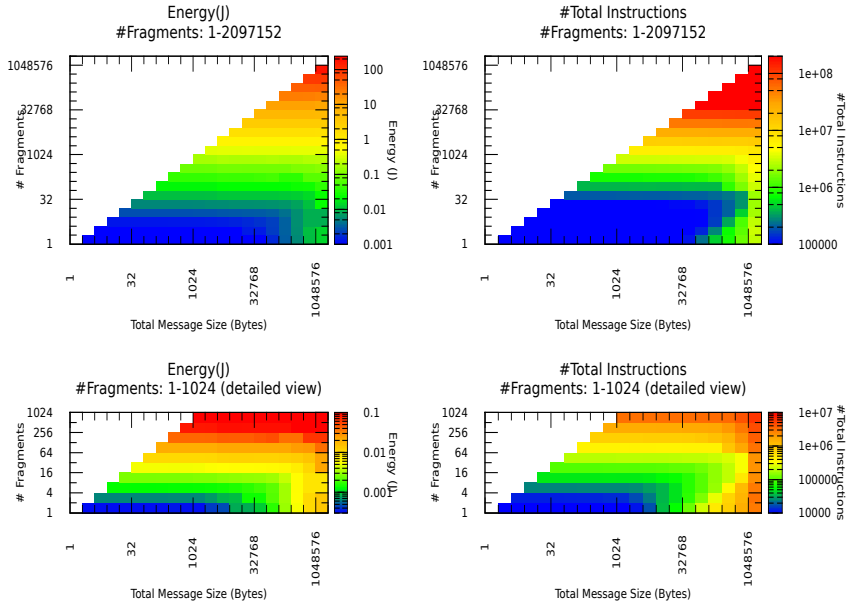


Fig. 6: Relationship between energy consumption by cores(left) and the total number of instructions executed(right). Top: Results for cases where: $Fragments \in [1, 2097152]$. Bottom: Results for cases where: $Fragments \in [1, 1024]$

respectively. Since the caches were flushed after every set of readings, one can speculate that every cache miss in L1 and L2 adds on to the memory pressure on the shared L3 cache thereby resulting in a proportional rise in cache misses. This effect can be observed in Fig. 7(III), which illustrates the number of L3 cache misses.

– From Fig. 7, the average power consumption by the system (CPU+DRAM) while servicing large bulk message sizes (28 Watts) is 21.73% higher than that consumed by small message sizes (23 Watts).

5 Related Work

There has been a great deal of research directed towards measuring and managing the energy and power consumption of applications. Proposals like Thrifty [2] have been put forth to direct large-scale research towards redesigning the complete computing stack. The goal of such efforts is directed towards building power-aware Exascale platforms.

Some of the model-based techniques provided by chip manufacturers to dynamically monitor and manage the power or energy consumption include: Intel's RAPL [1], AMD's APM module [3], NVIDIA's NVML [4].

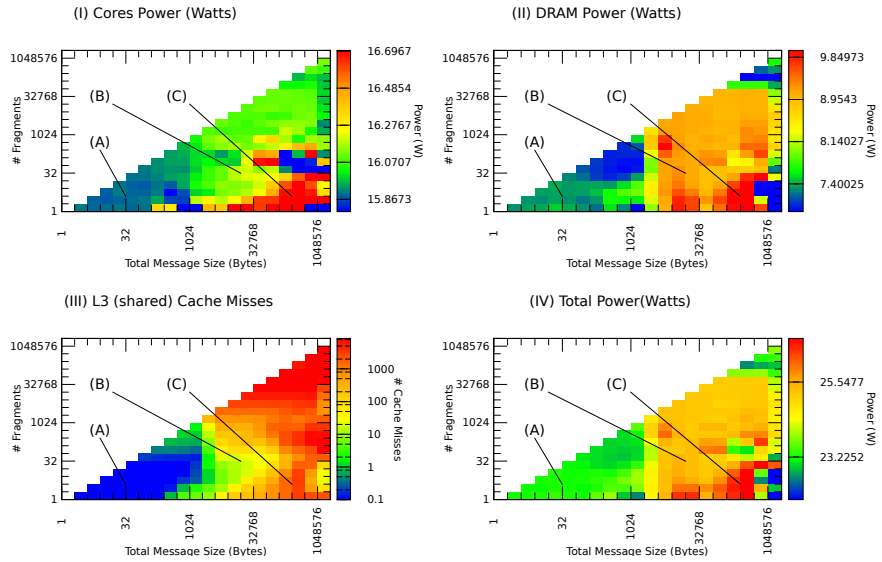


Fig. 7: (I,II,IV)Power consumed by CPU, DRAM, total system (III) Total L3 cache misses. The various distinct levels of power are represented as:
(A)Small payload sized(up to 2KB) transfers lead to less power consumption by the cores and DRAM;
(B)Medium to large message sizes(4K and beyond) imply accesses of large memory regions and this impacts power consumption;
(C)Large payload sizes with minimum fragmentation leads to higher power consumption by the cores. The underlying NIC is generally responsible for chunking such large transfers, the effect on which is not accounted for by the cores.

Hoefler [12] mentions discussions by the IEEE standard on energy efficient Ethernet specifications including - dynamic link-speed reduction, receiver modification, network routing, and deep sleep states. However, initial research indicates latencies and network jitter with these techniques.

Past efforts towards understanding and managing the power consumption trends of applications have been significant. One of the static based approaches for managing power consumption by processes is for the compiler to evaluate a program and determine sections within the code where the energy consumption profile changes. This knowledge in the form of *power management hints* can then be conveyed to the runtime to adjust the voltage/frequency scaling of applications [5]. Korthikanti and Agha [15] study the power consumption behavior of shared memory architectures while handling applications with different problem sizes. Li et al. [16] use DCT and DVFS techniques to study the opportunities of reducing power consumption of hybrid MPI-OpenMP applications. The focus of our work has been to perform a fine-grained study of OpenSHMEM communication interfaces which are responsible for remote memory accesses.

6 Conclusion

In this paper we presented our study of the energy and power consumption behavior of a system while participating in synchronizing global barriers and remote data transfers.

We observed that the energy and power cost is dependent on the time spent within barriers and the number of processes participating in the barriers.

Additionally, our study indicates that the energy and power cost incurred by a system while servicing remote data transfers are dependent on a number of factors characterizing the underlying the hardware and software stack. These include the sizes of the memory hierarchy, design of the communication protocols, and the capabilities of the interconnect solutions.

The impact of these factors depend on the size of the total data to be transferred within a communication phase of an application. In addition, the number of data transfers initiated to transfer this load also impact the energy and power consumption behavior of OpenSHMEM-like PGAS applications.

The results put forth in this paper motivate the need for taking energy and power costs into account while designing efficient PGAS libraries for large-scale systems.

7 Acknowledgments

This work is supported by the U.S. Department of Defense and used resources of the Extreme Scale Systems Center located at the Oak Ridge National Laboratory. Experiments included resources of the Oak Ridge Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Vampir trace is developed at the Center for Information Services and HPC of Technische Universität Dresden in Germany. PAPI is an open-source research project of the Innovative Research Laboratory of the University of Tennessee, Knoxville in the U.S.

Thanks are due to Joseph Schuchart and the Vampir-support team at Technische Universität Dresden, for offering support with Vampir. Special thanks are also due to Pavel Shamis from ORNL for his input on the usage of Infiniband verbs.

References

1. Intel 64 and ia-32 architectures software developers manual volume 3b: System programming guide, part 2
2. Thrifty: An exascale architecture for energy-proportional computing `http://science.energy.gov/~/media/ascr/pdf/research/cs/aa/A_oph_uiuc_thrifty_110215.pdf`
3. Linux tuning guide, amd opteron 6200 series processors (April 2012)
4. Nvml api reference manual, ver.5.319.43 (August 2013)

5. Aboughazaleh, N., Childers, B., Melhem, R., Craven, M.: Collaborative compiler-os power management for time-sensitive applications. Tech. rep., Department of Computer Science, University of Pittsburgh, Department of Computer Science, University of Pittsburgh (2002)
6. Benedict, S.: Review: Energy-aware performance analysis methodologies for hpc architectures-an exploratory study. J. Netw. Comput. Appl. 35(6), 1709–1719 (Nov 2012), http://dx.doi.org/10.1016/j.jnca.2012.08.003
7. Choi, J.W., Bedard, D., Fowler, R., Vuduc, R.: A theoretical framework for algorithm-architecture co-design. In: Proc. IEEE Int'l. Parallel and Distributed Processing Symp. (IPDPS). Boston, MA, USA (May 2013)
8. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: Rapl: Memory power estimation and capping. In: Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on. pp. 189–194 (2010)
9. Dongarra, J., Ltaief, H., Luszczek, P., Weaver, V.M.: Energy Footprint of Advanced Dense Numerical Linear Algebra Using Tile Algorithms on Multicore Architectures. 2012 Second International Conference on Cloud and Green Computing pp. 274–281 (2012), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6382829
10. group, H.P.C.T., at UH, E.S.S.C.a.O.: Openshmem application programming interface, version 1.0. Tech. rep., University of Houston (UH), Oak Ridge National Laboratory (ORNL) (2012), www.openshmem.org
11. Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M., Nagel, W.: Power measurement techniques on standard compute nodes: A quantitative comparison. In: Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on. pp. 194–204 (2013)
12. Hoefler, T.: Software and hardware techniques for power-efficient hpc networking. Computing in Science Engineering 12(6), 30–37 (2010)
13. Kerrisk, M.: Linux programmer's manual (2012), http://man7.org/linux/man-pages/man7/cpuset.7.html
14. Knpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Mller, M., Nagel, W.: The vampir performance analysis tool-set. In: Resch, M., Keller, R., Himmler, V., Krammer, B., Schulz, A. (eds.) Tools for High Performance Computing, pp. 139–155. Springer Berlin Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-68564-7_9
15. Korthikanti, V.A., Agha, G.: Towards optimizing energy costs of algorithms for shared memory architectures. Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures - SPAA '10 p. 157 (2010), http://portal.acm.org/citation.cfm?doid=1810479.1810510
16. Li, D., de Supinski, B.R., Schulz, M., Cameron, K., Nikolopoulos, D.S.: Hybrid MPI/OpenMP power-aware computing. 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS) pp. 1–12 (2010), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5470463
17. Markatos, E., Crovella, M., Das, P., Dubnicki, C., LeBlanc, T.: The effects of multiprogramming on barrier synchronization. In: Parallel and Distributed Processing, 1991. Proceedings of the Third IEEE Symposium on. pp. 662–669 (1991)
18. Mucci, P.J., Browne, S., Deane, C., Ho, G.: Papi: A portable interface to hardware performance counters. In: In Proceedings of the Department of Defense HPCMP Users Group Conference. pp. 7–10 (1999)