Analysis of Energy and Performance of Code Transformations for PGAS-based Data Access Patterns

Siddhartha Jana
University of Houston,
Texas
sidjana@cs.uh.edu

Joseph Schuchart
Technische Universität
Dresden
joseph.schuchart@tu-dresden.de

Barbara Chapman
University of Houston,
Texas
chapman@cs.uh.edu

ABSTRACT

One of the factors associated with the usability of distributed programming models in exascale machines is the energy and power cost associated with data movement across large-scale systems. PGAS models provide the user with explicit interfaces to transfer data to remote processes. From an application programmer's perspective, a number of factors impact the semantics and performance of communication-intensive kernels. Many of these directly impact the behavior of the underlying PGAS library. Examples include: the completion semantics of the data transfer constructs, the number of explicit library-calls used to perform such transfers, the size of data transferred during each call, the contiguity of the data buffers within the memory, the count and the management overhead associated with handling in-progress transfers, and the registration status of the buffers with the NIC.

In this paper, we discuss a number of transformations on RDMA-based data access patterns that have the potential of significantly reducing the energy dissipation during the execution of application kernels. Such transformations are mostly accompanied by a significant reduction in the transfer latencies. We present an empirical study of this impact by analyzing the energy consumption of two major sub-components of the system - the CPU and the memory. Since performance is a major criteria for PGAS programming models, we use the energy-delay product as a metric to justify the feasibility of these transformations.

We hope that this work motivates the incorporation of energy-based metrics within fine tuning PGAS implementations.

Categories and Subject Descriptors

D.1.3 [**Software**]: PROGRAMMING TECHNIQUES—*Concurrent Programming – Distributed Programming*

General Terms

Performance

Keywords

Energy efficiency, DVFS, Remote data transfers, PGAS Code Transformations, Data Access Patterns

1. INTRODUCTION

One of the primary challenges on the pathway to Exascale Computing is the 20 MW power consumption envelope established by the U.S. Department of Energy's Exascale Initiative Steering Committee [18]. The direct outcome of this has been a rising concern about the energy and power consumption of large-scale applications that rely on various communication libraries for efficient data movement in distributed systems. From our past study[10], we have learned that communication semantics provided by PGAS models have the potential of exploiting a number of energy saving opportunities while servicing data transfers across distributed systems. This is aided by their flexibility of decoupling synchronization costs from the actual transfer of the data-payload.

This work is an extension of our previous experience in studying the impact of one-sided communication in PGAS models[10]. We had learned that managing small-sized data transfers on RDMA-capable networks is more energy efficient than handling large bulk transfers. Most research efforts study the impact of data movement with respect to the size of the transfer. However, we believe that besides the size of the data payload, the data transfer patterns within applications play a major role in determining the impact. In this paper, we discuss a number of such patterns and perform an empirical analysis of the maximum possible savings that may be obtained while choosing one access pattern over the other. These motivate the need for static or dynamic transformations of communication techniques applicable across various levels of the software stack. We evaluate some well known techniques like aggregating contents of source buffers of multiple remote write operations, using non-blocking data transfer semantics, using pinned-down buffers, and managing the size of data payload packed within each transfer. We present empirical results that indicate that the savings (in terms of performance and energy) obtained through such techniques varies significantly and there is plenty of opportunity for system programmers to tune energy-efficient PGAS implementations.

To summarize, the main contributions of this paper are:

- We discuss a number of factors characterizing data access patterns that have the potential of impacting the energy signatures of PGAS applications

- We discuss multiple transformations on data access patterns that have the potential of significantly impacting energy savings without negatively affecting the performance.
- We present empirical evidence of the feasibility of such transformations by analyzing the impact of these transformations in terms of the reduction in CPU energy consumption, DRAM energy consumption, communication latency, and the energy-delay product.

We layout our study as follows. Section 2 provides a discussion on past related work. In Section 3, we discuss the various characteristics within PGAS communication kernels that have an impact on the energy and latency cost of applications. In Section 4, we define a data access pattern in terms of some basic elements and list a small subset of such patterns. This discussion is followed by a some examples of transformations that have the potential of energy savings. Our experimental setup for the analysis of these transformations is described in Section 5. Following this, we back the claim of potential energy savings by present empirical results to back the claim of significance numbers are presented in Section 6.

We hope that this work motivates the incorporation "energy-based" metrics while fine tuning PGAS implementations.

2. RELATED WORK

Proposals like Thrifty[19] 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.

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 [1]. Korthikanti and Agha [12] study the power consumption behavior of shared memory architectures while handling applications with different problem sizes. Li et al. [14] 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.

There has been a great deal of research in managing the energy consumption of applications. Most of these efforts target energy-based optimizations for applications running in a shared memory environment. The maximum impact on the energy savings in such platforms are goverened by the avoidance of penalty due to cache misses and memory-intensive operations. For example, Rahman et al.[17] propose reducing power consumption in scientific applications by decreasing the number of active threads and fine-tuning cache blocking and loop unrolling factors to achieve efficient execution. Research efforts show that power bottlenecks are common in case of "disagreements" between the application activity and the system power consumption and quite often the source of inefficiency can be tracked down to the use of power-hungry busy-waits[2, 3, 5].

Barreda et al.[4] discuss work on a Framework for aposteriori detection of power-sinks in the form of discrepancies between the application activity and the CPU C-states. Choi et al.[7] explore opportunities of using DVFS in case of memory intensive phases of applications. Their approach relies on prediction of this intensity by dynamically measuring the ratio of off-chip versus on-chip accesses.

The work closest to our focus are those by Vishnu et al[21], Kandemir et al.[11] and Venkatesh et al.[20]. Kandemir et al.[11] discuss static based techniques like traditional data flow analysis and polyhedral algebra to detect redundant communications and unwanted synchronizations in HPF-like languages. Vishnu et al.[21] exploit voltage frequency scaling and interrupt-based methods to achieve energy savings during remote memory operations. They implement this technique in ARMCI[16]. The energy savings discussed in this work only target individual data transfer operations. Venkatesh et al.[20] discuss techniques of energy measurement of MPI-based data transfers using Intel's RAPL scheme. Energy readings of point-to-point and collective operations are discussed. However, these efforts do not take into account the impact of multiple factors across the hardware and software stack. As we discuss in this paper, the cost of an independent data transfer constuct is dependent on its semantics and the data access pattern it participates in. The following sections discuss a number of similar factors and present analysis of empirical results that are significantly impacted by them.

3. DESIGN FACTORS IMPACTING ENERGY PROFILES OF PGAS COMMUNICATION KERNELS

This section describes some application-level design factors that have the potential of impacting the energy signatures of communication-intensive kernels. While these factors are controllable at the user-level, their use directly impacts the behavior of the underlying communication library.

At a higher level, we categorize these on the basis of (I) Properties of the communication kernel (II) Properties of individual data transfers

3.1 Properties of the Communication Kernel

The total size of the payload being transferred. From our past experience[10] and other research efforts[20, 21], we have learned that the total size of data participating in RDMA-based operations within a communication-kernel, has a direct impact on the energy consumption of an application[1]. Costs associated with handling non-contiguous data

[1] It must be noted that the significance of the impact of such a metric depends on the actual ratio of the number

buffers and multiple initiated remote transfers are bounded by this factor. While the payload size associated with data movement is important, the overhead associated with the software stack that services the transfer of the payload is equally significant. Therefore, one of the crucial factors that needs to be considered while evaluating energy and performance costs is the number of user buffers over which the payload is distributed. This is described next.

The number of explicitly initiated data transfers. Due to the importance of this metric, we compare all performance- and energy-based parameters with respect to this count for a given data-payload size. For the rest of the text, we refer to this metric as “fragments”. This metric takes multiple forms across the software stack:

At the application level: Given a fixed payload size, the latency and energy cost may increase with the number of explicitly initiated remote read/write operations. The exact count of such operations coupled with the actual number of bytes transferred per operation is largely dependent on the design of the data structures and access patterns used by the application programmer.

At the data transfer layer: The impact of this metric can also be related to the completion semantics of RDMA transfers. For example, in case of non-blocking remote write operations, this metric can also be translated to the number of outstanding in-progress PUTs as dictated by the application design. In such cases, the energy and latency costs are impacted not only by the cost for servicing the actual transfers, but also that for managing multiple communication handlers.

At the raw bytes transfer layer: The maximum number of bytes that can be transferred at a time through the physical layer is dictated by the constraints imposed by the NIC. Thus it is common for large bulk transfers to be divided into smaller chunks at this layer as well.

3.2 Properties of the Individual Data Transfers

The data-transfer completion semantics. Most modern interconnects support non-blocking transfers of data between the local and remote memories. The latency due to such remote transfers may therefore be overlapped by the available computation. This ensures efficient use of CPU cycles that would otherwise be wasted while polling for the completion status of otherwise blocking transfers. However, the use of non-blocking transfers comes with the price of: (a) having to manage multiple communication handlers, and (b) the count of the number of in-progress transfers. This management of large number of such transfers might lead to an increase in the participation of the CPU, thereby increasing the energy consumption.

The contiguity of the data-buffers in memory. While handling small to medium sized transfers, an application developer or the PGAS implementation itself may exploit the peak bandwidth of the underlying interconnect by merging multiple non-contiguous source buffers into a single contiguous chunk before sending the contents across the network. This tactic is well-established among PGAS implementations which support strided, indexed, or vectorized transfers[16]. However, one has to be wary of the latency and the energy cost associated with such mechanisms due to (a) the impact of local *memcpy()*s which are CPU and DRAM intensive, and (b) the maximum achievable bandwidth of the underlying interconnect. The benefits therefore depend on the extent of hardware support and the amount of computation available for overlapping the latency associated with bulk transfers.

The registration status of the source buffers with an RDMA-capable NIC. PGAS implementations built on top of OS-bypass mechanisms require the virtual-to-physical address mapping to be pinned down. This pinned region is registered with the NIC to enable RDMA-based accesses. If the application programmer uses a source buffer that is not pinned to the memory, a PGAS implementation typically performs a local copy of the contents of the buffer to a portion of a pre-registered memory[2]. As shown in further sections, such local memory copies are CPU and DRAM intensive and their cost is proportional to the size of the copied contents. Therefore, the status of registration of the user buffer has the potential of directly impacting the energy profiles of kernels.

4. CODE TRANSFORMATIONS THAT IMPACT ENERGY CONSUMPTION

In order to evaluate the potential impact of code transformations of communication-intensive kernels, it is necessary to first identify the remote data access patterns designed by the application developer. In this section, we discuss some access patterns and identify the type of transformations that aim at eliminating the cost factors discussed in Section 3.

It must be noted that in real world applications, the feasibility of such transformations would be constrained by a number of other factors like data dependencies, algorithm design, the memory model, the communication model, etc. The discussions here and the empirical results in Section 6 are therefore aimed at aiding the reader in obtaining an optimistic estimate of the maximum possible energy savings that can be expected.

4.1 Data Access Patterns

In order to study the energy behavior of data access patterns within a communication kernel, we needed to identify a set of design elements, based on which any one-sided communication-intensive pattern may be architected. These “design elements” correspond to basic operations over which a remote transfer may be built upon.

of local compute-based operations to those servicing remote transfers. This dependence on the “intensity” of a communication kernel is in alignment with similar empirical and energy model studies for shared-memory systems[6].

[2] Dynamic registration of memory is a very expensive operation [15, 22] and is therefore typically avoided

4.1.1 Basic Elements

RDMA Write constructs (or PUTs) in PGAS models may be built upon the following basic operations:

$P(x)$**:** This corresponds to the initiation of a one-sided Write transfer of x bytes of data from the source buffer on the active sender's node to a remote buffer on the passive receiver's memory. A call to this function does not guarantee completion of the data transfer. For an RDMA-capable interconnect solution that bypasses the kernel, this operation is equivalent to the transfer of contents from a pinned portion of the sender's memory to that of the receiver's memory. This pinning of memory with the OS corresponds to the registration of the memory location with the NIC. This simple operation simply corresponds. From the point of view of an OpenSHMEM developer, this corresponds to a call to *shmem_putmem()* where both the sender and the receiver addresses point to a portion on the globally accessible memory.

Q**:** This corresponds to a verification operation which guarantees completion of a previously posted PUT operation (P). In terms of OpenSHMEM terminology, this corresponds to a call to *shmem_quiet()* that returns once the the data contents of all previously posted PUTs are copied into the destination buffer at the receiver process.

M**:** A call to *memcpy()* that copies the content of the source buffer into a local buffer that is pre-registered with the NIC, thereby enabling RDMA operations on it.

A**:** A call to *shmem_swap* corresponds to an atomic operation that may be used by an active process to signal the completion of a transfer to a target process.

4.1.2 Examples

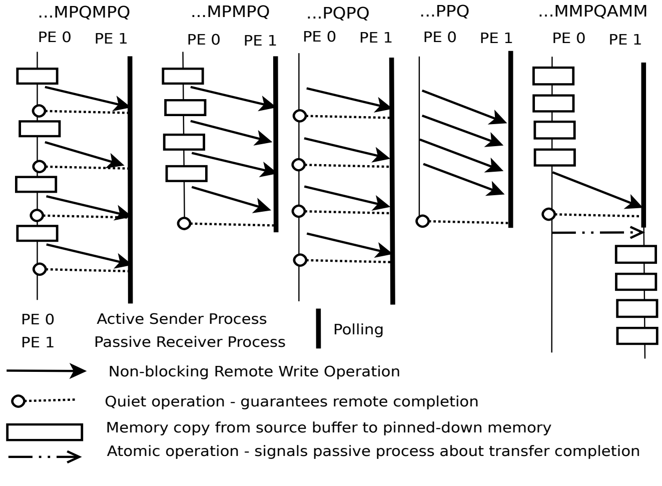
...MPQMPQ
PE 0
PE 1
...MPMPQ
PE 0
PE 1
...PQPQ
PE 0
PE 1
...PPQ
PE 0
PE 1
...MMPQAMM
PE 0
PE 1
PE 0
Active Sender Process
PE 1
Passive Receiver Process
Polling
Non-blocking Remote Write Operation
Quiet operation - guarantees remote completion
Memory copy from source buffer to pinned-down memory
Atomic operation - signals passive process about transfer completion

Figure 1: Line Diagrams of data access patterns

We discuss some examples of data access patterns that were designed as a combination of the basic elements discussed above[3][4]. Figure 1 illustrates these patterns, the impact of which, are discussed later in Section 6:

...$MPQMPQ$ **:** This benchmark represents the worst case which is a combination of all the expensive factors described above. Each PUT operation is preceded by a *memcpy()* and is followed by the *quiet* operation.

...$MPMPQ$ **:** Having every non-blocking PUT be preceded by a memory copy operation of the contents of the source buffer to a registered buffer takes into account the impact of using registered source buffers. After all the memory copies and the PUTs, this pattern ends with a single *quiet* operation, thereby guaranteeing completion of the data transfer.

...$PQPQ$ **:** Having every PUT be immediately followed by a *quiet* takes into account the impact of multiple blocking remote PUTs.

...PPQ **:** Having multiple consecutive PUTs followed by a single *quiet* takes into account the overhead of maintaining multiple handlers for non-blocking PUTs and of waiting for their remote completion.

...$MMPQAMM$ **:** To study the impact of the overhead due to aggregation of discrete user buffers, a number of factors need to be taken into consideration. These include: the cost of using multiple *memcpy()s* at the sender's side to copy data from the user-buffer into a pinned-down source buffer, the cost of actual transfer of the buffer contents to the remote process (using a single PUT), the cost of checking for remote completion of the transfer by the sender (a single *quiet*), the cost of signaling the completion of the transfer to the receiver process (a single *atomic* operation), the cost due to polling for the completion-signal by the receiver, and the cost at the receiver's side to copy back the contents from the destination buffer back to the final destination user buffers (using multiple *memcpy()s*). It must be noted that unlike the above patterns where the number of PUTs is equal to the number of user buffers, this pattern contains a single PUT following as many memory copy operations as the number of user buffers.

4.2 Transformations of Data Access Patterns

Figure 2 illustrates the set of microbenchmarks that were evaluated and the relation between them. The edges connecting the nodes of the graph depict different code transformations, the impact of which are discussed in Section 6.

[3] A note on the nomenclature used: A repetition of a substring in each pattern name corresponds to a discrete user buffer. e.g. Each 'MP' in ...$MPMPQ$ corresponds to operations over a different fragment at successive addresses in the heap. The actual count of this repetition i.e. the number of fragments, corresponds to the number of disjoint user buffers over which the access pattern operates.

[4] A note on the design of the microbenchmarks: Obtaining steady energy readings require running the synthetic microbenchmarks for large number of iterations. To avoid a data access pattern from falling prey to caching effects from past runs, it is essential to clear the contents of the cache before the start of each iteration.

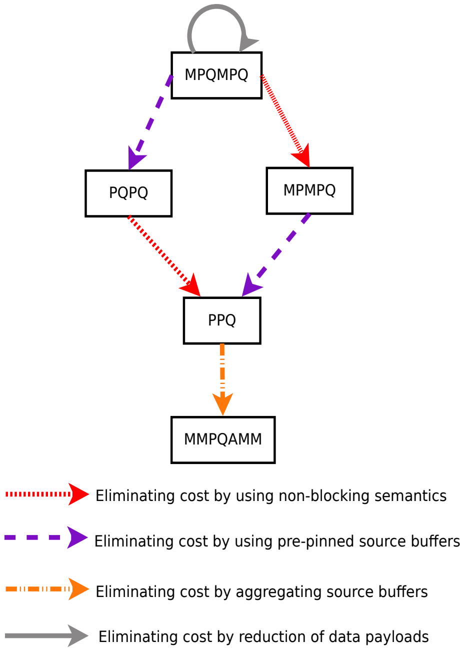
MPQMPQ
PQPQ
MPMPQ
PPQ
MMPQAMM
Eliminating cost by using non-blocking semantics
Eliminating cost by using pre-pinned source buffers
Eliminating cost by aggregating source buffers
Eliminating cost by reduction of data payloads

Figure 2: Different transformations of remote data access Patterns that have the potential impacting the energy profiles of communication-intensive application kernels.

We study the impact of four different OpenSHMEM code transformations that take into account the factors discussed in Section 3. Using the nomenclature above, we describe these transformations below:

Impact of using pinned source buffers: The impact of using unpinned source buffers and additional *memcpy()* operations may be studied by using globally visible or symmetric user-buffers, i.e. $MPQMPQ \rightarrow PQPQ$ and $MPMPQ \rightarrow PPQ$.

Impact of using non-blocking remote transfers: The possible cost-savings associated with converting blocking remote write operations to non-blocking ones may be studied by eliminating unnecessary calls to *quiet* after each PUT, i.e., $MPQMPQ \rightarrow MPMPQ$ and $PQPQ \rightarrow PPQ$.

Cost of aggregating user-buffers: The impact of using multiple memory copy operations to aggregate data into pinned memory instead of explicitly issuing multiple PUTs may be evaluated through the transformation $PPQ \rightarrow MMPQAMM$.

Cost of reducing the data-payload: While all the above transformations are dependent on the characteristics of the data access pattern within a communication kernel, this transformation deals with the the size of the data-payload as dictated by the input problem size. The impact of the data-payload size can be analyzed by studying the same test-case – $MPQMPQ$ – with different transfer-payload sizes.

Table 1: Characteristics of the power monitored node

Processor	Intel Xeon CPU E5-2690
Microarchitecture	Intel's Sandy Bridge
Hyperthreading support	Disabled
Main Memory	32 GB
Infiniband card	Mellanox MT27500, ConnectX-3
Linux kernel version	2.6.32 x86_64

It must be noted that many other transformations and their combinations such as $MPQMPQ \rightarrow MMPQAMM$ are also possible. However, since our scope lies on studying each of transformations independently, we do not discuss such cases here. Their impact may be compounded over more than one transformations listed above.

5. EXPERIMENTAL SETUP

5.1 Test-bed Characteristics

Our study was aimed at performing a fine-grained analysis of the impact on two main components that are dictate the energy and power consumption of a single compute node - the CPU and the memory. All experiments were conducted on two dual-socket Intel Sandy Bridge nodes described in Table 1. The two nodes are prototypes for an upcoming installation at the University of Technology Dresden, which are instrumented for fine-grained accurate power measurement[5]. Each node has instrumented voltage regulators (VRs) that are sampled with a sampling frequency of 1 KHz for both sockets and the four voltage lanes of the DIMMs on board. With the help of an FPGA, a digital filter is applied to smoothen the samples. Furthermore, a linear correction is applied to the measurement data coming from the VRs in order to ensure an accuracy of 3 %. Since the CPU and the DRAM are the single two most import consumers of energy on a node in a distributed system, we will focus on only these two components in the evaluation of our experiments. We limit our scope to the study of these two components. For studies on large scale systems, the contribution of the interconnect and the network topologies becomes crucial, as highlighted by recent study[13].

5.2 Software Stack Characteristics

The empirical results presented in this paper were obtained using synthetic OpenSHMEM microbenchmarks. They design of these benchmarks were based on the line diagrams depicted in Figure 1. The impact of the PGAS-based operations listed in 4.1.1 were studied using OpenSHMEM's constructs like *shmem_putmem()* and *shmem_quiet()* interfaces. The OpenSHMEM implementation used was *Mellanox Scalable SHMEM* (ver-2.2) over *OpenFabrics* Byte Transport Layer.

6. EMPIRICAL RESULTS

While one of the primary purposes of this paper is to discuss the impact of the transformations in access patterns, it is essential to understand the behavior of the patterns themselves. These are depicted in Figure 3. The figure

[5] More information about the High Definition Energy Efficiency Monitoring (HDEEM) project is available at `http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/hdeem`

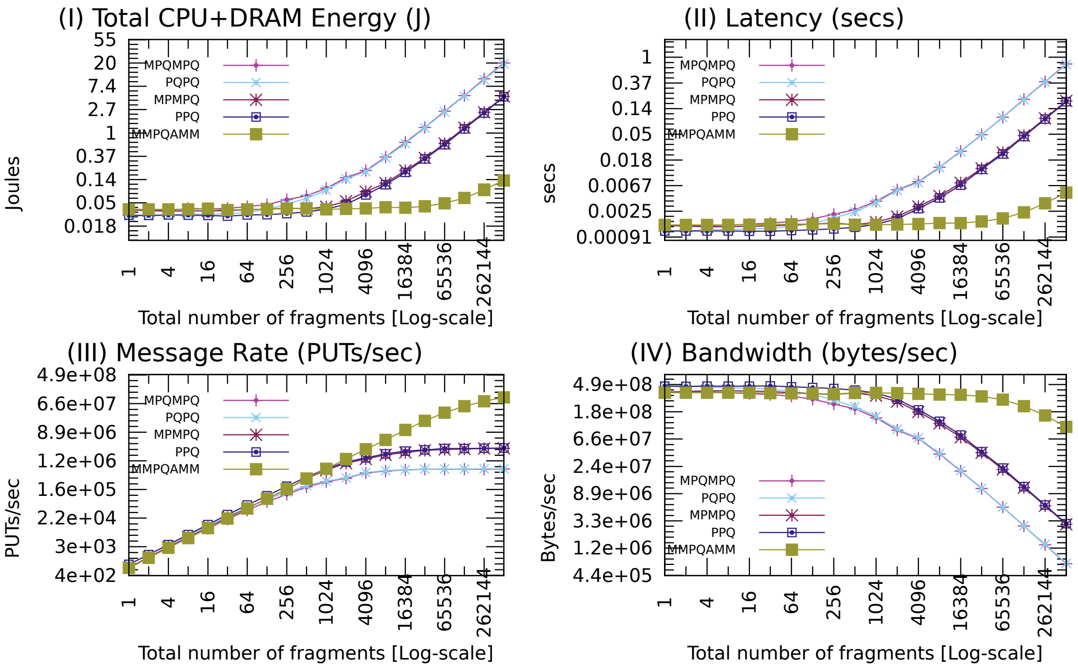

Figure 3: Impact of use of various data access patterns on the CPU+DRAM energy and the achievable latency for a remote PUT operation w.r.t. number of explicitly initiated transfers : Total Data-payload size = 0.5MB

depicts the energy costs, latency, message rate and bandwidth of a transfer of a 0.5MB transfer using different number of PUT opeperations, each of which corresponds to a discrete user-buffer (#Fragments on x-axis). We observe that most of the access patterns discussed before can be paired in terms of their closeness to their raw energy and latency values. $MPQMPQ$ and $PQPQ$ have the highest energy and latency cost. This is accompanied with a lower bandwidth and message rate. This trend can be attributed to the penalty associated with polling-based operations and additional memory management necessary to ensure remote completion of the transfers. $MPMPQ$ and PPQ have readings close to each other, but are cheaper than $MPQMPQ$ and $PQPQ$ because of the absence of the blocking semantics. When the number of PUTs is beyond 256, aggregation of data buffers ($MMPQAMM$) lead to the minimal energy consumption and latency. Another point to note is that for most of the patterns, the message-rate becomes limited beyond for less than 8KB per PUT (i.e. #Fragments = 64), and is accompanied with a steady drop in the bandwidth of the transfer. While these raw values provide an overview of the behavior of the data patterns, they do not give an indication of the potential savings due to the factors discussed in Section 3. A detailed study using the transformations discussed in Section 4 is presented next.

In this section, we discuss the impact of various transformations of data access patterns, as discussed in Section 3. These are supplemented with empirical results depicted in Figures 4, 5, and 6. These figures illustrate the impact of the transformations on various cost metrics.

In our case, the cost "metrics" studied are:

- Energy consumption by the CPU
- Energy consumption by the DRAM
- Latency of the transfer
- EDP or Energy Delay Product[6]

The "impact" of each cost metric is calculated in terms of the percent *reduction* in one of the above metrics. If a transformation T is applied on a data access pattern $C_{initial}$ such that: $T(C_{initial}) \rightarrow C_{final}$, then the impact of T in terms of percent reduction in a cost-metric M may be calculated as:

$$I = \frac{M(C_{initial}) - M(C_{final})}{M(C_{initial})} * 100$$

For all of these experiments, the graphs depict the values of various metrics as measured at the compute node servicing the active sender processes responsible for initiating the remote write operations. We restrict our discussion to study the behavior of this process and not the passive receiver process.

[6] While CMOS circuits have the ability to trade performance for energy savings, it becomes challenging to optimize of both simultaneously. The EDP, first proposed by Horowitz[8, 9], takes into account both the energy and the time costs in an implementation-neutral manner. For cases, where energy and performance have equal importance, this metric can be calculated as a product of the energy consumed and the time taken. For more complicated cases, where performance is given a higher priority, the weight of the "delay" factor is increased by squaring or cubing it[13].

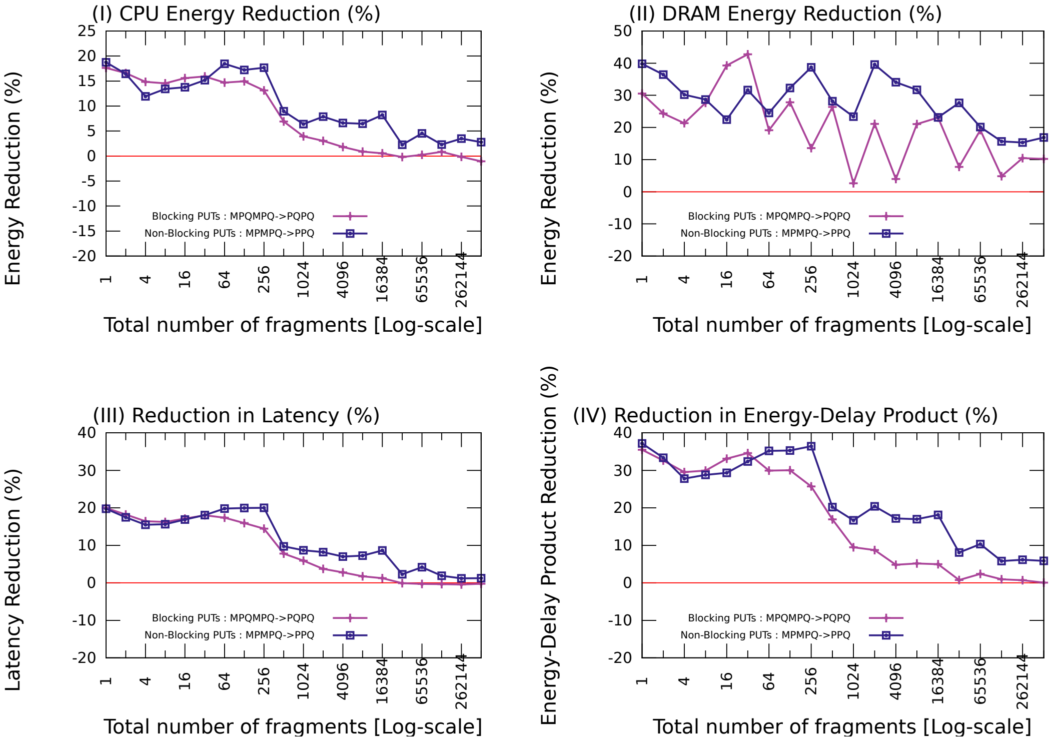

Figure 4: Impact of using pinned data buffers : Data-payload size = 0.5MB

It must be noted that the energy consumption of a passive process that's polling at a barrier, waiting for the completion of a transfer, cannot be ignored while performing large scale studies of distributed applications. In fact, our past study[10] indicates that the energy consumption increases proportionally with the time and its scale is very high. However, since the polling activity corresponds to a constant power consumption, it can be safely ignored in the following discussions that focus on the impact due to the remote data access patterns.

6.1 Impact of Using Pinned Buffers

From Figure 4, we observe that there is a definite reduction in energy consumption by the CPU and the DRAM while using NIC-registered user buffers instead of non-registered ones. However, the benefit of this transformation reduces with a rise in the number of discrete data buffers, i.e. with a rise in the number of PUT operations.

Influence of other cost factors:. We see that there exists a variation in the impact based on the blocking semantics of the PUT operations. The plots in the figure depict these two possible outcomes as "Blocking PUTs: $MPQMPQ \rightarrow PQPQ$" and "Non-Blocking PUTs: $MPMPQ \rightarrow PPQ$". Some observations include:

With Blocking PUTs: We observe that the impact on (or, the percent reduction in) the the CPU energy consumption and the latency is as high as 20% in case of bulk transfers. This is not surprising, as this type of transformation results in elimination of unwanted memory copy operations, which directly benefit the energy cost and the latency. This elimination of bulk memory copy operations leads to DRAM energy savings as high as 40%. The impact of this transformation however drops to less than 5% in terms of CPU energy and almost zero in case of latency. This downward trend is observable especially when the number of PUTs increases beyond 512 (i.e. buffer size $< 1KB$ per PUT). This point corresponds to the message rate limit (Figure 3) as well as the relatively negligible CPU energy costs of small memory copies. Nevertheless, since the size of the memory footprint of the user buffers remains constant across all data points, the DRAM energy savings do not drop below 20%.

With Non-Blocking PUTs: Similar to the case with blocking PUTs, we observe that the reduction in CPU energy consumption, latency, and the energy-delay product is higher for large bulk buffers (fragment count $<$ 512, buffer size $< 1KB$ per PUT). The drop in the energy savings for smaller buffers may be attributed to both the negligible savings while eliminating small memory copy operations and, the rise in the overhead of managing large number of in-progress PUTs. Beyond 16K PUTs, this transformation leads to CPU energy savings as low as 5%. One of the primary observations with regards to the DRAM energy consumption is the overall lesser impact of this transformation on blocking PUTs when compared with non-blocking

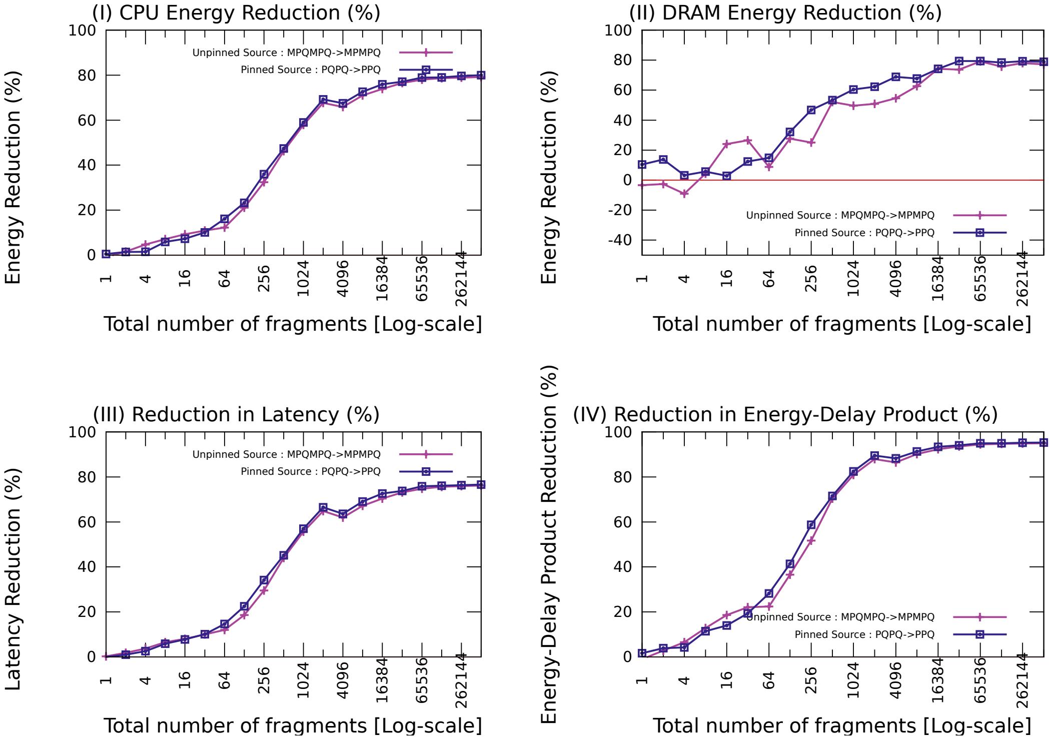

Figure 5: Impact of transforming multiple blocking operations to non-blocking

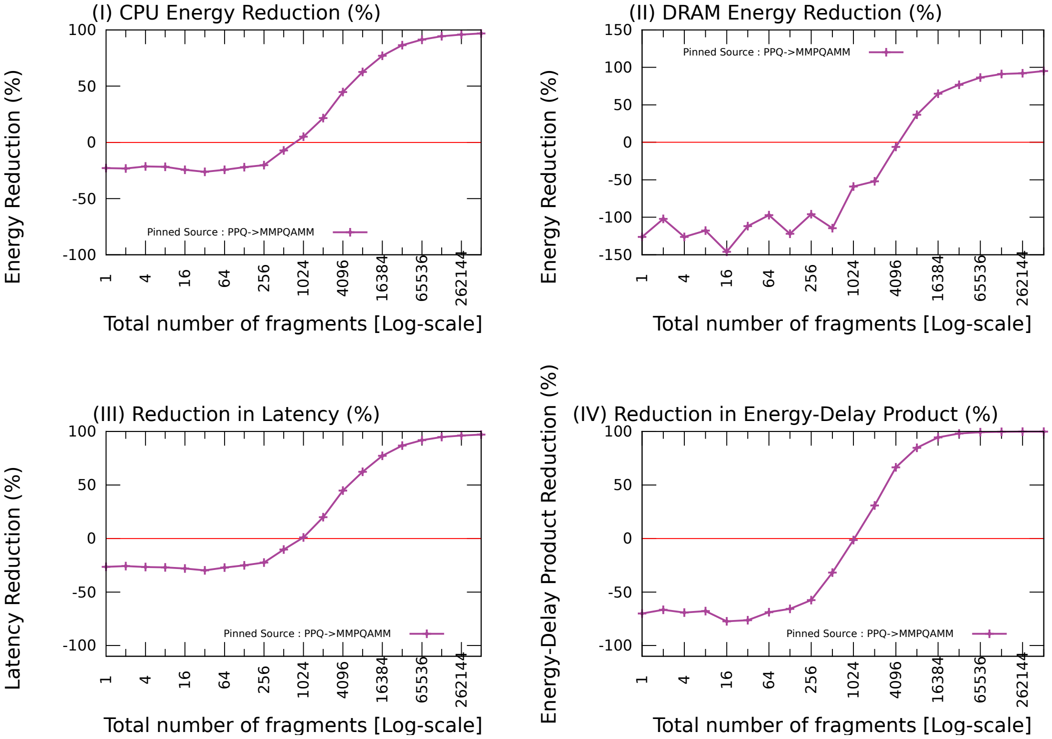

Figure 6: Impact of aggregation of multiple data buffer

PUTs. An interesting observation is the oscillating trend in DRAM energy savings. This was surprising because the total size of the data payload being handled across all the data points remains constant (0.5MB) and the primary source of DRAM energy savings in this transformation is the elimination of local memory copy operations. We are currently investigating the reason for this trend.

6.2 Impact of Using Non-Blocking Remote Transfers:

From Figure 5, we observe that the impact of replacing blocking transfers by non-blocking versions is significant in terms of reduction in CPU energy, latency, and the energy-delay product. As shown, the positive impact on the energy and the latency rises with an increase in the number of discrete PUTs and hits a limit (80%) when this count rises beyond 256. This can be attributed to the fact that the benefits of launching multiple non-blocking transfers is overshadowed by the cost of ensuring completion of these transfers (during the *quiet* operation). The benefits on the energy-delay product is significant. The 80% reduction in CPU energy and latency corresponds to an improvement in energy delay product by almost 95%. Slight variations in this behavior exist, as discussed below.

Influence of other cost factors:. We see that there is very little different between the impacts based on whether the source data buffers are pre-registered or not. The plots in the figure depict these two possible outcomes as "Unpinned Source: $MPQMPQ \rightarrow MPMPQ$" and "Pinned Source: $PQPQ \rightarrow PPQ$". Some observations include:

6.3 Impact of Aggregation of Buffers:

Unlike other data access patterns discussed in the text, a an access pattern like $MMPQAMM$ corresponds to an active participation by both the sender and the receiver. Moreover the RDMA-based data transfer is limited to a single PUT operation. The cost associated with handling multiple user buffers is dependent solely on the cost of local memory copy operations.

Figure 6 depicts the impact of convering a pattern like PPQ to $MMPQAMM$. The observations are described below.

For bulk transfers (#Fragments≤1024): At this phase, a pattern like PPQ is characterized by few bulk PUTs directly from the source buffer to that target. The transformed pattern $MMPQAMM$ is characterized by bulk local memory copy operations first by the sender, and later on by the receiver. The lattern pattern significantly raises the CPU and DRAM energy consumption. Moreover, this phase corresponds to the peak bandwidth achievable using PPQ. Thus, we see a negative impact on the energy metrics (25% to 125%) and the latency (25%). This negative impact amortizes any potential energy savings achievable through the use of a single bulk blocking PUT.

For small transfers (#Fragments>1024): We observe that the overall CPU and DRAM energy savings achieved using this transformation increases with the count of discrete source data buffers (fragmentation). Besides the obvious elimination of the software overhead of multiple PUTs and handling multiple in-progress transfers in PPQ, the high energy savings may be attributed due to this pattern's limiting message-rate and dropping bandwidth, as shown in Figure 3.

7. CONCLUSION

In this paper, we established the notion that the design of data access patterns play a critical role in impacting the energy profiles of communication-intensive PGAS applications. We investigated a number of factors that are dictated by the programming model and algorithm design that affect the energy cost of a process initiating a remote data transfer. These include – the contiguity of the data buffers in the memory, the total size of the payload being transferred, the registration status of the source buffers, the completion semantics of the data transfer operations, etc. For a fixed size of data-payload that is remotely transferred by a process, the extent of impact of these factors depend on the number of explicitly initiated data transfer.

Some of the lessons learned include - (a) Energy savings achieved by using pinned-down source buffers reduces with a rise in the number of explicitly initiated PUT operations, (b) Energy savings due to the use of non-blocking semantics is higher for smaller sized transfers; such savings hit a limit due to additional overhead due to management of multiple outstanding remote transfers, (c) Aggregating bulk-sized buffers into contiguous memory locations has a negative impact on the energy savings, the latency and the energy-delay product. Using multiple smaller transfers tend to benefit significantly in terms of such savings. We discuss these by evaluating multiple transformations on access patterns that account for these factors.

In future we hope to study the feasibility of these energy savings on real-world large-scale applications.

8. ACKNOWLEDGMENTS

This work has been supported through resources from the High Definition Energy Efficiency Monitoring (HDEEM) research project at the Center for Information Services and High Performance Computing (ZIH) at the Technische Universität Dresden. Special thanks are due to Daniel Hackenberg, Daniel Molka, Robert Schöne, and Thomas Ilsche for their help and guidance while using these resources.

9. REFERENCES

[1] N. Aboughazaleh, B. Childers, R. Melhem, and M. Craven. Collaborative compiler-os power management for time-sensitive applications. Technical report, Department of Computer Science, University of Pittsburgh, Department of Computer Science, University of Pittsburgh, 2002.

[2] J. Aliaga, M. Dolz, A. Martin, R. Mayo, and E. Quintana-Orti. Leveraging task-parallelism in energy-efficient ILU preconditioners. In A. Auweter, D. Kranzlmueller, A. Tahamtan, and A. Tjoa, editors, *ICT as Key Technology against Global Warming*, volume 7453 of *Lecture Notes in Computer Science*, pages 55–63. Springer Berlin Heidelberg, 2012.

[3] P. Alonso, M. Dolz, F. Igual, R. Mayo, and E. Quintana-Orti. Reducing energy consumption of dense linear algebra operations on hybrid cpu-gpu platforms. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 56–62, July 2012.

[4] M. Barreda, S. CatalÃąn, M. Dolz, R. Mayo, and E. Quintana-Orti. Automatic detection of power bottlenecks in parallel scientific applications. *Computer Science - Research and Development*, 29(3-4):221–229, 2014.

[5] M. Castillo, J. Fernandez, R. Mayo, E. Quintana-Orti, and V. Roca. Analysis of strategies to save energy for message-passing dense linear algebra kernels. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 346–352, Feb 2012.

[6] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. A theoretical framework for algorithm-architecture co-design. In *Proc. IEEE Interational. Parallel and Distributed Processing Symp. (IPDPS)*, Boston, MA, USA, May 2013.

[7] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18–28, Jan 2005.

[8] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of*, 31(9):1277–1284, Sep 1996.

[9] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *Low Power Electronics, 1994. Digest of Technical Papers., IEEE Symposium*, pages 8–11, Oct 1994.

[10] S. Jana, O. Hernandez, S. Poole, C.-H. Hsu, and B. Chapman. Analyzing the energy and power consumption of remote memory accesses in the openshmem model. In S. Poole, O. Hernandez, and P. Shamis, editors, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools*, volume 8356 of *Lecture Notes in Computer Science*, pages 59–73. Springer International Publishing, 2014.

[11] M. Kandemir, A. Choudhary, P. Banerjee, J. Ramanujam, and N. Shenoy. Minimizing data and synchronization costs in one-way communication. *Parallel and Distributed Systems, IEEE Transactions on*, 11(12):1232–1251, Dec 2000.

[12] V. A. Korthikanti and G. Agha. Towards optimizing energy costs of algorithms for shared memory architectures. *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures - SPAA '10*, page 157, 2010.

[13] J. Laros, K. Pedretti, S. M. Kelly, W. Shu, K. Ferreira, J. Dyke, and C. Vaughan. *Energy-efficient high performance computing*. London: Springer, 2013.

[14] D. Li, B. R. de Supinski, M. Schulz, K. Cameron, and D. S. Nikolopoulos. Hybrid MPI/OpenMP power-aware computing. *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, 2010.

[15] J. Liu, W. Huang, B. Abali, and D. K. Panda. High performance vmm-bypass i/o in virtual machines. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ATEC '06, pages 3–3, Berkeley, CA, USA, 2006. USENIX Association.

[16] J. Nieplocha and B. Carpenter. Armci: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In J. Rolim, F. Mueller, A. Zomaya, F. Ercal, S. Olariu, B. Ravindran, J. Gustafsson, H. Takada, R. Olsson, L. Kale, P. Beckman, M. Haines, H. ElGindy, D. Caromel, S. Chaumette, G. Fox, Y. Pan, K. Li, T. Yang, G. Chiola, G. Conte, L. Mancini, D. MÃľry, B. Sanders, D. Bhatt, and V. Prasanna, editors, *Parallel and Distributed Processing*, volume 1586 of *Lecture Notes in Computer Science*, pages 533–546. Springer Berlin Heidelberg, 1999.

[17] S. F. Rahman, J. Guo, and Q. Yi. Automated empirical tuning of scientific codes for performance and power consumption. In *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, HiPEAC '11, pages 107–116, New York, NY, USA, 2011. ACM.

[18] J. Shalf, S. Dosanjh, and J. Morrison. Exascale Computing Technology Challenges. pages 1–25, 2011.

[19] J. Torrellas, D. Quinlan, A. snavely, and W. Pinfold. Thrifty: An exascale architecture for energy-proportional computing.

[20] A. Venkatesh, K. Kandalla, and D. Panda. Evaluation of energy characteristics of mpi communication primitives with rapl. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 938–945, May 2013.

[21] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji. Designing energy efficient communication runtime systems: a view from pgas models. *The Journal of Supercomputing*, 63(3):691–709, 2013.

[22] J. Wu, P. Wyckoff, and D. K. Panda. Pvfs over infiniband: Design and performance evaluation. In *IN THE 2003 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP) 03*, pages 125–132, 2003.