Impact of Frequency Scaling on One Sided Remote Memory Accesses

Siddhartha Jana  
Department of Computer Science  
University of Houston  
Houston, Texas, USA  
sidjana@cs.uh.edu

Barbara Chapman  
Department of Computer Science  
University of Houston  
Houston, Texas, USA  
chapman@cs.uh.edu

Abstract—CPU Frequency scaling is a common approach used for achieving energy savings in parallel applications. A typical approach for achieving power savings is by reducing the frequency of a processor whenever the invested CPU cycles do not contribute to the progress of an application (e.g. polling for events). Many recent research efforts have been directed towards employing this approach within HPC applications. However, one of the important challenges that have remained unexplored is its potential impact on data transfer operations within communication-intensive applications.

This paper presents a detailed analysis of the impact of frequency scaling on the performance and energy efficiency of one-sided point-to-point communication within PGAS models. As an outcome of this study, we observe that the extent of this impact depends on a number of factors like the implementation approach of a data transfer operation and the number of explicit remote transfers initiated. Besides the characteristics of the software stack, hardware features like the microarchitectural design also impact the performance of such data movement while adopting voltage scaling techniques. These inferences are supplemented with empirical analysis of common approaches of implementing remote write (PUT) operations.

The empirical results presented in this paper should act as a guide for future design of software solutions targeting energy efficiency in distributed systems.

Keywords—Dynamic Voltage Frequency Scaling (DVFS); Energy efficiency; Energy Delay Product (EDP); Remote Data Memory Accesses; Energy Efficient RDMA; Data Access Patterns; OpenSHMEM; PGAS

I. INTRODUCTION

As power consumption continues to be a major concern for exascale systems, research efforts have been directed towards using hardware and software codesign principles to achieve energy efficiency. In accordance to this, one common approach is to use frequency scaling capabilities of modern processors to achieve energy savings. This is commonly referred to as Dynamic Voltage Frequency Scaling (DVFS). Reducing the frequency allows the processors to operate at a lower voltage level thereby leading to energy savings.

In a one-sided point-to-point communication model, a single software agent (process). A common scenario in the design of PGAS kernels is for the passive process to rely on the completion of this data transfer in order to make further progress into the application. The CPU servicing this process is therefore subjected to polling for a certain semaphore-based event that signals the completion of the transfer. At such a point, theoretically, energy savings may be achieved by scaling down the frequency of a CPU core that services the passive process. This time-frame is commonly referred to as ‘slack period’ and there have been multiple research efforts directed towards using DVFS techniques to reduce the energy consumption without significant performance impact[1], [2], [3], [4]. This paper highlights the claim that this lack of “significant performance impact” during data movement in a distributed environment is heavily dependent on the underlying implementation. It presents empirical evidence that the extent of these savings depends on the implementation approach of one-sided communication interfaces. This analysis is presented with respect to the use of one-sided remote write (PUT) operations in OpenSHMEM[5], an SPMD-based PGAS model. This paper discusses the potential impact on the energy and latency costs incurred by the sender and the receiver process1 in an environment where the latter is serviced at a reduced CPU frequency. The main contributions of this paper are:

- Discussion of the challenges of using DVFS in a distributed environment (Section III)
- Description of different cost factors within the software stack that affect the energy consumption and the performance of remote data transfers (Section IV)
- Discussion on common approaches of implementing remote PUT operations that have the potential of being affected by DVFS (Section V)
- An empirical analysis that presents the impact of using DVFS on the above approaches (Section VII)

The empirical analysis incorporates a fine-grained study of the energy consumption by the CPU and the DRAM servicing the sender and receiver processes. These readings were obtained using computational resources described in Section VI.

1or in OpenSHMEM terminology, ‘processing element (PE)’
The results presented in this work should be useful to system programmers incorporating DVFS techniques in a distributed environment.

**Note:** In this paper, the term 'Sender' refers to an SPMD process that initiates an RDMA operation to access a remote data object, and the term 'Receiver' refers to the process that owns the remote data object. No additional meaning is implied in terms of the extent of participation while servicing the data movement.

## II. Related Work

There have been multiple research efforts directed towards exploring energy savings using DVFS during blocking data transfer and synchronization operations. Some examples include work by Newsom et al. [1], Gamell et al. [2], Li et al. [3], and Lim et al. [4].

Newsom et al. [1] use locality-aware of PGAS data transfers to determine the feasibility of using DVFS for energy savings. Their analysis takes into account the energy savings achievable using hardware-controlled as well as application (user/compiler) driven DVFS techniques. They highlight the potential energy savings achievable at the application layer, by discussing the impact of applying frequency scaling while prefetching remote data objects within stencil-based kernels.

Gamell et al. [2] explore the feasibility of using DVFS during different UPC operations. Their experiments are limited to communication among multiple cores within a single node. They conclude that energy savings using DVFS is achievable during UPC `memget` and `wait` operations.

Li et al. [3] explore opportunities for energy savings using DVFS and DCT within Hybrid MPI+OpenMP applications. In their work, they introduce a power-aware performance prediction model which aid in determining the frequency and concurrency (number of threads) settings for different OpenMP phases in hybrid applications.

Lim et al. [4] use DVFS techniques within the MPI runtime library. Their approach is geared towards controlling the frequency at the granularity of individual MPI calls. For cases where the overhead of frequency scaling is too high, the granularity is increased to control frequency switch across multiple MPI function calls.

All the above papers focus on using frequency scaling as a means to reduce the energy consumption during slack-periods during data movement in a distributed-memory environment. One of the major questions that remain unanswered is the performance impact on the data movement due to (a) the actual value of the CPU frequency and, (b) the data access pattern adopted by PGAS communication phases. This paper addresses these questions by comparing different implementation approaches of point-to-point communication interfaces within PGAS models.

## III. Challenges of Using DVFS in a Distributed Environment

- **Choosing the correct frequency level:** CPU cores in modern processors are capable of operating at multiple different frequencies. While operating a CPU at a lower frequency leads to power savings, running it at a higher frequency leads to increased throughput. It has been well established that the choice of this operating frequency depends on the design of the application kernel. Recent study by Gotz et al. [6] present empirical evidence that shows that applications with varying computational demands attain energy efficiency at different CPU clock speeds. This observation is in alignment with the Roofline Model of Energy [7], which relates these 'computational demands' to the ratio of the number of compute operations to memory accesses within application kernels. This article focuses on only RDMA transfers. More specifically, it describes how operating a receiver at different frequencies during a point-to-point data transfer, leads to varying performance and energy consumption.

- **Sibling cores with contradicting frequency demands:** While designing an energy efficient software, one must be aware of the impact of frequency scaling of a single CPU core on the performance of other cores. The extent of this impact varies with the architectural design of the target processor. For example, in case of the Sandy Bridge architecture, all the CPU cores lie on the same frequency plane [8]. This means that a single CPU core cannot operate at a different frequency than others. In such an environment where all the cores share the same clock-speeds, conflicting demands by a single CPU core may affect the performance of the rest of the cores. In case of PGAS applications targeting a multi-core environment, using DVFS has the potential of severe performance degradation.

- **DVFS dependent Memory/Cache bandwidth:** DVFS also affects the local cache and memory bandwidths within a processor. Schöeene et al. [9], in their study, present empirical evidence suggesting that this impact varies among different x86_64 processors. For a Sandy Bridge-EP processor, they show that the memory bandwidth can drop by as much as 44% depending on the operating frequencies and the number of cores in use. The L3 cache bandwidth also gets affected and follows an almost linear relationship with the drop in the CPU frequency. In case of PGAS applications, this impact on the bandwidth of the internal memory

---

2 A hardware logic unit, called the Power Control Unit, is responsible for ensuring that the internal clock of all the cores is maintained at a frequency that meets the core with the highest performance demand.

3 This is because in a Sandy Bridge architecture, the interconnecting ring-bus runs on the same frequency as the CPU cores. Also, the cores, the bus, and the last-level shared L3 cache, all lie on the same power plane [8].
Table I: Overview of different factors that contribute to the performance and energy consumption. Each row lists the cost factor, the system components involved as well as the potential impact on the CPU and DRAM energy/performance metrics.

<table>
<thead>
<tr>
<th>(I) Phase</th>
<th>(II) Components affected</th>
<th>(III) Time spent within this phase</th>
<th>(IV) Activity of components affected</th>
<th>(V) Sender CPU Energy</th>
<th>(VI) Sender DRAM Energy</th>
<th>(VII) Receiver CPU Energy</th>
<th>(VIII) Receiver DRAM Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Initiating async PUTs</td>
<td>Sender CPU, Sender DRAM</td>
<td>No impact</td>
<td>No</td>
<td>Constant</td>
<td>Constant</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>B: Polling for completion</td>
<td>Sender CPU, Sender DRAM</td>
<td>No impact</td>
<td>No</td>
<td>Constant</td>
<td>Constant</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>C: Polling for UNPACK COMPLETE</td>
<td>Sender CPU</td>
<td>Increased time period</td>
<td>No</td>
<td>Energy rise proportional to the time spent</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>D: Polling for TRANSFER COMPLETE</td>
<td>Receiver CPU, Receiver DRAM (DMA)</td>
<td>No impact</td>
<td>Yes</td>
<td>–</td>
<td>–</td>
<td>Trade off between time spent idling v/s initiating local memory copy operations</td>
<td>Reduction of energy</td>
</tr>
<tr>
<td>E: Unpacking user buffers</td>
<td>Receiver CPU, Receiver DRAM</td>
<td>Increased time period</td>
<td>Yes</td>
<td>–</td>
<td>–</td>
<td>Trade off between energy saved while polling v/s initiating local memory copy operations</td>
<td>Depends on the impact on memory access rate</td>
</tr>
<tr>
<td>F: Packing user buffers</td>
<td>Sender CPU, Sender DRAM</td>
<td>No impact</td>
<td>No</td>
<td>Constant</td>
<td>Constant</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>G: Polling for incoming packets + Copying data to destination buffers</td>
<td>Receiver CPU, Receiver DRAM</td>
<td>Increased time period</td>
<td>Yes</td>
<td>–</td>
<td>–</td>
<td>Trade off between energy saved while polling v/s initiating local memory copy operations</td>
<td>Depends on the impact on memory access rate + rate of incoming buffers</td>
</tr>
</tbody>
</table>

hierarchy leads to an impact on the performance of data transfers - both remote or local to a process.

IV. ENERGY COST FACTORS ASSOCIATED WITH RDMA TRANSFERS

In this section, we identify multiple energy and performance factors within the software stack which have the potential of affecting the energy consumption of PGAS implementations of point-to-point interfaces. These cost factors are mapped to various phases of implementation approaches illustrated in Figure II. This section describes the cost factors with respect to remote write (PUT) operations. However, it must be noted they are also applicable for implementations of remote read (GET) operations.

To complement this discussion, Table I maps these costs to the CPU and the DRAM servicing the sender and the receiver processes.

- **Initiating Asynchronous PUTs (Phase A):** This phase, executed by the sender, corresponds to the initiation of a one-sided PUT operation of \( x \) bytes, from its local address space to a remote memory address at the receiver.

With RDMA-capable hardware stack, this operation can progress by bypassing the Operating System\(^4\). While the sender’s CPU is initially responsible for communicating the required information to the NIC (Network Interface Controller) drivers, the actual data transfer can proceed without any CPU intervention at the sender and the receiver’s end\(^5\). In terms of OpenSHMEM, this corresponds to a \textit{shmem\_put*} operation. It must be noted that the energy and performance costs during this phase are not only dependent on the memory footprint of the transferred payload, but also the number of explicitly initiated PUTs\([11],[12],[13]\).

- **Polling for completion (Phase B)** This phase, executed by the sender, corresponds to a polling operation which, on completion, guarantees completion of all previously initiated PUTs during Phase A. In terms of OpenSHMEM, this corresponds to a \textit{shmem\_quiet} operation.

- **Polling for UNPACK COMPLETE (Phase C)** This phase is applicable for implementations that require unpacking of received data packets and distribution of its contents to discrete user buffers. In such cases, this

\(^4\)OS-bypass is possible by registering and pinning down a set of contiguous memory locations with the NIC. This helps avoid page faults and control access permissions, respectively.

\(^5\)The lack of CPU intervention is made possible using DMA based techniques in modern processors. In case of Intel’s processors, the DMA transfer between the NIC and the cache hierarchy is serviced using Intel Data Direct I/O (DDIO) technology[10] via the ring interconnecting bus. This bus is shared by the CPU cores, the shared L3 cache, graphics, and the system agent (which among other components, includes the PCI-Express 2.0).
Table II: Block diagrams and pseudo-codes used to study some of the different approaches of handling remote PUT operations. (i) Servicing PUTs with no participation by the receiver (ii) Servicing PUTs with active participation by the receiver (iii) Servicing PUTs with an additional thread supporting the receiver

<table>
<thead>
<tr>
<th>Line charts</th>
<th>Pseudo codes</th>
</tr>
</thead>
</table>
| ![Diagram A](#) | ```
shm_mem_barrier_all()
if (myid == sender)
    cflush(...)
    foreach (src_buffer[i])
        do
            shm_mem_put(src_buffer[i],
                        dest_buffer[i])
        done
    shm_mem_quiet()
endif
shm_mem_barrier_all()
``` |
| ![Diagram B](#) | ```
index = 0; cflush(...) 
shm_mem_barrier_all()
if (myid == sender)
    foreach (src_buffer[i])
        do
            /* Pack all buffers into 
            stemp_buff */
            stemp_buff[index] <- src_buffer[i]
            size = sizeof(src_buffer[i])
            index = index + size
        done
    shm_mem_put(stemp_buff, dtemp_buff)
    shm_mem_quiet()
else /* myid == receiver */
    shm_mem_int_wait_until (flag ...)
    foreach (dest_buffer[i])
        do
            /* Unpack all buffers from 
            dtemp_buff */
            dest_buffer[i] <-
                dtemp_buff[index]
            size = sizeof(dest_buffer[i])
            index = index + size
        done
endif
``` |
| ![Diagram C](#) | ```
shm_mem_barrier_all()
if (myid == sender)
    cflush(...)
    foreach (src_buffer[i])
        do
            shm_mem_put(src_buffer[i],dtemp_buffer[i])
        done
    shm_mem_quiet()
endif
```
phase is an additional overhead borne by the sender after Phase B. It corresponds to a polling operation by the sender to receive an acknowledgment by the receiver that the unpacking phase is complete. The completion of phase C indicates that the remote memory locations have been updated with the corresponding data contents and are available for future local/remote accesses.

- **Polling for TRANSFER COMPLETE (Phase D)** This phase, executed by the receiver, corresponds to polling for a signal sent by the sender to flag the arrival of data packets at the receiver. This is a crucial factor in case of implementations that rely on the receiver to participate in the data transfer operations.

- **Unpacking user buffers (Phase E)** It must be noted that the completion of Phase D does not guarantee that the incoming data contents arrived at the final destination buffers. It might be the case that the contents need to be copied from a temporary storage buffer to the final destination. This phase corresponds to this software overhead borne by the receiver while transferring the contents to its final intended destination address. It must be noted that this Phase E is identical to Phase C; the only major difference being the actual process servicing it.

- **Packing user buffers (Phase F)** This phase, executed by the sender, corresponds to the preparation of a user buffer before initiating an RDMA operation. This involves memory management tasks like copying the contents of user buffers from user address space to pinned-down memory buffers. In case of OpenSHMEM, the communication model does not require the source buffer of a remote PUT operation to be remotely accessible (‘symmetric’ in SHMEM terminology) itself. In such a case, this operation is typically performed by the underlying implementation.

- **Memory management by a support thread (Phase G)** Unlike phase E where the receiver bears the overhead of managing the contents at the intended destination addresses, an implementation may choose to use a dedicated software agent to handle this operation. Depending on the implementation, this asynchronous agent may be launched during the initialization phase of an application (shmem_init, in case of OpenSHMEM) and remain active throughout the lifetime of a process. The use of computational resources to service this agent is an additional cost factor that needs to be accounted for.

V. APPROACHES FOR IMPLEMENTING RDMA PUTS

This section highlights some common approaches for implementing remote PUT operations in a PGAS library. As described below, these approaches may be divided into a number of different phases listed in Section IV. The data flow within these approaches are illustrated as line charts in Table II along with their pseudo-codes. The achievable bandwidth for each of these patterns at different CPU (receiver) frequencies is depicted in Figure 1. Clearly, changing the frequency of the receiver leads to an impact in the performance of the data transfer. This is discussed later in Section VII.

1) **Servicing PUTs with No Active Participation by the Receiver (Table II[A])**

This case corresponds to the ideal scenario with minimal CPU intervention and software overhead during a remote write operation (PUT). Low latency of such transfers is typically achieved using RDMA support provided by modern interconnects like InfiniBand. Such operations do not require the active participation of the remote CPU and bypass the OS on the remote node.

2) **Servicing PUTs with Active Participation by the Receiver (Table II[B])**

This corresponds to cases where additional software overhead is added by the communication library to implement data access patterns that are not directly supported by the underlying hardware. In order to handle the transfer of discrete user buffers across the network, an implementation may choose to aggregate or pack multiple discrete memory fragments into a single contiguous memory chunk. This operation is performed locally before transferring the contents to the remote host. On detecting the arrival of the incoming packets (which typically involves a handshaking signal), the receiver is responsible for unpacking the contents of the buffer and copying them to their intended destination buffers. One such use case is implementation of strided-data communication interfaces, which are common among PGAS models. These interfaces allow the user to initiate transfer of multiple data objects that are not aligned contiguously in memory.

3) **Servicing PUTs Using an Additional Software Agent Supporting the Receiver (Table II[C])**

In order to ensure progress of asynchronous PUT operations without interrupting the receiver’s CPU, an additional thread may be launched at the receiver’s end for polling the network for incoming transfers. Once this thread detects an incoming packet, it aids the completion of the data transfer operation by copying the data contents to the final destination buffers. This leaves the receiver free to perform a different set of operations, thereby leading to communication-computation overlap. A use case for such an approach is ensuring the progress of asynchronous communication on platforms that lack network support for RDMA-based transfers.
VI. EXPERIMENTAL SETUP

A. Methodology

In order to conduct the study with respect to different approaches of implementing RDMA patterns, we designed synthetic microbenchmarks based on data access patterns in communication libraries. The pseudo codes for each of these benchmarks are listed in Table II.

The patterns were evaluated using two OpenSHMEM processes (PEs), each launched on a separate but identical compute node and bound to their respective CPU cores. Each PE played the role of either the sender or the receiver, but not both. This isolation ensured a comparative study between the two processes. In Table II, the first two patterns were evaluated using the Mellanox Scalable SHMEM version 2.2 (over OpenFabrics Byte Transport Layer). For the third pattern, the OpenSHMEM reference implementation was used (over GASNet with IBV conduit).

As discussed before, the purpose of the study is to analyze the impact of a receiver operating at a scaled-down CPU frequency. In our experiments, the CPU frequency of the sender was held constant at 2.901GHz (turbo-frequency). To study the impact of frequency scaling of the compute node servicing the receiver, the experiment was repeated multiple times with the node operating initially at 2.901GHz and later at 2.4GHz and 1.2GHz.

The results of the microbenchmarks are depicted in Figures 2, 3, and 4. Every data point in the figure corresponds to a transfer of a fixed data payload of 512KiB. The x-axis indicates the number of fragments used to transfer the fixed payload. In this work, the term ‘fragments’ corresponds to the number of explicitly initiated OpenSHMEM PUT operations - a user controllable parameter. It must be noted, however, that the data payload may be further split into smaller packets by the underlying software and hardware stack.

The effect of frequency scaling was studied in terms of the impact on 6 different metrics:

1) The energy consumed by the CPU servicing the sender process
2) The energy consumed by the CPU servicing the receiver process
3) The energy consumed by the DRAM servicing the sender process
4) The energy consumed by the DRAM servicing the receiver process
5) The unidirectional point-to-point latency (as measured at the sender’s side)
6) The Energy Delay Product (EDP)

The impact $I$ may be represented as the percent reduction in the above metrics $M$ due to application of a DVFS technique $T$ that scales down the frequency of the receiver

While CMOS circuits have the ability to trade performance for energy savings, it becomes challenging to optimize for both simultaneously. The EDP, first proposed by Horowitz[14], [15], 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.
Table III: Characteristics of the Test Platform

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Xeon CPU E5-2670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microarchitecture</td>
<td>Intel’s Sandy Bridge</td>
</tr>
<tr>
<td>L3 cache per die</td>
<td>20MiB</td>
</tr>
<tr>
<td>Cores</td>
<td>2x 8</td>
</tr>
<tr>
<td>Main Memory</td>
<td>32GiB</td>
</tr>
<tr>
<td>Infiniband card</td>
<td>Mellanox MT27500, ConnectX-3</td>
</tr>
<tr>
<td>Linux kernel version</td>
<td>2.6.32 x86_64</td>
</tr>
</tbody>
</table>

From an operating frequency $F_{initial}$ to a reduced frequency, $F_{final}$. It can be calculated as follows:

$$I = \frac{M(F_{initial}) - M(F_{final})}{F_{initial}} \times 100$$

From the expression above, it must be noted that a negative or positive impact value of $I$ suggests a corresponding rise or drop in a metric $M$ due to application of $T$. A zero value indicates an absence of any impact on $M$. The impact on each metric is illustrated in Figures 2, 3, and 4.

B. Test-bed Characteristics

Our test platform comprised of two Sandy Bridge nodes connected via InfiniBand. The characteristics of these nodes are listed in Table III.

C. Power/Energy Measurement

Each node contains power monitoring support and reports energy/power readings at the CPU, DRAM, and the node level. Each node has instrumented voltage regulators (VRs) that are sampled at a frequency of 1 kHz for both sockets and the four voltage lanes of the DIMMs (Dual In-line Memory Modules) on board. With the help of an FPGA, a digital filter is applied to smooth the samples. Furthermore, a linear correction is applied to the measurement data coming from the VRs in order to ensure an error margin not exceeding 3%. Our study was aimed at performing a fine-grained analysis of the impact on two main components that dictate the energy and power consumption of a system - the CPU and the DRAM.

VII. RESULTS

This section discusses the empirical results obtained on scaling down the operating frequency of the CPU servicing the receiver process. The impact is discussed for each of the implementation approaches depicted in Table II.

A. No Participation by the Receiver

- For the sender process, Figures 2-a/b suggest that there is no significant impact (≈ 0%) of frequency scaling on the energy consumption by the CPU and DRAM. This is true regardless of the extent of fragmentation (number of discrete buffers) used for transferring the data payload (Phase A) and ensuring its completion (Phase B). This may be attributed to the fact that the latency of the transfer is dictated by the bandwidth of the network interconnect (InfiniBand, in this case) which is orders of magnitude smaller than that of the I/O interconnect between the network adapter and the last-level (L3) cache on the receiver’s side. Since scaling down the frequency of the receiver CPU does not affect the bandwidth of the network interconnect, there is no significant impact during the actual RDMA-based transfers (Phases A and B).

- For the receiver CPU, Figure 2-c indicates that definite energy savings can be achieved due to scaling down of the operating frequency. Also, these savings are higher with a greater drop in the frequency (≈ 50% versus 68% when the frequency is scaled down from 2.901GHz to 2.4GHz and 1.2GHz respectively). This holds true regardless of the number of discrete fragments being transferred. This is not surprising because the CPU at the receiver’s end does not contribute to the data transfer operation and therefore the savings can be attributed to the reduced rate of polling at the synchronization point (Phase D).

- Figure 2-d shows that the energy savings for the receiver DRAM is almost constant (≈ 50%) regardless of the frequency level to which the CPU is scaled down to. It must be noted that due to Intel’s direct-I/O technology[10], there is almost no participation by the receiver DRAM during this transfer; the contents of the data transfer is directed to the L3-cache without the need for accessing the DRAM. Nevertheless we see significant savings in its energy consumption when the CPU frequency is scaled down. The fact that the savings is non-zero and independent of the final frequency suggests that the energy consumed by the DRAM is higher when the CPU operates at the turbo-frequency (2.901GHz) and is almost constant at other lower non-turbo frequency levels. Another observation is that for high fragmentation count (number of PUTs > 32K), there is a drop in energy savings. This suggests a rise in memory accesses for higher fragmentation. This is because, despite the use of the L3-cache described above, Intel’s chipsets limits the use of this cache up to 10% of its size - which, on our platform is about 2MB. With a rise in fragmentation (and hence, smaller sized PUTs), the relative overhead per network packet increases. This may be the potential cause for the L3-cache limit getting exhausted thereby leading to direct-I/O operations that target the DRAM.

- Figure 2-e highlights one of the major observations of this experiment. It affirms the fact that reducing the CPU frequency of the receiver in case of a one-sided transfer with no participation by the receiver CPU leads to no impact on the latency of the data transfer pattern.

- In terms of the net impact on the two-node system, we...
Figure 2: Impact of frequency scaling on energy and performance metrics for implementations which do not require active participation by the receiver during a one-sided point-to-point remote PUT operation. The line-chart and the pseudo-code of this approach is depicted in Table II-[A].
Figure 3: Impact of frequency scaling on energy and performance metrics for implementations which depend on active participation by the receiver in order to ensure completion of one-sided point-to-point remote PUT operation. The line-chart and the pseudo-code of this approach is depicted in Table II-[B].
Figure 4: Impact of frequency scaling on energy and performance metrics for implementations which relies on an additional asynchronous software agent to ensure completion of one-sided point-to-point remote PUT operation. The line-chart and the pseudo-code of this approach is depicted in Table II-[C].
see that the energy savings at the CPU servicing the receiver dominates the savings in Energy Delay Product (EDP) (Figure 2-f).

B. Active Participation by the Receiver

Figure 3 depicts the impact of frequency scaling on a remote PUT operation that involves aggregation of discrete user buffers by the sender and the corresponding unpacking by the receiver. It can be observed that the impact of this pattern is significantly different from that discussed in Figure 2.

- From Table II-B, we see that the time spent by the sender CPU within Phase C is dependent on the performance of the receiver in Phase E. From Figure 3-a, we observe that the energy consumption by the sender CPU is dependent on the frequency to which the receiver CPU is scaled down to. During Phase C, the sender CPU is primarily involved in a polling operation. As a result, the energy consumption is directly proportional to the time spent in this phase, which in turn is dependent on the frequency of the receiver. This explains the relatively higher (negative) impact on the receiver CPU energy during Phase-C ($\approx 0 \text{ to } (\sim 20\%)$ versus $\sim (75\%)$ to $\sim (98\%)$ when the frequency is scaled down from 2.901GHz to 2.4GHz and 1.2GHz respectively).

- Figure 3-b suggests that there is a negative impact on the receiver’s DRAM (rise in the energy consumption) when the receiver is operated at 1.2GHz. In this communication pattern, the two phases during which the receiver’s DRAM participates are phases F and B. From Table I, neither of these phases have the potential of being affected by scaling down the frequency of the receiver. Therefore, we do not completely understand the cause for the rise in energy. We are currently performing additional experiments to understand this behavior. It must be noted that this does not affect further analyses of this pattern simply because the magnitude of the DRAM energy here, is of the order of tens of milliseconds, which is negligible in comparison to that of the CPU (with energy consumption that is higher by two orders of magnitude).

- Figure 3-c shows that there is a definite rise in the energy savings of the receiver’s CPU, due to scaling down its frequency. It is important to note that regardless of the frequency down to which the CPU is scaled, the impact remains almost equal. This suggests that as long as the CPU is not operating in turbo frequency (2.901GHz), consistent energy savings ($\approx (50\%)$) can be expected.

- Figure 3-d shows that the impact of energy consumption by the receiver DRAM is dependent on the CPU frequency scaling. We observe that reducing the frequency from turbo (2.901GHz) to 2.4GHz has a positive impact on memory access rate. This leads to energy savings for the DRAM ($\approx (25\%)$). However, dropping the frequency to 1.2GHz drops the memory access rate to a point that leads to an energy inefficient transfer of the same size of data payload ($\approx (\sim 30\%)$).

- The unidirectional latency of this approach appears to follow a similar trend to that of the sender’s CPU energy: From Figure 3-e, the extent of impact on the latency is dependent on the frequency to which the receiver’s CPU is scaled down from 2.901GHz to 2.4GHz and 1.2GHz respectively.

- In terms of impact on the Energy Delay Product (EDP) due to frequency scaling, Figure 3-f shows that the behavior is dictated by the impact on the receiver’s DRAM. We see that reducing the receiver CPU frequency from 2.901GHz to 2.4GHz leads to positive savings ($\approx (10\%)$). However, reducing the frequency to 1.2GHz leads to negative impact as high as 90%.

C. Additional Thread Supporting the Receiver

Figure 4 depicts the impact of frequency scaling on a remote PUT operation that is completed with the assistance of an additional thread coupled with the receiver. This implementation approach suffers from the design of CPUs with compute cores sharing the same voltage plane. In this study, this architecture characteristic was true for the target SandyBridge processors.

- In order to decrease the frequency of the SandyBridge core servicing the receiver process, all the cores on the same voltage plane have to be scaled down. As seen in Figure 4-e, this impacts the unidirectional latency of the transfer process. More specifically, when the frequency of the receiver CPU is scaled down from 2.901GHz to 1.2GHz, the latency increases by up to 50% (negative impact). However, the impact is different when the CPU is scaled to 2.4GHz instead. In fact, it is observed that there is either zero or up to 20% drop (positive impact) in the latency. The variability in the impact may be attributed to the fact that there is a trade-off between the energy costs associated with two different phases at the receiver’s end - (a) The polling operation by the receiver process (Phase-D) (b) The memory management by the support thread (Phase-G). On reducing the frequency to 2.4GHz, the energy savings during Phase-D dictates that of the entire CPU. However, reducing the frequency to 1.2GHz significantly impacts the performance of the support thread which makes Phase-G contribute strongly to a rise in the latency (negative impact).

- Similar to the latency, the energy consumption by the sender’s CPU also varies based on the operating frequency of the receiver CPU (Figure 4-a). Since the frequency of this CPU is not altered, the similarity in the energy and latency characteristics may be attributed
to the CPU time invested to synchronize with the receiver (Phase-B).

- **Figure 3-f** summarizes the net impact of frequency scaling in terms of the achievable Energy Delay Product (EDP). Up to 64 PUTs, the EDP of the data transfer at 2.4GHz is higher by 50% than at 2.901GHz. Beyond 256 PUTs, the impact is almost negligible. However, while on dropping the frequency to 1.2GHz, there is a significant performance degradation or 50% or higher.

**Using DVFS in a multicore environment**

The scope of this paper is limited to evaluating the performance/energy metrics of RDMA operations between two processes, each bound to a single CPU core on different nodes. For the first two implementation approaches, the study was restricted to studying the impact of only these two cores participating in the data transfer operation. However, in real-world multicore HPC environment, it is almost always the case that a process running on one core is accompanied by additional software agents (OS processes / threads) running on sibling cores. In such an environment, using DVFS on a single core has a potential of affecting the activity of other cores.

The feasibility of using DVFS in a multicore environment is heavily dependent on its design and architecture of the target processor. Consider the case of Sandy Bridge processors. In this case, all the CPU cores lie on the same voltage/frequency plane[8]. In other words, all the cores operate at the same frequency level. Thus, a naive energy efficient solution of operating all the frequencies at a lower frequency in order to favor a single core might lead to significant performance degradation of software agents operating on other cores. This is illustrated in Figure 5(b). This issue may be alleviated in case of processors like the Haswell series where each core can be operated at a voltage/frequency setting which is independent of other sibling cores[16]. This approach has the potential of alleviating the performance impact of implementations which rely on using an additional thread to handle data transfers. This is shown in Figure 5(c).

**VIII. CONCLUSION AND FUTURE WORK**

This work details the impact of CPU frequency scaling on the performance and energy consumption of remote data transfers. The empirical results presented are instructional for system developers of energy efficient solutions for distributed memory programming models, especially PGAS. The focus was to analyze the impact of using DVFS (Dynamic Voltage Frequency Scaling) on the performance and energy metrics of system components servicing one-sided RDMA operations. Multiple cost factors that affect the energy and performance during DVFS-based techniques were identified. These factors are dependent on not only software stack but also various factors.
microarchitectural design factors. Their impact was analyzed with respect to three common implementation approaches of PGAS point-to-point communication - (a) Using RDMA capable underlying software and hardware stack to service transfers without the active participation of a target process, (b) Relying on the receiver process to participate in the data transfer to ensure its completion, and (c) Using an additional software agent (e.g. OS thread) at the receiver’s side to assist in completion of the operation.

The main lessons learned using empirical analysis of each of the above approaches are listed below:

- High energy savings with negligible performance impact may be achieved when the target process of a remote PUT operation does not participate in servicing the data transfer. This is applicable for implementation approaches that rely on RDMA-based capabilities of the underlying interconnect.
- For an implementation where the target process does participate in a data transfer operation, scaling down the frequency of that process not only affects the unidirectional latency but also the energy consumption, which worsens with a drop in the CPU operating frequency.
- For an implementation using an additional software agent for servicing a transfer, the extent of impact depends on: (a) The number of explicit PUT operations used to transfer the data payload, (b) The actual operating frequency of the CPU servicing the receiver, and (c) Architectural design of the target CPU in terms of whether multiple cores on a CPU share the same voltage plane or not.

Based on the challenges discussed, the next stage of this work will be directed towards exploring PGAS implementation approaches on multicore platforms characterized with CPU cores lying on independent voltage plane / frequency domain. This would enable tuning different phases of SPMD processes to different core frequencies.

IX. ACKNOWLEDGMENTS

Thanks are due to the Center for Information Services and High Performance Computing (ZIH) at the Technische Universität Dresden (TUD) for their support in terms of computational resources and their energy monitoring capabilities. This work was supported by the United States Department of Defense (DoD) and used resources of the DoD-HPC Program at the Oak Ridge National Laboratory and at the Oak Ridge Leadership Computing Facility. Thanks are also due to Robert Schöne (TUD), Tony Curtis (UH), Dounia Khaldi (UH), and the anonymous reviewers for their useful critiques and feedback.

REFERENCES