

SERA-IO: Integrating Energy Consciousness into Parallel I/O Middleware

Rong Ge Xizhou Feng

Department of Mathematics, Statistics and Computer Science
Marquette University, Milwaukee, WI
rong.ge@marquette.edu xizhou.feng@marquette.edu

Xian-He Sun

Department of Computer Science
Illinois Institute of Technology, Chicago, IL
sun@iit.edu

Abstract—Improving energy efficiency is a primary concern in high performance computing system design. Because I/O accesses account for a large portion of the execution time for data intensive applications, energy-aware parallel I/O subsystems are critical for addressing challenges related to HPC energy efficiency. In this paper, we present an energy-conscious parallel I/O middleware approach that combines runtime I/O access interception and Dynamic Voltage and Frequency Scaling capability available on modern processors to intelligently schedule the system’s power-performance mode for energy savings. We implement this approach into *SERA-IO*, an MPI-IO based middleware to enable energy consciousness for I/O intensive applications. Experimental evaluations conducted on real systems using multiple parallel I/O benchmarks show that *SERA-IO* can reduce system energy by 9% to 28% without decreasing application performance. With the emerging of large-scale data intensive applications and ever larger and more complex parallel computing systems, intelligent, energy conscious software and runtime systems such as *SERA-IO* are critical for the success of future high-end computing.

Keywords—parallel I/O, energy-aware computing, power management

I. INTRODUCTION

As large computing systems consume tens of megawatts of power, improving energy efficiency becomes a primary concern in high performance computing system design. For example, without disruptive technology, an exascale computer consisting of ~100 million processor cores would require close to 100 megawatts of power [15]. Such enormous power consumption not only incurs high energy cost but also limits system scalability and sustainability. To keep the power budget of future HPC systems at a reasonable level (e.g. ~20MW for an exascale computer), in addition to energy-efficient hardware, energy-conscious and intelligent runtime system and software are also necessities.

Energy-conscious I/O subsystems are critical for developing energy efficient HPC for two main reasons. First, the ever increasing gap between computation speed and I/O speed is transforming traditional CPU intensive applications into I/O-bound workloads. It is common that I/O access time contributes a large portion of the total execution time of applications, especially for those that routinely process ever-growing, complex data set. Second, during I/O access phases, the computing systems perform little computation but consume disproportional, enormous power. As scientific

applications become increasingly data-intensive, the need of energy-conscious I/O subsystems becomes even more pressing.

In this work, we present a novel, energy-conscious parallel I/O approach that complements existing work on energy efficient I/O subsystems. This approach relies on the dynamic voltage and frequency scaling (DVFS) technology available on today’s microprocessors, and integrates DVFS control at the parallel I/O middleware layer to reduce energy consumption of parallel computer systems. Particularly, this approach exploits two facts in parallel applications: (1) during I/O access phases, the full computation capability of the compute nodes is not required, and thus processors can run at low performance/power modes for energy savings; (2) many parallel applications use parallel I/O middleware (e.g., *MPI-IO*) for high performance data access and software portability, and their I/O patterns can be accurately captured by the parallel I/O library.

Numerous previous studies have used DVFS to save energy for parallel system [13], [26], [16], [18]. Most of them are tailored for communication and memory intensive applications. A few DVFS schedulers [12], [8], [14] are implemented at system level and therefore work for I/O intensive applications. However, they normally don’t make effective use of application information to achieve maximum energy savings **and** best application performance. To our best knowledge, this work is the *first* to explore energy-conscious approach and parallel I/O technology for I/O intensive parallel applications.

Here, we use the word *energy-conscious* to emphasize energy savings without performance loss. Unlike existing DVFS schedulers such as the on-demand governor included in the Linux kernel, energy-conscious schedulers aim to maximize energy savings with guaranteed performance. There are several inherent challenges in designing energy-conscious schedulers for real systems. First, power mode transitions incur overhead because processors can’t compute but still consume energy during the transitions. Frequent transitions for short periods of time hurt performance or even consume more energy. Second, scheduling must be done timely and properly. Switching processors to a wrong mode or at a wrong time misses energy saving opportunities or hurts performance, or both.

Our contributions: In this work, we introduce *SERA-IO*, a System Energy Reduction Agent for parallel I/O workload, which integrates energy consciousness into the middleware layer of parallel I/O subsystems. Combining runtime I/O access interception and accurate knowledge of the energy requirement of the I/O accesses, *SERA-IO* optimally schedules the processors’s power/performance modes (e.g., frequency) in time, and transparently and automatically saves energy without impacting application performance. We compare *SERA-IO* with several DVFS scheduling strategies. The experimental results with typical I/O intensive parallel applications on real clusters show that *SERA-IO* achieves up to 28% total system energy savings and still maintains best performance for these applications. This finding demonstrates energy-conscious software is a promising path to improving the energy efficiency of high performance computing systems.

We organize this paper as follows. After reviewing related work in section II, we describe the design of *SERA-IO* in section III, followed by experimental setup IV and results analysis V. Finally, we provide a summary in section VI.

II. RELATED WORK

Improving energy efficiency of HPC systems has been an active research topic in the past several years. Recognizing that I/O access is a leading performance bottleneck in HPC, this work mainly focuses on parallel I/O energy efficiency.

As a basic building block, energy efficient storage is critical for the overall system energy efficiency. In the past, researchers study using multi-speed disks [5], [10], [27], disk accesses coalescing [5], [10], [27], and write buffer disks [19] to reduce disk energy consumption. Recently, using low power non-volatile flash memory to improve energy efficiency of parallel disk systems is also investigated [17]. In general, these technologies reduce the energy consumption at the storage system side. Most of them don’t consider energy saving opportunities in compute nodes, which are the dominant power consumers in HPC systems.

Optimizing parallel I/O performance also results in improved I/O energy efficiency. Commonly used techniques include caching, prefetching [24], collective I/O, data sieving [23], and file system optimization [4]. These techniques either overlap the I/O access with computation, trade more network communications for less I/O requests, or provide concurrent I/O accesses and high bandwidth for I/O requests. Due to algorithm complexity and hardware capacity, completely hiding the I/O access latency is a nontrivial or impossible mission, especially for applications that routinely access large amounts of data.

The energy-aware parallel I/O approach described in this paper is complementary to I/O performance optimization. It can be applied to both un-optimized and optimized I/O accesses. This approach explores energy saving potentials on compute nodes with certain power-aware components.

Energy-aware parallel I/O also augments existing power-aware HPC research that improves system energy efficiency using power-aware components including processors [8], [18], memory modules [6], hard drive [27], and network devices [22].

Numerous DVFS scheduling methods have been developed for HPC systems. Though these methods can be implemented at different software levels, most of them explore the CPU slackness existing in communication or memory bound applications. For example, several researchers have studied compiler approach for detecting memory bound regions and inserting DVFS control into these regions [13], [26]. Previous work also investigate intercepting the MPI calls to locate CPU slackness for communication bound applications [16], [18]. Several DVFS schedulers can exploit all possible energy saving opportunities at system level [8], [14] for all types of applications. However, without application-specific knowledge of boundaries of CPU intensive phases and non-CPU intensive phases, it is difficult for the system-level schedulers to maximize energy savings. Different from existing work, this work focuses on parallel I/O workload and explicitly uses application information for scheduling decisions. Shang et al [20] also study data intensive programs. However, their algorithm is for sequential applications. It depends on local disk I/O activities for DVFS scheduling decisions and is not applicable to parallel applications using parallel I/O.

III. SERA-IO DESIGN AND IMPLEMENTATION

A. System Design

We design *SERA-IO* to meet three objectives. First, it improves the energy efficiency for I/O and data intensive parallel applications without losing performance. Second, it is portable across different platforms. And third, it conserves energy transparently to application developers without requiring any code change in application programs. To achieve these goals, we choose to implement *SERA-IO* at the middleware layer on the parallel I/O software stack and

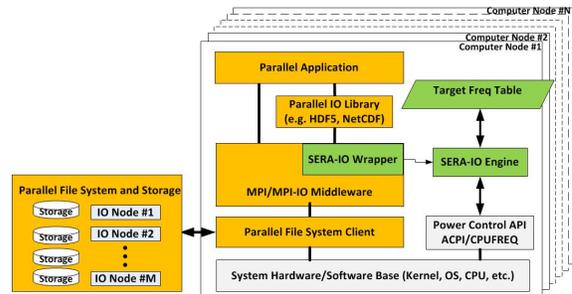


Figure 1. The system diagram of *SERA-IO*. It consists of three main components (in green): *SERA-IO* wrapper, *SERA-IO* engine, and target frequency table. The yellow blocks are parallel I/O software layers, and the white blocks are native OS system and hardware on the compute nodes.

leverage DVFS technology to reduce energy consumption of parallel I/O phases during applications' execution.

As shown in Figure 1, *SERA-I/O* resides on the compute nodes of clustered HPC systems. It consists of three main components: a *SERA-I/O* wrapper, a *SERA-I/O* engine, and a target frequency table. Both the wrapper and engine are packaged into the `libSERAIO.a` module. The target frequency table is provided as a configuration file.

SERA-I/O works as follows. During its installation, the target frequency table is constructed by profiling the performance and energy consumption of a set of parallel I/O micro benchmarks with available processor frequencies on the compute nodes. Once the table is available, *SERA-I/O* is ready for use. To make a parallel program *SERA-I/O* enabled, users **only** need to link the program to `libSERAIO.a` during its compilation. At runtime, the *SERA-I/O* wrapper intercepts the parallel I/O function calls and invokes *SERA-I/O* engine before and after parallel I/O accesses. *SERA-I/O* engine analyzes the I/O access pattern, looks up the target frequency for the intercepted I/O access in the table, and applies DVFS control via the ACPI/CPUFREQ interface provided by the Linux kernel.

1) *SERA-I/O Wrapper*: The wrapper is integrated into the parallel I/O middleware layer on the parallel I/O software stack. Currently, *SERA-I/O* is based on MPI-IO, which is defined as part of the MPI-2 standard and is a de facto standard for writing parallel I/O libraries and applications.

The wrapper intercepts the MPI-IO calls initiated by the MPI processes. By interception, the wrapper knows exactly when a parallel I/O phase starts and ends. Meanwhile, the wrapper passes the information about I/O access to the *SERA-I/O* engine. With the timing and I/O access pattern information, *SERA-I/O* can power down the processors just at the start of I/O operations and restore to the default (the highest) power mode for computation right after the I/O accesses finish.

The wrapper uses MPI profiling interface (*PMPI*) to intercept MPI-IO function calls and wraps them within *SERA-I/O* functions. All *SERA-I/O* wrapper functions share a common structure. Take *MPI_File_read_at_all* as an example, its corresponding wrapper function is as follows:

```
int MPI_File_read_at_all(...)
{
    int ret_val;
    SERA_Io_begin(...);
    ret_val = PMPI_File_read_at_all(...);
    SERA_Io_end(...);
    return ret_val;
}
```

Here, *PMPI_File_read_at_all* invokes the actual MPI-IO implementation for collective file reading. Both *SERA_Io_begin* and *SERA_Io_end* are implemented in the *SERA-I/O* engine. The former informs the start of an I/O

phase and its access pattern, and the latter signals the end of the current I/O operation.

2) *SERA-I/O Engine*: *SERA-I/O* engine decides how to schedule the power/performance modes for the processors with which the MPI processes are associated. The engine can be implemented in multiple ways. In this work, we choose a table lookup approach. After receiving a parallel I/O function name and parameters in the *SERA_Io_begin* function call, the engine parses the information to determine the access pattern, looks up the target frequency table for the target frequency, and then applies a DVFS schedule command through the CPUFREQ interface. One exception is: if the returned target frequency by looking up the table is the highest frequency available on the processors, *SERA-I/O* skips the last step and doesn't perform DVFS schedule since the current mode and the target mode are the same.

Because *SERA-I/O* schedules the power/performance modes of processors that execute the MPI processes, it is important to ensure that the scheduling decisions always apply to the correct physical cores. In this work, we enforce hard CPU affinity for both *SERA-I/O* enabled applications and normal applications to bind each MPI process to a fixed core during MPI initialization. We observed that hard CPU affinity normally leads to better performance than the natural CPU affinity included in Linux kernel 2.6 as the default process scheduler. The natural affinity attempts to keep a process on the same core during the execution. However, it doesn't prevent process migrating from one core to another.

3) *Target Frequency Table*: *SERA-I/O* relies on the target frequency table to store the target frequencies for various I/O access patterns. Currently the table is constructed offline and can be expanded gradually. In the future, we will consider online learning approaches to refine the table.

The table implements a map function $M : A \mapsto F$, mapping an I/O access pattern $a \in A$ to a frequency $f \in F$. Here A is a set of possible I/O access patterns and F is the set of frequencies available on the processors on the compute nodes in the platform. This table is looked up by the *SERA-I/O* engine at runtime. If an I/O access is not found in the table, then the highest available frequency is chosen for the target frequency.

B. Table Construction

Given a specific I/O access, its target frequency should maximize energy savings with zero or minimum performance loss. Two major factors must be taken into consideration in the table construction to avoid performance loss and energy waste. The first factor is the performance penalty of running at a lower frequency for CPU activities involved in the I/O accesses. The second factor is mode transition overhead, especially when frequent transitions undertake in a short period of time.

1) *I/O Access Pattern*: Our earlier studies [7] showed that for scientific computing, I/O access times are strongly

correlated with I/O access patterns. Currently, we characterize access patterns with 5 attributes listed in Table I: operation type, data size, locality, collective/individual, blocking/nonblocking. All these attributes can be extracted from the MPI-IO function name and parameters. For example, from the function `MPI_File_read_all(fh, buf, size, MPI_INT, &status)`, we can extract its I/O access pattern as follows.

```

Operation Type    = Read
Data Size        = sizeof(MPI_INT) * size
Spatial locality  = Contiguous
Collective?      = YES
Blocking?        = YES

```

Table I
THE ATTRIBUTES OF I/O ACCESS PATTERNS.

Attribute	Value	Extraction
Operation Type	Read Write Seek Open Close	Function Name
Data Size	Number of bytes	Data Type, Count
Spatial Locality	Contiguous Strided	Data Type
Collective?	YES NO	Function Name
Blocking?	YES NO	Function Name

2) *Table Fields*: The target frequency table is organized as a list of entries, each comprising an access pattern and a target frequency. The first fields of the entry match the attributes of an I/O access pattern and the last field is the target frequency for the pattern. Table II provides a simple example of the target frequency table. In this table, the field *Data Size* is coded as a range with a lower bound and an upper bound, instead of a single value.

When *SERA-IO* looks up the table for a given pattern, the engine first lists all entries matching the attributes for the given pattern except data size. Then it finds the entry whose data size field contains the data size of given pattern. If one table entry is found, the recorded frequency in the entry is returned as the target scheduling frequency. Otherwise, the default high frequency will be returned.

Table II
A SAMPLE TARGET FREQUENCY TABLE

OpType	DataSize	Locality	Coll?	Block?	Freq.
read	(1MB, 2MB)	contig	NO	YES	1.8GHZ
read	(64MB, 128GB)	noncontig	YES	YES	1.3GHZ
write	(1MB, 2MB)	contig	NO	NO	2.5GHZ

3) *Target Frequency Identification*: Currently, we identify the target frequencies of various I/O access patterns using a profiling-based approach. Specifically, we run a set of micro benchmarks to enumerate a set of I/O access patterns of interest, collect the time, power and energy profiles of these patterns for each available frequency on the compute nodes, and then choose the most efficient frequency satisfying the performance constraint as the target frequency. In this study, we use PIO-Bench [21] to generate the parallel I/O access

patterns. We instrument the PIO-Bench code to integrate with the PowerPack [9] toolkit for power-energy profiling.

Table III shows the profiles of two example patterns. For each access pattern, we identify its target frequency with two steps. In the first step, we filter short accesses that last less than a time threshold. We take this step because for short I/O accesses (type I), using a low frequency won't save much energy but incur extra transition overhead. Therefore, we use the highest available processor frequency for the target frequency for these types of patterns. The second step only applies to long I/O accesses (type II). For those accesses, we evaluate the bandwidth and average power for all available frequencies, and identify the frequency that results in the smallest EDP value as the target frequency. For the type II example shown in Table III, the target frequency is set as 1.3GHz as it gives the smallest EDP value.

Table III
SAMPLE PROFILING RESULTS USING PIOBENCH AND POWERPACK.

(a) A type I example: collective, noncontiguous, blocking read with data size 1KB

Freq	Time	Bandwidth	Average Nodal Power
1.3GHz	0.11ms	9.41MB/s	159W
2.5GHz	0.10ms	9.76MB/s	200W

(b) A type II example: collective, noncontiguous, blocking read with data size 2MB

Freq	Time	Bandwidth	Average Nodal Power
0.8GHz	17.1ms	125.43MB/s	141W
1.3GHz	16.4ms	127.88MB/s	152W
2.5GHz	16.7ms	126.13MB/s	183W

IV. EXPERIMENTAL SETUP

A. Experimental Environment

The experiments are conducted on a 9-node power-aware cluster with Gigabit Ethernet. Each node in the cluster has dual AMD Opteron quad-core 2380 processors running Linux. Each core in a node can be independently scheduled among four frequencies: 0.8GHz, 1.3GHz, 1.8GHz, and 2.5GHz. Each core has a 64KB L1 instruction cache, a 64KB L1 data cache, and a unified 512KB L2 cache. The four cores on the same chip share one 6MB L3 cache.

Two file systems, NFS and PVFS2, are examined on this cluster. When NFS is used, the head node is used as the NFS server to provide a 1.4TB storage using three directly attached RAID disks configured with RAID-5. Each individual RAID disk is a WD7500AYPS Raid Edition 7200rpm SATA hard drive, which has 750GB capacity and 16MB data cache, and supports a maximum 3GB/s buffer to host transfer speed. When PVFS2 is used, unless explicitly stated, the storage is distributed over two IO nodes on the cluster, each with one 160GB Western Digital WD1600AYPS Raid Edition 7200rpm SATA hard drive.

The reported energy numbers in the results section are the total energy over all participating compute nodes for each

test. For tests (e.g., $NP = 36$) that do not use all the cores on one node, the power consumption of that node is prorated to the number of cores being used. For each compute node, we use a Wattsup meter to measure its system power, which is the total power consumed by CPUs, memory modules, disks, fans, and other components on the node. The power measurement is synchronized with the performance timing using PowerPack [9]. The energy consumption of a single node is calculated as the integral of the measured system power over the entire application execution time.

B. Parallel Benchmarks

In our experiments, we use four parallel I/O benchmarks: BT-IO [25], MADBench [3], FLASH-IO [1], and Tile-IO [2]. The first two benchmarks consist of interleaved computation and I/O phases, representing typical scientific workload. The other two involve only I/O phases. Both BT-IO and FLASH-IO are implemented in Fortran, and the other two are written in C. Among the four benchmarks, FLASH-IO relies on the high level I/O library HDF5 for parallel file write, while the others directly invokes MPI-IO functions.

V. RESULTS AND ANALYSIS

To evaluate the performance of SERA-IO, we compare the performance and energy of each benchmark under three scheduling strategies as shown in table IV.

Table IV
THE THREE SCHEDULING STRATEGIES AND THE CORRESPONDING SETTINGS OF POWER/PERFORMANCE MODES ON COMPUTER NODES.

Scheduling Strategy	Processor Frequency
High Performance (2500)	Fixed at 2.5 GHz
Low Power (800)	Fixed at 800 MHz
SERA-IO(SERA-IO)	Dynamic

The *High Performance* strategy fixes the frequency at 2500MHz, while the *Low Power* strategy fixes the frequency at 800MHz. These two strategies reflect system’s default high performance setting and low power setting respectively. To enable *SERA-IO* strategy, we link the applications to the *SERA-IO* library `libSERAIO.a`.

In addition, we compare the performance of *SERA-IO* with the *ondemand* governor included in the Linux kernel as part of the *cpufreq* module. The *ondemand* governor is a system wide DVFS scheduler which automatically scales down the frequency of processors with low utilization. System wide DVFS schedulers work best for system idle or long I/O periods, but usually do not perform consistently. For real scientific applications, a system level scheduler may increase both execution time and energy consumption.

A. SERA-IO Performance for Typical Applications

BT-IO. BT-IO tests parallel writing a 3D matrix data distributed among MPI processes. The code employs a

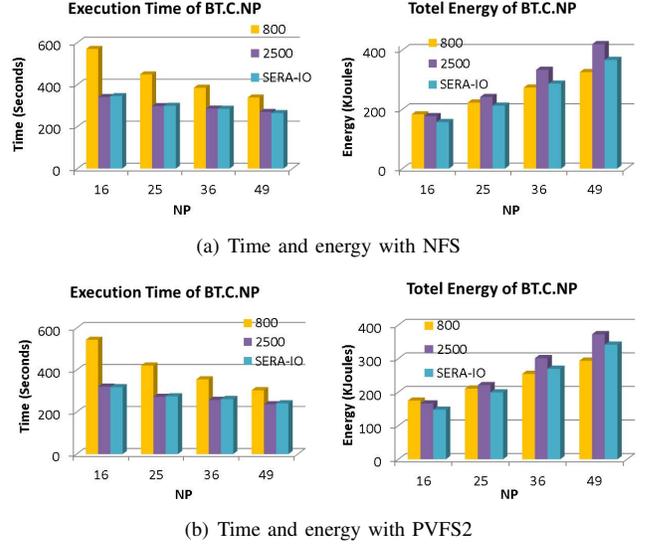


Figure 2. The execution time and energy consumption of BT-IO benchmark (class C problem size and full MPI I/O subtype) with NFS and PVFS2.

complex diagonal multi-partitioning domain decomposition and distributes its data set among MPI-processes. Each process is responsible for multiple Cartesian subsets that are noncontiguous in the entire data set. BT-IO writes the entire solution periodically to a file. In our study, we use BT-IO *full or collective* I/O subtype, which involves a single I/O access pattern characterized by blocking collective write with explicit offset.

Figures 2 show the execution time and energy consumption of BT-IO with problem size of class C and full MPI I/O subtype. Using the scheduling strategies described above, we run the tests on both NFS and PVFS2 file systems and vary the number of processes (NP) from 16 to 49. From the experimental results, we make the following observations.

1) Comparing with the *High Performance* strategy, *SERA-IO* reduces energy with nominal performance change for all the test cases. With NFS, the observed energy savings range from 11.3% to 14.0%. With PVFS2, the observed energy savings range from 9.1% to 12.5%. For all cases, the performance difference between *SERA-IO* strategy and *High Performance* strategy is within $\pm 2\%$.

2) Comparing with the *Low Power* strategy, *SERA-IO* always results in significantly higher performance. With NFS, *SERA-IO* enabled execution runs 39.4% to 21.8% faster when NP increases from 16 to 49. With PVFS2, the values become 41.4% and 21.5% respectively. The energy results are little complex. With NFS, *SERA-IO* enabled execution consumes 14.2% less energy when NP = 16 but 12.6% more energy when NP = 49. With PVFS2, the values are 15.5% and 16.2% respectively.

3) PVFS2 delivers better performance and consumes less energy than NFS. With *SERA-IO* enabled, BT-IO runs 8%

faster and consumes 6% less energy with PVFS2 than with NFS. This is because PVFS2 provides multiple data streams between storage devices to processes, and thus achieves a higher I/O bandwidth. We find that the performance-NP curve with each strategy doesn't vary with the file system, and so is the energy-NP curve. Because results with NFS are usually poorer and don't provide extra insights, in later sections, we only analyze the effects of *SERA-IO* under PVFS2 file system for other benchmarks.

***SERA-IO* versus ondemand scheduler:** Because most recent Linux distributions include an ondemand DVFS governor that automatically reduces power and energy when the computer system is either idle or under utilized, it is necessary to compare *SERA-IO* with the *ondemand* governor to justify why optimal DVFS scheduling requires application specific information. In Table V, we compare the experimental results of BT-IO benchmark under *SERA-IO* and ondemand governor for varying numbers of processors. From the results, we find that for all system sizes, both *SERA-IO* and *ondemand* governor result similar execution time for the benchmark. However, ondemand governor consistently consumes about 10-14% more energy than *SERA-IO*.

Table V
THE PERFORMANCE OF *SERA-IO* VERSUS ONDEMAND GOVERNOR FOR BT-IO BENCHMARK

NP	Execution Time (Seconds)		Energy (Joule)	
	<i>ondemand</i>	<i>SERA-IO</i>	<i>ondemand</i>	<i>SERA-IO</i>
16	324.15	319.11	167543.86	147721.60
25	273.47	273.51	214863.98	194699.28
36	261.76	259.48	300870.57	263224.92
49	240.48	240.92	377922.92	338341.08

To understand why *SERA-IO* performs better than the *ondemand* governor, we plot the power traces of BT.C.25 benchmark on a single compute node under each scheduler in Figure 3. From this figure, we draw two observations. First, *SERA-IO* can capture the I/O access phases and make accurate and timely decisions on power/performance mode transition. In contrast, *ondemand* normally delays setting and restoring the processor frequency. Second, the power drop under the *ondemand* governor is much smaller than the power drop under *SERA-IO*. This is because the *ondemand* governor makes DVFS decisions based on an average of most recent history data and is unable to know what exactly will happen at next time point. To conclude, we justify integrating application specific information into DVFS scheduler design, i.e., the approach represented by *SERA-IO* can lead to more efficient energy-conscious systems.

MADBench. MADBench is a stripped-down version of the Microwave Anisotropy Dataset Computational Analysis Package used by Cosmic Microwave Background studies. The main I/O pattern of MADBench is the concurrent contiguous read/write of the local subsection of the dataset on all processes with large buffer size. The IO mode MADBench

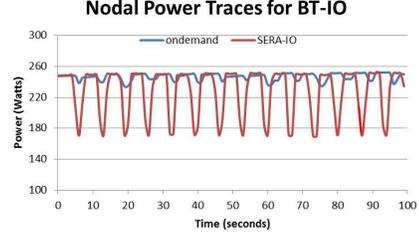


Figure 3. The power traces of BT.C.25 under ondemand and *SERA-IO* scheduler on a single node. The figure only shows data for the first 100 seconds.

benchmark consists of three phases: *S*, *W*, and *C*. Each phase has a different combination of computation and I/O operations. *S* involves write, *C* involves read, and *W* involves both read and write. In our study, all of these phases use blocking collective MPI IO functions to access a set of 2D matrices.

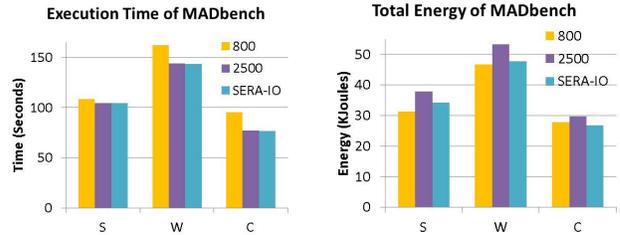


Figure 4. The execution time and energy consumption of MADBench on PVFS2 with the three scheduling strategies. $NP = 16$ is used for this test.

Figure 4 shows the execution times of MADbench benchmark running on 16 processes using PVFS2. In the test, each process accesses 0.5 GB of data in 16 iterations. At IO mode, MADbench replaces the actual computation with busy work and the users can adjust the total number of computations using a parameter called busy work exponent. In our experiments, this parameter is set to 1.2, indicating the computation intensity is between level 2 and level 3 BLAS.

From this figure, we have the following observations: 1) Comparing with the *High Performance* strategy, *SERA-IO* enabled execution consistently saves about 10% energy without increasing execution time for all three phases. 2) Comparing with the *Low Power* strategy, *SERA-IO* enabled execution consistently runs faster: 4.0% for phase *S*, 11.6% for phase *W*, and 19.3% for phase *C*. The higher performance of *SERA-IO* enabled execution sometimes comes at the expense of more energy.

These observations agree with the results of BT-IO benchmark. Both results demonstrate that for typical parallel applications, *High Performance* strategy is far from optimal in term of energy use, and *Low Power* strategy leads to significantly poor performance. The energy-aware strategy

is a promising solution which results in both optimal energy use and good performance.

B. SERA-IO Performance for Benchmarks Using Libraries

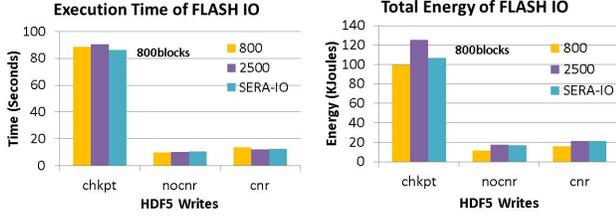


Figure 5. The execution time and energy consumption of FLASH IO with PVFS2 file systems. $NP = 64$ and $block\ size = 800$ in this test.

The FLASH-IO benchmark recreates the I/O pattern of the FLASH application that simulates the evolution of multiple physical quantities over time on a Cartesian, structured mesh. It produces one checkpoint file and two plot files for centered data and corner data respectively. The access pattern of the FLASH code is noncontiguous both in memory and in file. Compared with BT-IO and MADBench, FLASH-IO uses HDF5 libraries to handle the complex data model and doesn't directly calling MPI-IO functions. In addition, it has little computation. We use FLASH-IO to test the support of SERA-IO for such applications. Here, we show the time and energy of writing the checkpointing data (chkpt), the centered data (nocnr), and the corner data (cnr) with PVFS2 and $NP = 64$ in Figure 5. From this figure, we have the following observations.

1) Similar to previous benchmarks, SERA-IO enabled execution saves 14.5% energy comparing with the *High Performance* strategy for FLASH-IO. This confirms that SERA-IO can improve the energy efficiency of MPI-IO based IO library such as HDF5.

2) During all three I/O phases, different DVFS strategies do not make much difference on the execution time. This is because parallel checkpointing and plot file writing don't require much processor resources. Since the I/O access time is not strongly correlated with processor frequency, running at a lower frequency is preferred. More interesting, SERA-IO enabled FLASH-IO completes checkpointing operations slightly faster than normal FLASH-IO with the *High Performance* strategy. This result is counterintuitive but repeatable. We speculate this could be related to contentions on shared system resources but need further investigation.

In addition, these results provide hints for further improvement of SERA-IO. For FLASH-IO, the *Low Power* strategy is more energy efficient than SERA-IO with roughly the same performance. We identify two reasons for explanation. Firstly, the target frequency table in SERA-IO is conservatively built, meaning higher power modes than ideal

are often used to avoid performance penalty. Especially for the I/O access patterns that are not included in the table, the highest power mode is used. Secondly, when the FLASH-IO benchmark runs with 64 processes, there are some synchronization phases during which the *Low Power* strategy saves energy, while SERA-IO doesn't. As a part of future work, we will enhance SERA-IO to explore energy savings from both I/O accesses and synchronization.

C. SERA-IO Performance for Single I/O Phase Benchmarks

Tile-IO tests the performance of tiled data accesses that exist in many visualization and scientific applications. It partitions a big matrix into sub-matrices or blocks, and assigns each process to access one block. In this way, the spatial locality of I/O accesses is nested strided. Tile-IO has a single I/O phase dominated by collective read with minimal computation. Tile-IO maps a square matrix to a square processes and each process accesses a submatrix. In our experiments, we choose a data size that is sufficient large so that the processes can not hold all the data in memory to take advantage of prefetching and caching.

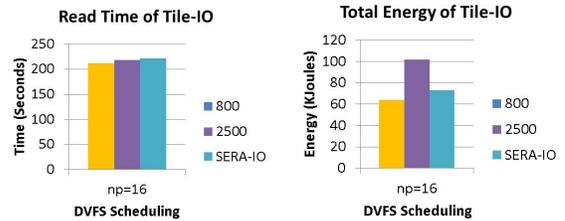


Figure 6. The execution time and total system energy of Tile IO with PVFS2 file systems.

Because Tile-IO only has a single I/O phase, it provides a base case to measure the maximum energy saving that SERA-IO can achieve. Figure 6 shows the execution time and energy of Tile-IO running on 16 processors using PVFS2. From the figure, we find that running Tile-IO at 800 MHz consumes 37% less energy than running it at 2500MHz. This is the upper bound that SERA-IO could save on our experimental cluster. In contrast, SERA-IO enabled Tile-IO execution results in 28% energy saving. This difference indicates a potential room to optimize SERA-IO.

On another perspective, for single I/O phased applications, end users may be able to use *Low Power* strategy to achieve highest energy efficiency. However, this requires the users have a deep understanding of both applications and systems, which usually involves performance profiling and code analysis. SERA-IO alleviates such burden for users who don't care profiling performance or understanding details of systems. They can link the programs to the SERA-IO library and rely on SERA-IO to save energy.

VI. SUMMARY

In addition to traditional data-intensive applications, scientific applications also become increasingly data-intensive due to new tools such as animation and data mining. In the meantime, data input and output speed improvement is at snail's speed compared to the computing speed improvement [11]. This leads to the I/O-wall problem. Intensive efforts have been made recently to improve I/O performance so that the CPU power can be utilized. In this study, however, we take a different approach. Recognizing I/O is often the performance bottleneck, we lower computing power during application's I/O intensive phases to save energy.

We design and evaluate *SERA-IO* as an energy-aware middleware. *SERA-IO* automatically schedules processor frequency based on I/O access patterns transparently to applications. It saves energy without impacting application performance. Experimental results on parallel I/O benchmarks and applications show a minimum of 9% and maximum of 28% energy savings, depending on the application's I/O intensity. As modern high-end computing systems become more power hungry and complex, and applications become more data intensive, energy-conscious middleware such as *SERA-IO* are becoming increasingly valuable, and will be a necessity for tomorrow's exascale computing systems.

An emerging trend in HPC is that data access has become the leading performance bottleneck of modern computing systems. I/O is only part of the data access hierarchy. In the future we plan to extend *SERA-IO* to explore other energy saving opportunities related to data access including communication, memory access and I/O access.

ACKNOWLEDGEMENTS

The authors would like to thank the National Science Foundation for sponsoring this work under grants NSF CNS-#1116691.

REFERENCES

- [1] FLASH I/O Benchmark. <http://flash.uchicago.edu/zingale/flashbenchmark.io/>.
- [2] Parallel I/O Benchmarking Consortium. <http://www.mcs.anl.gov/research/projects/pio-benchmark/>.
- [3] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas. Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms. In *International Conference on Parallel Processing 2005 (ICPP 2005)*, pages 119–128. IEEE.
- [4] P. Carns, W. Ligon III, R. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th annual Linux Showcase & Conference-Volume 4*, pages 28–28. USENIX Association, 2000.
- [5] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *the 17th International Conference on Supercomputing*, 2003.
- [6] X. Fan, C. Ellis, and A. Lebeck. The Synergy between Power-aware Memory Systems and Processor Voltage Scaling. *Power-Aware Computer Systems*, pages 151–166, 2005.
- [7] R. Ge. Evaluating Parallel I/O Energy Efficiency. In *IEEE/ACM Conference on Green Computing and Communications (GreenCom)*, pages 213–220, 2010.
- [8] R. Ge, X. Feng, W.-C. Feng, and K. W. Cameron. CPU MISER: a Performance-Directed, Run-Time System for Power-Aware Clusters. In *International Conference in Parallel Processing (ICPP) 2007*, Xian, China, 2007.
- [9] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron. PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications. *IEEE Transactions on Parallel and Distributed Systems*, 99(1), 2010.
- [10] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *the 30th Annual International Symposium on Computer Architecture*, page p. 169, San Diego, California, 2003.
- [11] J. Hennessy, D. Patterson, and D. Goldberg. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann, 2006.
- [12] C.-H. Hsu and W.-C. Feng. A Power-Aware Run-Time System for High-Performance Computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, 2005.
- [13] C.-H. Hsu and U. Kremer. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. In *PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 38–48, New York, NY, USA, 2003. ACM.
- [14] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO*, pages 359–370, 2006.
- [15] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, et al. Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. DARPA IPTO Workshop Report, Washington, DC, 2008.
- [16] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. MPI and Communication - Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *Proceedings of the ACM/IEEE Supercomputing 2006 (SC'06)*, 2006.
- [17] M. Nijim, A. Manzanares, X. Ruan, and X. Qin. HYBUD: An Energy-Efficient Architecture for Hybrid Parallel Disk Systems. *Computer Communications and Networks, International Conference on*, 0:1–6, 2009.
- [18] B. Rountree, D. Lownenthal, B. de Supinski, M. Schulz, V. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd international conference on Supercomputing*, pages 460–469. ACM, 2009.
- [19] X. Ruan, A. Manzanares, S. Yin, Z. Zong, and X. Qin. Performance Evaluation of Energy-Efficient Parallel I/O Systems with Write Buffer Disks. In *ICPP '09: Proceedings of the 2009 International Conference on Parallel Processing*, Washington, DC, USA, 2009.
- [20] P. Shang and J. Wang. A Novel Power Management for CMP Systems in Data-intensive Environment. In *2011 IEEE International Parallel & Distributed Processing Symposium*, pages 92–103. IEEE, 2011.
- [21] F. Shorter. Design and Analysis of a Performance Evaluation Standard for Parallel File Systems. MS in Computer Science, Clemson University, 2003.
- [22] V. Soteriou, N. Eislely, and L.-S. Peh. Software-directed Power-aware Interconnection Networks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(1):5, 2007.
- [23] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. *frontiers*, page 182, 1999.
- [24] N. Tran and D. Reed. Automatic ARIMA Time Series Modeling for Adaptive I/O Prefetching. *Parallel and Distributed Systems, IEEE Transactions on*, 15(4):362 – 377, april 2004.
- [25] P. Wong and R. Van der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical report, NAS Technical Report NAS-03-002, 2003.
- [26] Q. Wu, V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D. W. Clark. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *the 38th IEEE/ACM International Symposium on Microarchitecture*, pages 271–282, Barcelona, Spain, 2005.
- [27] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping Disk Array Sleep Through the Winter. In *the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, 2005.