5-2023

Green HPC: Optimizing Software Stack Energy Efficiency of Large Data Systems

Grant Wilkins

Green HPC: Optimizing Software Stack Energy Efficiency of Large Data Systems

A Honor's Thesis
Presented to
the Honors College of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Departmental Honors
Computer Engineering

by
Grant Wilkins
May 2023

Abstract

High-performance computing (HPC) is indispensable in modern scientific research and industry applications, but its energy consumption is a growing concern. This thesis presents two novel approaches to optimize energy consumption in large data systems. The first chapter of the thesis will discuss the use of Dynamic Voltage and Frequency Scaling (DVFS) to optimize the energy efficiency of two popular lossy compression algorithms: SZ and ZFP. By adjusting the voltage and frequency levels of computing resources, DVFS can reduce energy consumption while maintaining the desired level of performance and accuracy. The second chapter of the thesis will focus on a detailed comparison and analysis of asynchronous and synchronous checkpointing energy consumption using the VELOC and GenericIO libraries. The study investigates the trade-offs between these two checkpointing techniques, offering insights into their energy consumption patterns and performance impacts on large-scale HPC systems. Based on the analysis, we provide recommendations for choosing the most energy-efficient checkpointing method for specific application scenarios. Together, these two approaches contribute to the development of Green HPC, paving the way for more sustainable and energy-efficient large data systems. This thesis will provide valuable insights for researchers and industry practitioners aiming to optimize energy consumption while maintaining high-performance computing capabilities.

Acknowledgments

I would like to thank Dr. Jon Calhoun, for without his mentorship, this project and all other projects similar to it would not be possible.

Hurricane Isabel data produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR, and the U.S. National Science Foundation (NSF) This material is based upon work supported by the National Science Foundation under Grant No. SHF-1910197.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The dawn of the exascale computing era has brought unprecedented computational power and opportunities for scientific discovery, but it also presents significant challenges in terms of energy efficiency and sustainable computing. High-performance computing (HPC) systems have seen a tremendous increase in the number of components, resulting in higher power consumption and shorter mean times between failures. As data volumes generated by these systems reach tens of petabytes, efficient and energy-conscious data management techniques are more crucial than ever. The software stack, in particular, is a critical area where energy efficiency improvements can be achieved, as it governs the overall behavior and performance of the HPC system. This Honor's Thesis, titled "Green HPC: Optimizing the Energy Consumption of Large Data Systems," delves into two critical aspects of HPC: energy-efficient resilience strategies and power optimization in data compression, both targeting the software stack.

The first part of the thesis examines the energy efficiency trade-offs and the impact of various factors on the performance and scalability of checkpoint/restart (C/R) techniques within the software stack. As HPC systems continue to grow in scale, traditional resilience strategies such as C/R must evolve to become more energy efficient while minimizing runtime overheads. In this context, C/R techniques have advanced to employ asynchronous multi-level resilience methods that leverage heterogeneous storage. To provide a comprehensive understanding of energy efficiency in C/R, two state-of-the-art C/R libraries, VELOC and GenericIO, are studied in depth. The research reveals that asynchronous C/R provides significantly higher throughput while consuming considerably less energy compared to synchronous C/R. It is concluded that improving and maintaining high throughput should be prioritized over hardware throttling to effectively reduce energy consumption in exascale computing. Additionally, the study offers insights into how developers can create more energy-efficient C/R techniques to meet the ever-growing demands of exascale systems.

The second part of the thesis investigates energy consumption optimization for lossy data compressors and data writing in HPC systems, specifically targeting the software stack. As lossy compressors become indispensable for handling large data volumes due to their superior compression ratios and runtime performance, it is vital to optimize their energy consumption. Focusing on the SZ and ZFP lossy compressors, the study explores power consumption patterns while varying CPU frequency, scientific data sets, and system architecture on a cloud HPC system. A power model for lossy compression is developed, and a tuning methodology is proposed to reduce energy overhead by an average of 14.3%. The model is demonstrated to save 13% of energy on average for 512GB I/O, leading to more energy-efficient lossy data compression and I/O. Furthermore, the research offers valuable information on how to optimize compressor and data I/O power usage to contribute to sustainable computing goals and manage limited power budgets.

By addressing these two key areas within the software stack, this Honor's Thesis contributes to the ongoing efforts to make HPC systems greener and more sustainable, thus ensuring that the benefits of exascale computing are realized while minimizing the impact on the environment and staying within limited power budgets. The research highlights the importance of targeting software stack optimizations to achieve significant energy efficiency gains in HPC systems. Ultimately, this work demonstrates the potential of software-based solutions to address the energy challenges of large-scale computing and paves the way for future research in this critical domain.

Chapter 2

Modeling Power Consumption of Lossy Compressed I/O for Exascale HPC Systems [54]

2.1 Introduction

As high-performance computing (HPC) approaches exascale, computing centers consume significant amounts of power, on the scale of tens of MWs [20]. These HPC systems run highly complex applications that evolve large volumes of data. Scientific workloads now require moving data for analysis and processing, yet large-data I/O is a computationally expensive operation. For example, Hardware/Hybrid Accelerated Cosmology Code (HACC) [23], can generate enough snapshots to require 10 hours to transmit at a 500 GB/s bandwidth [29]. To mitigate I/O bottlenecks, applications use lossy compressors such as SZ [16,28,49,57] and ZFP [30] to compress floating point data, reduce the storage size, and save I/O time. Lossy compressors have the advantage of better space-savings and runtime efficiency over lossless compressors, making them more desirable for compressing petabytes of floating-point data.

However, compressing large datasets can require a non-trivial amount of time. Consider the situation of needing to write a large amount of data. For dumping data one can compress and then write the compressed data, or simply transmit without compressing. I/O runtime savings on data-dumping can be up to 25% with lossy compression, yet there are cases where the compression itself can outweigh the runtime for reading and writing the compressed data [29]. A smaller, compressed file transports faster, but how long it takes to compress that data can mitigate those advantages.

How does this affect the energy involved in the I/O? That is, if compression takes significant runtime, how does that affect how much energy is drawn for the I/O operation? Assuming single CPU compression and I/O, the CPU usage and clock frequency of the cores directly correlates with the energy-draw of that CPU [26]. Since power draw is a given for any system, our goal is then to minimize the compression and data transit energy usage by lowering the clock frequency of the chip.

The motivation for focusing on compression and I/O with respect to CPU frequency is that in HPC workloads, I/O is not as performance-sensitive as a simulation running. As one lowers CPU frequency, applications execute more slowly despite saving energy. When a user runs simulations, one needs the full CPU power as these jobs are computationally exhaustive and time can be limited due to competitive resource schedules. On the other hand, compression and I/O can afford a longer runtime in many use cases, as this data transit process is several orders of magnitude shorter than most scientific simulations. Therefore, finding where runtime and energy consumption are optimized via trade-off based on the CPU frequency is possible in the context of compression and data dumping.

In this paper, we measure the energy usage of SZ and ZFP compressing floating point data on a cloud HPC system with a network file system (NFS). Also we measure the energy usage of moving different-sizes data on an NFS. With these results we construct models of energy and average power that provide a CPU frequency tuning framework to optimize energy usage for lossy compression and data transmission, characterizing an I/O workflow. Using our models we demonstrate energy-savings in the use-case of lossy compression and then transmitting data.

Our contributions in this paper are as follows:

- We derive and compare power consumption models of SZ and ZFP to enable CPU-frequency tuning to find optimal power and energy consumption.
- We provide recommendations to reduce energy usage of lossy compression in I/O by 19.4% on average by lowering the CPU frequency by 12.5%.
- We construct a model for data transit power consumption and achieve reduced energy usage by 11.2% on average by lowering the CPU frequency by 15%.

- We utilize our frequency tuning models for energy-optimizing lossy compression and data transmission and demonstrate 14.3% overall energy savings.
- We apply our model to a real-world example of compressing and writing 512GBs of floating point data on an NFS, saving 6.5 kJs on average, which equates to 13% energy savings.

We organize our paper as follows. In Section II we describe related works to our project, as well as motivation concerning past studies in energy-efficiency, lossy compression, and data movement. In Section III we outline the different methods, softwares, and datasets we use in our experiments. In Sections IV, V we develop our model of power consumption for lossy compression and data transit, respectively. In Section VI we discuss how we optimize the power consumption of I/O using our models. Section VII then shows use-case driven examples of tuning an HPC system for energy-savings.

2.2 Related Work

2.2.1 Lossy Compression and Data Dumping

As HPC systems approach exascale, the question of how to transport data on cloud systems for analysis becomes important. Lossy compressors, such as SZ [16, 28, 49, 57] and ZFP [30] provide the advantage of decreasing size with impressive compression ratios. Lossy compressors achieve this by eliminating redundancies in chunks of scientific, floating-point data via encoding the binary representation of each datum. Based on the granularity required, an error bound can be set to preserve more or less data. These space savings make lossy compression a valuable tool in scientific computing, particularly in data movement.

Previous work [29] has shown the possibility for lossy compression to decrease transit time on an I/O-bound operation. Liang et al. focused on the different configurations of SZ and ZFP in the context of runtime optimization for transporting large datasets, finding time-savings when compressing before disk writing. A primary interest in data dumping is sending a small number of larger files over a large number of smaller files due to I/O bottlenecking in HPC systems [31]. Lossy compression on large scientific datasets achieves this by retaining the number of files and reducing their size, rather than increasing the number of files with smaller size. Using lossy compression on large scientific datasets, our study looks to take a similar approach, however taking energy savings in place of time-savings for transporting large volumes of data via tuning CPU frequency.

2.2.2 Energy Optimization Techniques in HPC

Large supercomputers in the age of exascale computing can have a power rating on the order of tens of MWs [11, 36, 39]. One approach in particular is CPU frequency scaling, as previous work in [11, 39] have shown to be effective. These studies focus on how dynamic voltage and frequency scaling are used to achieve energy savings, much like what we work to achieve in our study. Mòran et al., have proven success in not only optimizing the energy consumption of checkpoint/restart systems but also parameterizing it. Using CPU frequency as an independent variable, they explore different experimental settings to then characterize the system's energy consumption. Other work [36] demonstrates similar results to Mòran, finding a critical power slope. This curve shape denotes a sharp exponential increase in power over the range of CPU frequency, meaning energy savings can be achieved by lowering CPU frequency.

Similarly, estimation of energy consumption proves to be a useful tool in optimizing the energy efficiency of an HPC system [37]. Studies which perform this analysis focus on surveying a large number of energy optimized algorithms to then select, per-system, which algorithm is the best fit for a specific use case [37]. With this study we add the perspective of energy estimation in the context of lossy compression and data dumping towards exascale systems, adding a technique for that area of software.

In our work, we use frequency scaling for energy reduction in the context of lossy compression. We focus on parameterization and also a discussion about how to optimize I/O energy usage with efficient lossy compression. We look to evaluate the impact that the CPU clock frequency can have on the power usage of the HPC system.

2.3 Methods

The total energy, E_{total}, of a process is a product of average power, P_{avg}, and runtime, t_{run}:

$$E_{total} = P_{avg} \cdot t_{run}. \tag{2.1}$$

To establish a model for power consumption, one needs to measure the energy and runtime of that operation over a range of frequencies. Using `perf`, a Linux performance-measurement tool, we sample the total energy and runtime of compression. In our case, we model power consumption of I/O by recording energy and runtime of compression and data transmission.

We generalize our power experiments for compression with the usage of (1) the SZ and ZFP lossy compressors with different error bounds, (2) CPU frequency scaling, (3) data size and dimension to compress, and (4) hardware specifications for our experimental platforms.

2.3.1 Lossy Compressors and Error Bound

We use two leading HPC lossy compressors SZ and ZFP. ZFP [30] compresses by transforming floating-point data to fixed-point values block-by-block and adopts an embedded coding to encode generated coefficients. SZ [16, 28, 49] compresses blocks of data using a series of steps: data prediction, error quantization, Huffman encoding, and lossless compression. An important component of lossy compression is the error bound: the tolerance of how well the data should be reconstructed at decompression. A smaller error bound in general yields lower compression ratios and is more runtime expensive.

We use the absolute error bound in SZ and the ZFP fixed-accuracy mode, both of which bound the compression error at a fixed level. The error-bounds we target for both compressors are $1\mathrm{e}-1, 1\mathrm{e}-2, 1\mathrm{e}-3, 1\mathrm{e}-4$. These bounds are used to generalize compressor use-cases for the energy-tuning model, as different users utilize different granularity depending on the level of accuracy needed in data reconstruction.

2.3.2 CPU Frequency

We set the CPU frequency for all CPU cores using the Linux system call, `cpufreq-set`. Changing the CPU frequency affects the amount of voltage supplied to the chip. The range of frequencies we consider in this study goes from the the minimum clock speed (800MHz) to the maximum clock frequency for the given CPU with a step size of 50MHz. This step size was chosen as it would provide a sufficient granularity to notice trends in the total energy measured for each compression job conducted.

2.3.3 Data Characteristics

The dimensionality and size of data has a large impact on compression and data I/O. For example, a 10GB 4-D array of floating-point data is generally more difficult to compress than a 100MB 1-D array [33], because there are more points and dimensions to encode and process. SDRBench [45] provides floating-point, scientific data sets to assist in benchmarking compressor performance. In Table 2.1 we summarize the characteristics of the data we compress.

Domain	Dimensions	Size of Fields
CESM-ATM [27]	$26 \times 1800 \times 3600$	673.9MB
HACC [23]	1×280953867	1046.9MB
NYX [3]	$512 \times 512 \times 512$	536.9MB

Table 2.1: Data Sets Considered in Study

We utilize diverse data as it better generalizes the runtime and total energy the system consumes for a given compression operation. This step ensures greater ubiquity of results for scientific, floating-point data.

Note that for data transmission, only the size of the data matters. Therefore, for our experiments involving transmitting different chunks of data, we vary the size without worrying about dimension or domain.

2.3.4 CloudLab Hardware

We utilize different hardware platforms to better generalize our power consumption results. CloudLab is a cloud system which provides HPC nodes with remote, root access [21]. All experiments are run using CloudLab on the m510 and c220g5 node types. We summarize the pertinent, single-core hardware specifications in Table 2.2.

CloudLab	CPU	CPU Min - Base Clock	Series
m510	Xeon D-1548	0.8GHz - 2.0GHz	Broadwell
c220g5	Xeon Silver 4114	0.8GHz - 2.2GHz	Skylake

Table 2.2: Hardware Utilized

2.4 Modeling Power Consumption

In this Section, we develop models of lossy compressor and data writing power consumption using regression. This model has the purpose of providing an objective function to minimize the amount of energy lossy compression and data dumping consume. With a model parameterized in terms of CPU frequency, we predict and therefore simulate compressor behavior *ab initio*. We then understand how a system's power and runtime behaves with respect to compression and data writing, informing energy-savings for HPC system-users. First, using the data collected from compressing data at different frequencies with SZ and ZFP, we develop models for the power draw of lossy compression. Then we perform the same analysis for the power draw of writing data on an NFS.

2.4.1 Proposed Models for Compression Power Draw

Our goal is to create a general model for lossy compressor power consumption that provides the ability to estimate trade-offs in power and runtime for energy-efficiency. To achieve this we collect energy and runtime results for lossy compression. In our experimental setup, we performed single-core SZ and ZFP compression on the datasets in Table 2.1 on the two nodes in Table 2.2. We perform compression on the range of CPU frequencies in Table 2.2 with a step size of 50MHz. To ensure our model accounts for many different use cases we compress the data with four error bounds, as outlined in Section 2.3.1. Finally, to ensure validity we also repeat each compression 10 times per frequency step and average the results.

Since we use two compressors on two sets of hardware, we evaluate the five models of power draw which combine these variables. For each model we either regressed one compressor with both chips, or two compressors on a singular hardware configuration. We detail our five models for power consumption in Table 2.3.

Model Data	Compressor(s)	CPU(s)
Total	SZ, ZFP	Broadwell, Skylake
SZ	SZ	Broadwell, Skylake
ZFP	ZFP	Broadwell, Skylake
Broadwell	SZ, ZFP	Broadwell
Skylake	SZ, ZFP	Skylake

Table 2.3: Models Produced for Tuning

As shown in Figures 2.1 and 2.2, the power consumption data with respect to frequency is non-linear. Choosing the correct non-linear model is a matter of minimizing root-mean-squared-error (RMSE), sum-of-squared-error (SSE). Note that R^2 is not always accurate for non-linear modeling; however, it can still explain model variance [10]. From observing Figure 2, it is clear that there is a constant region with a sudden jump. We use the MATLAB™ Curve Fitting Toolbox [34] to find the model of frequency versus power, sliced along the partitions outlined in Table 2.3. This toolbox finds the most optimal model, minimizing SSE and RMSE. In this case all equations correspond with the following equation

$$P_{fit}(f) = af^b + c, \tag{2.2}$$

where f is CPU-frequency, and a, b, c are model fit parameters.

In Table 2.4 we present the models for power consumption of lossy compresssion, along with the goodness of fit metrics (GF), demonstrating how well each model predicts the power usage of compression.

Model Data	$P_{Compress}(f)$	SSE	RMSE	R^2
Total	$0.0086f^{4.038} + 0.757$	11.407	0.0442	0.5771
SZ	$0.0107f^{3.788} + 0.754$	5.964	0.0441	0.5864
ZFP	$0.0062f^{4.414} + 0.7589$	5.359	0.0440	0.5725
Broadwell	$0.0064f^{5.315} + 0.7429$	2.463	0.0279	0.8731
Skylake	$2.235\text{e} - 9f^{23.31} + 0.7941$	1.372	0.0226	0.8185

Table 2.4: Model Equations and GF for Compression

From the models and GF in Table 2.4, the Broadwell and Skylake power consumption models have a lower SSE and RMSE and an R^2 closer to 1, meaning power consumption is less dependent on the choice of lossy compressor. Therefore, power consumption results and energy-savings should turn towards hardware specifications when using the model-based tuning technique.

2.4.2 Proposed Models for Data Writing Power Draw

Here we continue our modeling of power consumption and extend these results to modeling data transit in terms of CPU frequency. With a power consumption model for data writing, we can optimize the energy usage of sending data over a NFS.

In our experiments for data transit, we measure the energy of transmitting floating point data. First, we allocate fixed sizes of data from 1GB to 16GB and then we copy that data to an NFS, using single-core data movement. During the data transport we measure the total energy and runtime. We perform this same operation over a range of frequencies for the single-core processes on both the Skylake and Broadwell chips. To reduce variance in results, we repeat each data transit at each frequency-step 10 times, averaging the results.

In our experiments, we vary the data size being written and the chip where we perform our tests. To find a regression model for power consumption, we must consider the power results from each CPU separately. Staying consistent with compression, our models are (1) all of the aggregated power results for data transit on both chips, (2) the power results from just the Broadwell chip, and (3) the power results from just the Skylake chip.

Regression of the power comes from plotting frequency and power from these different data partitions in the MATLAB^{TM} Curve Fitting Toolbox [34]. In the case of our models for power consumption of data transit, this was of the same form as Eqn. 2.2.

Model Data	$P_{Data}(f)$	SSE	RMSE	R^2
Total	$0.0133f^{3.379} + 0.7985$	0.8446	0.05631	0.4361
Broadwell	$0.0261f^{3.395} + 0.7097$	0.03423	0.01675	0.9578
Skylake	$9.095\text{E} - 9f^{20.9} + 0.888$	0.07875	0.02355	0.5992

Table 2.5: Models and GF Data Transit

From Table 2.5 we notice that the SSE and RMSE are minimized for the CPU specific models, as compared to combining the results. This implies that data transit power savings should be modeled on a hardware to hardware basis. The lower values of error do show that one can find the optimal frequency for power over the range of available frequencies per chip. The low R^2 for the Skylake data transit model is persistent among repeated tests, and shows that R^2 is an inconsistent metric for non-linear regression, as statistics like SSE and RMSE better characterize the error in a given model.

Using the models found in Tables 2.4 and 2.5, we are able to optimize data I/O. Treating I/O in two steps as compression and then data writing, different optimal frequencies are chosen, creating a piecewise model for I/O optimization.

2.5 Power Optimization of Compressed I/O

We propose the use case of tuning a CPU to achieve a more efficient power-state during compression and I/O for an HPC workflow. As noted in Sections 2.4.1 and 2.4.2, the choice of the hardware has the greatest influence on the model of power. In this Section, we analyze the energy consumption and runtime results to show how one can use the power model to improve I/O on a per CPU basis.

2.5.1 Characteristic Plot Results Analysis

Energy consumption and power draw can vary in magnitude on different systems. For example, the Intel Xeon D-1548 has a thermal design power (TDP) of 45W, as compared to the Intel Xeon Silver 4114's 85W TDP [21]. To make power and energy results comparable, we scale power-usage and energy consumption down by the max clock frequency's power consumption and total energy, respectively. Scaling down the power results has the advantage of putting all of the power on the same range as a percentage. Power as a percentage better illustrates the change in power over increasing clock frequency.

We note that for consistency in comparison, we also scale the runtime by the compression runtime at the max clock frequency. This has the added advantage of discerning the impact of a lower clock frequency on the runtime. Therefore, all results presented are in terms of their scaled value, not their magnitude, easing the analysis of changing clock frequency.

We also note that in Figures 2.1 and 2.2 we display the error bounds we compressed at, yet the trends are close to indiscernible as we found no significant difference between these results when scaling the power and runtime by the peak values on their range. However, one should note that there is a difference based on error bound in the magnitude of energy consumed, as shown in Figure 2.6. Similarly, we found no significant difference in the power consumption or runtime based on data size for Figures 2.3 and 2.4 after scaling results. Therefore, we do not show the different lines for data size.

Figures 2.1–2.4, summarize our scaled power and runtime result by plotting the scaled power consumption and runtime over frequency for compression and data writing. The two graphs also show different lines for the CPU and compressor. Around each trend we shade a 95% confidence interval, to account for possible noise or error in results.

2.5.1.1 Power Dissipation

The characteristic in Figure 2.1 represent the scaled power consumption of SZ or ZFP on the Broadwell and Skylake architectures. Since we seek power savings, any percentage below 1

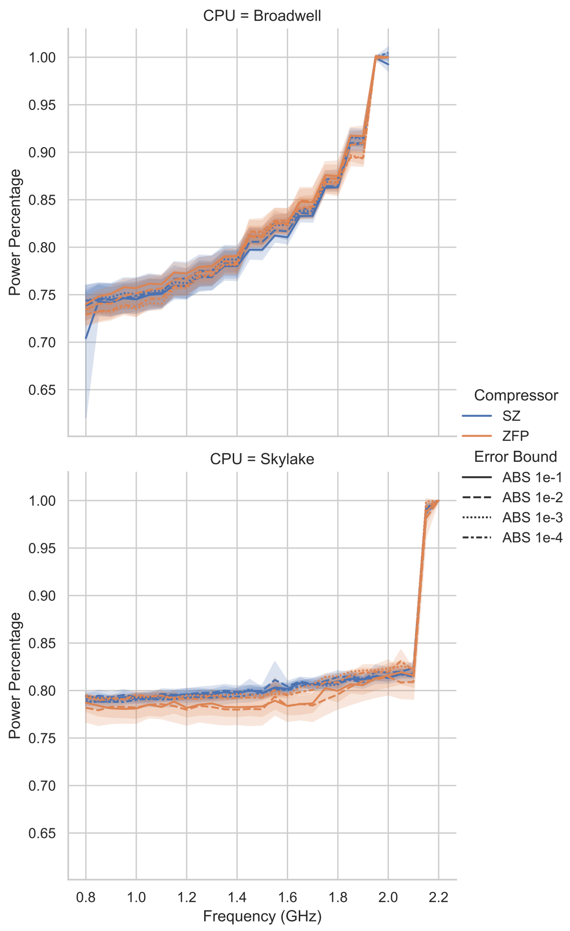


Figure 2.1: Compression Scaled Power Characteristics.

is considered power savings. Therefore, when optimizing for only power consumption we look for where the plot reaches its minimum. This occurs at the lowest frequency for both chips and all compressors. However, after analyzing runtime, we find that this is not the best choice for CPU frequency, as this is when the runtime is the greatest.

The error bounds that are included for the compression results do not greatly influence the scaled power consumption of either compressor or CPU. However, in Figure 2.6, one can see the effect of error bound on the magnitude of the energy without scaling. Based on the method of scaling, the calculation of power from Eqn. 2.1 factors out the magnitude of difference in runtime and energy that occurs between error bounds. One should note that with a more fine error bound, compression will typically take longer and have a lower compression ratio [28]. To represent different error bounds in the data transit case, we vary the size of the data being sent.

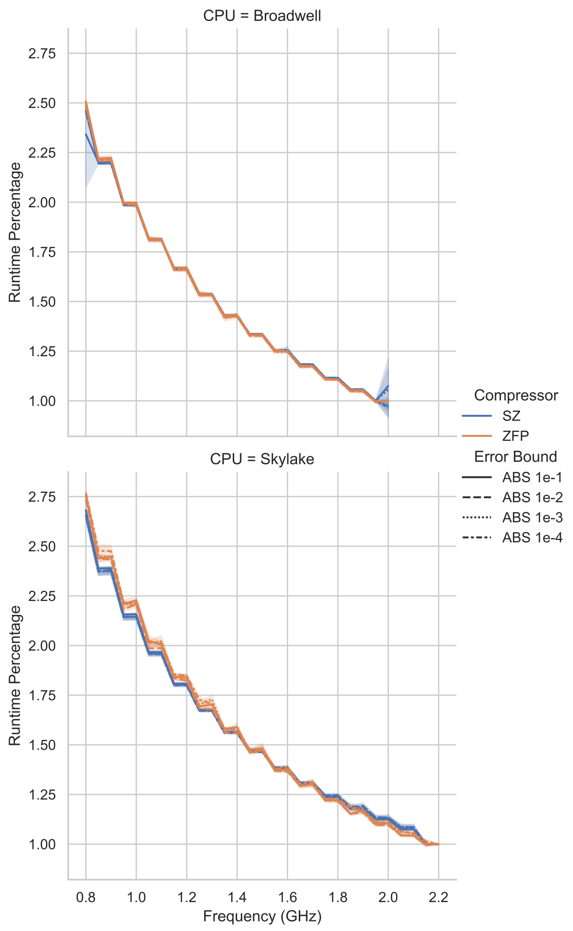


Figure 2.2: Compression Scaled Runtime Characteristics.

Similarly, in Figure 2.3 we present scaled power consumption of writing different sizes of data on Broadwell and Skylake architectures of a series of frequencies. We find that the lowest power consumption comes at the lowest frequency value. The primary difference between the Skylake and Broadwell is the Skylake doesn't have as large of a range as the Broadwell. This is consistent with the compression results. The reason for a power percentage of 0.9 in data writing as compared to 0.8 during compression is that data writing can be more intensive than the steps to compress. Therefore, we do not see as significant of power savings during data writing.

Figures 2.1 and 2.3 demonstrate the critical power slope which is discussed in [36]. The critical power slope is a sharp, increasing curve of CPU power draw over a set of frequencies. In this model, the values in the curve are nearly constant for a large range of frequencies. This is the same behavior seen in Figures 2.1 and 2.3.

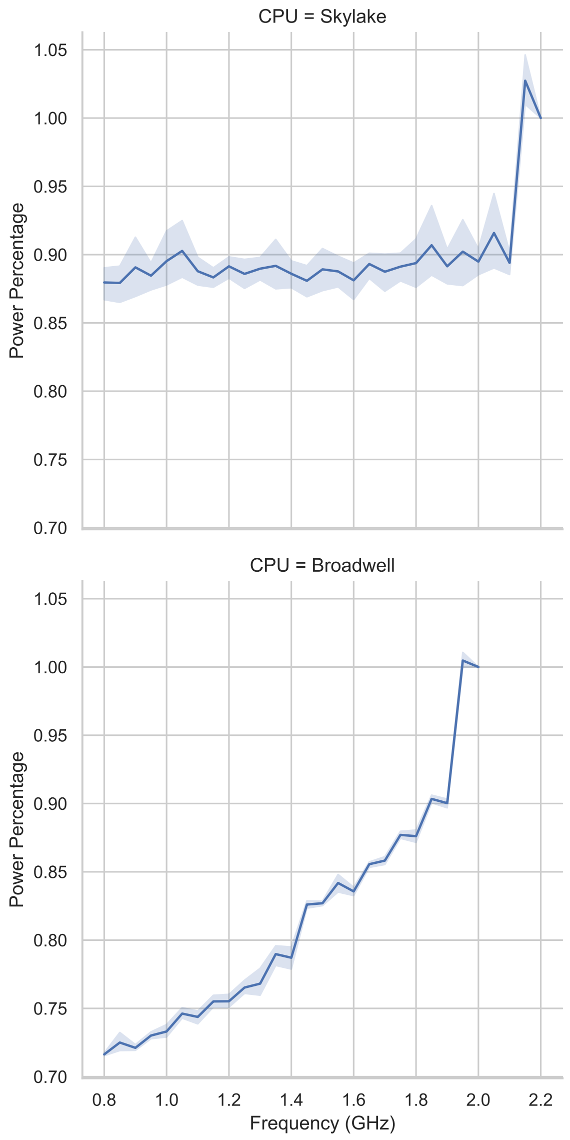


Figure 2.3: Data Transit Scaled Power Characteristics.

By utilizing the models found in Table 2.4 and the data in Figure 2.1, we compute that for compression on both CPUs and both compressors we achieve approximately 19.4% power savings during compression, by lowering frequency by 12.5%. Similarly, using the data in Figure 2.3 and models from Table 2.5, we achieve approximately 11.2% power savings on average during data writing, by lowering the CPU frequency of both chips by 15%.

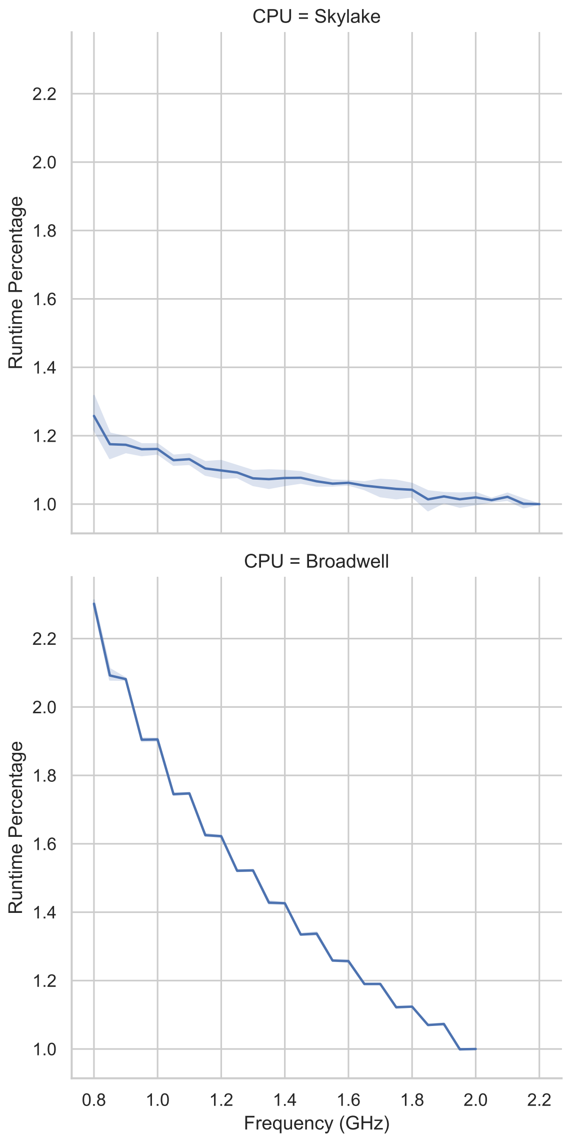


Figure 2.4: Data Transit Scaled Runtime Characteristics.

2.5.1.2 Runtime

The values in Figure 2.2 represent the scaled runtime of SZ or ZFP compressing data over a range of clock frequencies. In this graph, we interpret a percent runtime below 1 as runtime savings. Notice that the best compressor runtime in Figure 2.2 comes at the highest clock frequency, meaning that if a user desires very fast compression, they should utilize the max clock on a CPU. However, this is not the minimal power consumption, as this occurs at the lowest clock frequency. Therefore, we find where power is minimized and runtime is minimized. This is at the point of 12.5% frequency reduction for compression.

Yet again, the included error bounds for the scaled runtime in compression do not influence the trend. This is again due to the technique of scaling, as the magnitude of runtime is higher for finer error bounds.

Similarly, in Figure 2.4 we show the scaled runtime of writing different sizes of data over a range of frequencies. The lowest runtime still occurs at the max clock frequency. Yet, as mentioned above this is not where power is minimized as well. This point then occurs for data transit at 15% frequency reduction during data writing.

We note also that in Figure 2.2, the trends overlap showing consistent runtimes between SZ and ZFP. We find that the runtime is stagnant in data writing for the Skylake processor, as this is similar to the power in Figure 2.3 not scaling during this operation. This is likely indicative of the load data writing puts on a single-core for the Skylake generation. Also we note that energy and power analysis of the Skylake chips have not shown major improvement in energy efficiency over older generations, despite being newer and having better performance [44]. Therefore, the stagnant scaling is indicative of this lack of energy efficient scaling.

An important note about reducing clock speeds is with a lower clock speed, there is an increase in the runtime of an application, as seen in Figure 2.2. The equation for average power, Eqn. 2.1, factors out time, meaning those results are invariant of runtime. Therefore, one must notice the affect that the intertwined metrics have on the energy of the system.

2.5.1.3 Energy-Aware Trade-off

As mentioned previously, the goal of this study is to optimize the energy consumption of lossy compression and writing data with CPU frequency for HPC I/O. In Figures 2.1–2.4 we present an approach for monitoring the system power and time usage during compression. In our analysis of the runtime and power dissipation separately, we find that the best power and time savings are at opposite ends of the frequency spectrum. The most power is saved at the lowest clock frequency, whereas runtime is optimal at the highest clock frequency.

Decreasing energy through lowering power with clock frequency is a trade-off. Would a user benefit from faster compression? or less energy-consumed? This trade-off is a user and system consideration, yet the decreased runtime nets significant energy-savings during data I/O. We achieve a maximal 19.4% power savings with a net 7.5% increase in runtime, by decreasing the CPU clock speed by 12.5% in both models of compression. For data writing we achieve 11.2% average power savings with a net 9.3% increase in runtime, by decreasing the CPU frequency by 15%. Symbolically, we represent our recommendations for tuning the CPU frequency f in Eqn. 2.3, with respect to the maximum clock frequency f_{max}.

$$f_{I/O} = \begin{cases} 0.875 f_{max} & \text{lossy compression} \\ 0.85 f_{max} & \text{data writing} \end{cases} \tag{2.3}$$

Averaging these two savings, we find that on average these savings are equivalent to 14.3% energy savings with a net increase of 8.4% in runtime during lossy compression and data writing.

2.6 Frequency Tuning for Energy Savings

In this Section, we utilize the findings of Sections 2.4 and 2.5 to show how our power model and recommendations for tuning in Eqn. 2.3 are used to predict and lower energy usage in HPC I/O. We present two examples that demonstrate the versatility of our model and solution against non-optimized clock frequency compression and data dumping. In the first example, we show our model for the Broadwell chip against different datasets not included in model regression, to demonstrate how well our work estimates new results. In the second, we measure energy consumption for data dumping on a file system (NFS) through Cloudlab, utilizing our optimized CPU frequency recommendations.

2.6.1 Estimating Power Consumption Model

To test our model for new results, we perform the same experiment with the Hurricane-ISABEL dataset from SDRBench [45]. Hurricane ISABEL represents a weather simulation for different weather metrics by the National Center for Atmospheric Research. The uncompressed floating-point data snapshots have dimension $100 \times 500 \times 500$; we compress six 95MB fields (PRECIP, P, TC, U, V, W) using both SZ and ZFP with a $1\mathrm{e}-4$ error bound.

Measuring the average power across all frequencies, we produce similar results to Figures 2.1 and 2.2. In Figure 2.5 we demonstrate how well our power-model fits against the power dissipation in the new results. We calculate $\text{SSE} = 0.1463, \text{RMSE} = 0.0256$ for these curves, demonstrating that the model estimates the data well with little error . From this, we determine that our model estimates power behavior well, even with data not factored into our model.

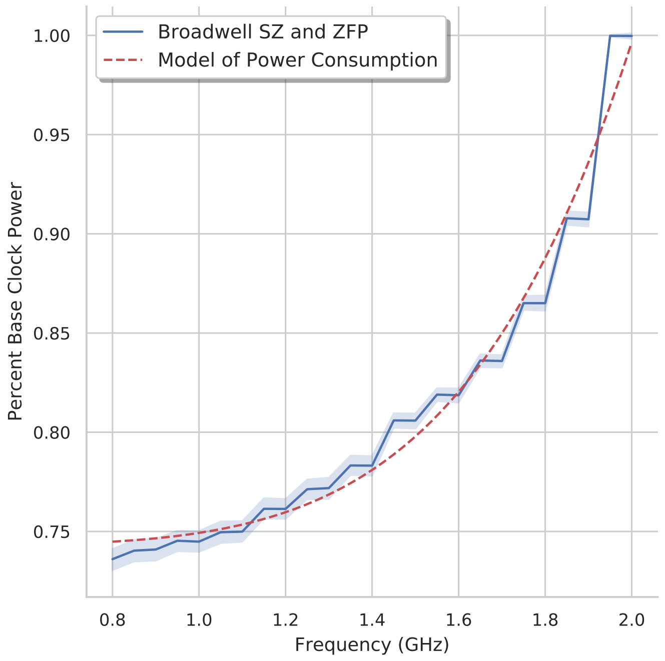


Figure 2.5: Broadwell Chip Model for Power Consumption

2.6.2 Data Dumping with Lossy Compression

Data dumping is a popular method for transferring data on an HPC system. Lossy compression is often used to reduce size and expedite data movement. However, compression can take significant energy before data movement. As the impetus for this study, we use our tuning framework in Eqn. 2.3 as a test against non-optimized data movement.

As an experiment we simulate compressing and transmitting 512 GBs of NYX data, attained by concatenation, over an NFS with and without tuning the clock frequency. In particular, we lower the clock frequency by 12.5% for SZ compression and then by 15% to tranport the data, as per Eqn. 2.3. In both our frequency scaling and the base clock we record the total energy dissipated in the experiment. We compress using SZ over several error bounds $(1e-1, 1e-2, 1e-3, 1e-4)$ to demonstrate how the difference in compressed data size affects data transit and the magnitude of runtime results for large datasets. In this example, we compress and transmit a 512GB velocity-x field of the NYX dataset on a 10Gbps ethernet connection.

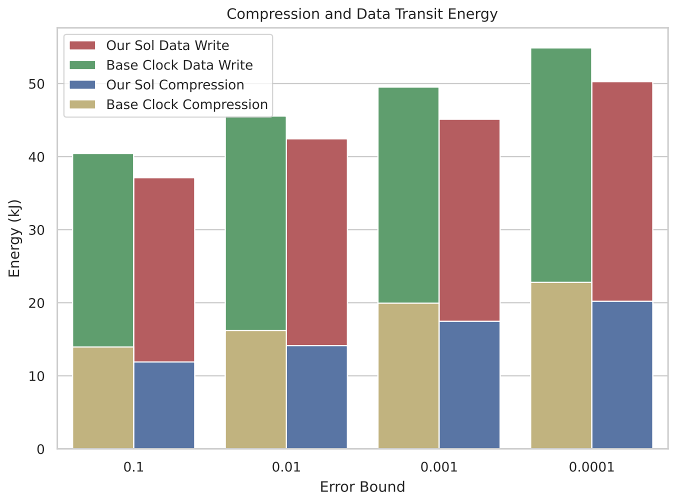


Figure 2.6: Energy Dissipation for Data Dumping

As shown in Figure 2.6, our solution always reduces the amount of energy consumed in data compression and movement. The non-scaling compression and data transit is represented by the tag Base Clock. We show that compression energy usage and data movement is reduced by lowering the frequency. On average this is 6.5kJ, or 13%, of energy saved over all of the error bounds using our solution. While future studies will strive to address whether these trends hold on different CPUs, with exascale data, these initial results show promise in reducing the energy consumption of data transit with a compression stage.

2.7 Conclusion

In this study, we present a framework for modeling the power consumption of lossy compressors and I/O on HPC systems. We recommend lowering the CPU frequency by 12.5% during compression and by 15% during data writing to minimize energy usage. Through our analysis of CPU power-results, we optimize energy-usage of I/O bound jobs by 14.3% on average, with only a net 8.4% increase in runtime. We find this to be consistent across different CPU architectures, datasets, error bounds, and compressors. We also demonstrate that our model estimates power consumption accurately, and saves 6.5kJs on average for compressing and writing 512GB of floating point data. Applications of these findings in HPC computing centers will help meet green-computing initiatives and assist sustainable computing goals.

Chapter 3

Analyzing the Energy Consumption of Synchronous and Asynchronous Checkpointing Strategies [56]

3.1 Introduction

Applications are facing ever-increasing runtimes and produce massive amounts of intermediate data [41] as production-ready HPC systems continue to scale. In June 2022, Frontier became the U.S.'s first Exascale capable machine, with a reported 8,730,112 compute cores, a 72% increase from its predecessor, Summit [1]. Historically, similar increases have led to difficulties in the form of: (1) high energy consumption due to relatively slower advancements in power technology and increasing complexity of software stacks [58]; and (2) lower mean-time-between-failures (MTBF), raising questions about hardware reliability [6].

In 2008, the scientific community set an ambitious goal to power future exascale-capable machines on 20 MW per year [7]. Even if we assume a very low cost of energy in the U.S. (<\$0.05/kWH), 1 MW of power results in almost \$1 million USD per year [46], meaning upwards of \$20 million USDs is required just to power the system. However, power budgets are subject to change based on new chip technology, cooling strategies, and facility sizes [46]. Thus, it is clear that power draw, and therefore energy, is a major expenditure. Such large energy costs raise problems for users as well, as they run the possibility of system usage being capped [43]. With energy being a key variable to control for both users and site administrators, it is pertinent to understand the impact different applications and middle-ware libraries have on energy consumption in the HPC stack.

On the other hand, the dramatic increase in number of components leads to lower mean-time-between-failures (MTBF) [6], which prompts the need for scalable, low-overhead resilience techniques. In the context of HPC applications, checkpoint-restart (C/R) is the most common resilience technique due to the tightly coupled nature of the processes and tasks. In a nutshell, C/R captures the global state of an application in a resilient fashion at a given point in time, and restarts from that state in case of failures, which dramatically reduces the amount of lost runtime compared to re-running the application from the beginning.

C/R approaches have evolved over time to leverage heterogeneous storage stacks (node-local memory hierarchies and SSDs, external repositories such as a parallel file system –PFS–, key-value stores, etc.) as part of multi-level resilience strategies that adapt to various classes of failures. Based on empirical observations, simple failures like application bugs happen more frequently than catastrophic failures like a large number of nodes going offline at once. Thus, multi-level resilience strategies employ lightweight "levels" to protect against simple failures more frequently (e.g. checkpoint to and restart from local storage) and more expensive "levels" to protect against catastrophic failures less frequently (e.g. checkpoint to and restart from a PFS that offers durability at the expense of high I/O overheads).

Even so, shorter mean time between failures (MTBF) means checkpointing needs to be performed more often at all levels of resilience. As a consequence, checkpoints are written to a PFS more frequently, which introduces unacceptable I/O overheads. To alleviate these overhead, asynchronous multi-level checkpointing techniques [41] block the application only until the checkpoints have been written to local storage (fastest level), then proceed with the other resilience strategies in the background (e.g., flush checkpoints to the PFS using separate threads), while the application keeps running. This overlap hides the high I/O overhead of accessing a PFS, but is more complex to implement and introduces competition for resources with the application, which is non-trivial to predict and mitigate [25, 51, 52].

The combined effect of more frequent checkpointing using increasingly complex techniques, coupled with the need to increasingly emphasize energy efficiency, has prompted the need to design multi-level checkpointing techniques that are not only low-overhead and scalable, but also energy efficient. In this context, previous studies analyze the effects of the C/R software-stack (e.g. I/O design choices [52] and different storage layers [41] on application time memory, and bandwidth consumption. However, energy efficiency has comparatively received limited attention in the literature. There are recent efforts into exploring the energy *consumption* of checkpoint-restart, but these works only consider synchronous checkpointing libraries [39], and/or model certain aspects (e.g. checkpoint frequency [14], recovery mechanisms [35], or storage tier [4]) of C/R, not the libraries themselves (which has unaccounted for inefficiencies due to software implementation).

Thus, there is a need to analyze the energy efficiency in a *holistic* fashion that takes into account the overall complexity of checkpointing libraries. This is particularly important in the context of asynchronous multi-level checkpointing libraries, which have both complex implementations and runtime behaviors that involves competition for resources with the application. Just as understanding the interplay between applications and checkpointing runtimes is necessary to mitigate contention and reduce performance degradation, understanding the trade-off between performance, scalability, and energy consumption using various combinations of parameters and resilience strategies is a crucial step in the design of next-generation energy-efficient checkpointing libraries. In this paper we focus on the study of the aforementioned trade-off. We summarize our contributions below:

- We perform weak and strong scalability analysis of the trade-off between the performance overheads and energy consumption of VELOC [41], a production-ready asynchronous checkpointing library, as compared with GenericIO [24], an optimized synchronous checkpointing library for writing checkpoints directly to a PFS.

- We compare the impacts of C/R configurations such as storage hardware, I/O methods, and parallelism on the energy consumption and throughput of C/R applications.

- We detail both the CPU and DRAM energy consumption of C/R on different storage tiers at scale, combining these measurements to report aggregate results.

- We find asynchronous C/R writing to DRAM provides nominal improvement over SSDs in regards to throughput and energy consumption. Therefore, efforts should be taken to efficiently use both resources to provide high throughput and reduce contention for fast, local memory.

3.2 Background and Related Work

3.2.1 Checkpoint-Restart

HPC checkpoint-restart (C/R) captures the global state of multiple processes as a set of checkpoints that can be used to restart from in case of failures. Typically, C/R captures only the data structures residing in the memory of the processes, while other additional states are discarded and reconstructed on restart.

System-level vs. Application-level C/R System-level C/R (such as DMTCP [5]) employs a transparent approach that captures the full set of in-memory data structures, while application-level C/R captures only the critical data structures and relies on the application to explicitly reconstruct the other data structures on restart. In this paper we study the latter approach. However, without loss of generality, our results can be used to reason about system-level checkpointing libraries too, as the same resilience techniques are applicable once checkpoint files have been produced by either approach.

Synchronous C/R in this case, all multi-level resilience strategies are applied while blocking the application. A representative example is SCR [38]. In practice, it is often employed as a single level that involves blocking flushes directly to a PFS. In this context, several optimization are employed to reduce the I/O pressure on the PFS, such as aggregating checkpoints on a subset of compute nodes that are responsible for concurrent writes, as illustrated by GenericIO [24]. This type of C/R helps ensure a consistent global-state across a distributed application [39]. Further, synchronous C/R eliminates competition between the checkpointing runtime and the application, thus eliminating the need for interference mitigation, which is a non-trivial issue [52]. However, at scale, synchronous checkpointing strategies have high (and often unpredicatable) overheads, especially when concurrent writes to PFS(s) are involved, which is typically subject to I/O bottlenecks.

Asynchronous C/R Asynchronous multi-level checkpointing techniques block the application only until the checkpoints have been written to local storage (fastest level), then proceed with the other resilience levels in the background, while the application keeps running (e.g. flush checkpoints to the PFS using separate threads). A representative example is VELOC [41], which supports several multi-level resilience strategies: (1) capture the checkpoints in-memory or to node-local storage; (2) partner replication, (3) peer-to-peer erasure coding, (4) flush to PFS(s), (5) flush to burst buffers or (6) flush to key-value stores. In this paper, we focus on a simple two-level scenario that is frequently used in practice: (1), which is blocking, followed by (4), which is performed asynchronously.

3.2.2 Energy-Aware Computing

The growing energy consumption of computing systems is immense and requires significantly greater costs and/or power capping users. In recent years, energy-aware computing (EAC) has become a focus of the HPC community at large, with efforts being made to reduce the environmental impact of supercomputers. The main directions of research for energy reduction are through (1) analytic, (2) hardware, and (3) software optimizations/comparisons. Here we outline the differences of these approaches and discuss where our work falls.

Analytic EAC In distributed computing, analytic approaches to optimizations are popular, as mathematical models of parallel systems yield interesting insights into potential pitfalls in performance. While not exhaustive, this section focuses on the analytic models of C/R that optimize energy and power models through aspects of C/R such as checkpoint intervals, scalability, and probabilistic execution. Some works [2, 17, 18] focus on optimizing checkpoint frequency via the Young-Daly equation [13], often modeling runtime speedup with this equation. Others find a more appropriate probabilistic model for C/R which is proven to allow a more fine-tuned model of energy consumption, resulting in better prediction and future optimization [8, 53]. We do not proceed with analytic methods, instead indirectly use the results as they are implemented by C/R application developers [24, 41].

Hardware-Focused EAC Hardware-based EAC refers to optimizing the energy consumption of a node via controlling hardware variables such as CPU/GPU frequency and voltage or different storage locations/hardware. This level of control is achieved through dynamic voltage and frequency scaling (DVFS). Voltage and frequency are easily tunable variables through OS-specific kernel tools (e.g. `CPUfreq` for Linux) and directly lowers power consumption of a device.

In the context of C/R, previous works [40,48,55] suggest DVFS is useful in targeting different CPU frequencies to find an optimal power consumption without large performance loss. Other works look at how different hardware types (e.g. NVMe v.s. SSD) impacts the energy efficiency of C/R applications [32, 42]. In this paper, we present a greater focus on the different storage hardware types. Since resource contention introduced by asynchronous C/R is not well understood, we choose to leave DVFS work to more in-depth future studies, where it may have adverse effects on application runtime.

Software EAC Another area of EAC optimizes energy consumption through software. Often times this comes in the form of analyzing similar applications to explore which configuration yields the greatest performance and lowest energy cost. The methods employed by these studies typically consist of power/energy monitoring over a series of different tests for a series of applications [12]. While studies currently exist that have looked at the trade-offs in energy efficiency of synchronous C/R such as [19, 47], none to our knowledge have explored the gap of comparing the energy consumption of synchronous and asynchronous C/R from a software perspective. We fill this gap in our study by analyzing the energy scalability of asynchronous C/R and the varying storage tiers compared to other synchronous, single-leveled C/R solutions.

3.2.3 Energy efficiency of C/R

Moran et al. [39] modeled ways to predict the energy consumption of C/R in HPC, however they do not assume heterogeneous storage. Various other studies model energy consumption of C/R at Exascale; one such study by Dauwe et al. [14] in 2017 predicted Exascale hardware configurations and metrics rather well and took into account various storage tiers used by multi-level checkpointing. However, they analyze the energy efficiency of the storage levels independent of C/R. Amrizal et al. [4] assume checkpoint and restart times are constant, which does not apply for PFS(s) that show significant variability in aggregated I/O bandwidth. Miao et al. [35] looks at the interplay between certain scientific workloads and various synchronous resilience strategies, emphasizing various trade-offs. Other studies that seek to improve energy efficiency [9, 35] of C/R suggest throttling CPU power. However, such studies typically assume synchronous C/R, which already has unacceptable overheads at Exascale that would only be amplified by power CPU throttling. In asynchronous C/R, power throttling could also have adverse effects: while the I/O operations are overlapped with the application runtime, they need to finish by the time the next checkpoint request arrives, otherwise resilience is not guaranteed. If this does not happen naturally, the checkpointing library needs to guarantee it by blocking the application, which reduces the effectiveness of asynchronous techniques.

Overall, a comprehensive study of the trade-off between performance overhead, scalability and energy efficiency of C/R is necessary in order enable the design of efficient next-generation asynchronous C/R libraries. To our best knowledge, we are the first to address this gap.

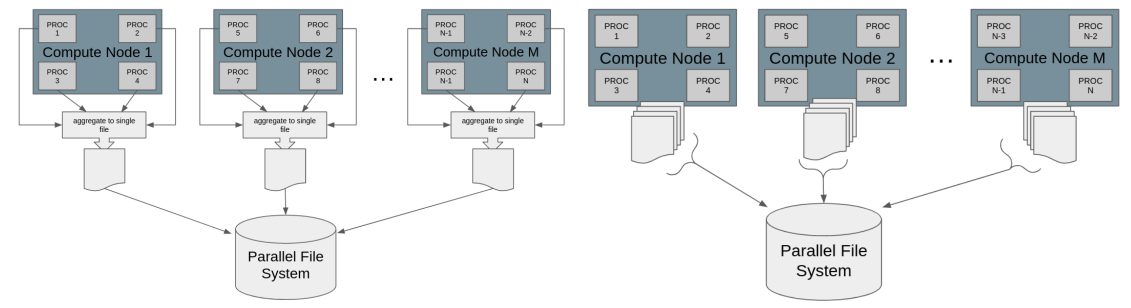
3.3 Methodology to Study Energy Consumption of C/R

To study the overhead vs. energy efficiency trade-off for C/R, we investigate both synchronous and asynchronous resilience strategies under various write scenarios driven by a benchmark we implemented specifically for this purpose. The benchmark eliminates competition between the application and C/R library, allowing us to accurately identify C/R settings (i.e., I/O strategy, synchronicity, storage hardware) that inherently contribute to high energy overheads. If some configurations generate high energy overheads in isolated testing, it is reasonable to conclude that such overheads would be exacerbated in the presence of competitive applications.

We focus on both weak and strong scalability. Weak scalability illustrates how energy is impacted as the problem size and participating processes grow and provides valuable insight into how we predict the C/R libraries to perform on Exascale workloads and systems. Strong scalability is a corollary to Amdahl's law, allowing us to estimate how higher degrees of parallelism affects the upper limit of energy consumption. For application developers, interpreting our results provides insight for choosing C/R strategies and their optimal settings to reduce energy consumption without sacrificing performance; for C/R developers, this work helps identify areas of C/R that contribute to high energy overheads. We detail our methodology below.

3.3.1 Compared Approaches

We have chosen to focus on two representative checkpointing approaches: (1) GenericIO (GIO) [24], which is representative of an application-level, optimized synchronous checkpointing approach that directly writes the checkpoints to a PFS; (2) VELOC [41], which is representative of an application-level, optimized multi-level asynchronous checkpointing approach that first captures the checkpoints to local storage and then flushes them in the background to the PFS. GIO can use



(a) GenericIO: MPI collective I/O using $N \rightarrow M$ aggregation

(b) VELOC: POSIX I/O using one file-per-process flush strategy

Figure 3.1: Checkpointing libraries: GenericIO vs. VELOC

a variety of ways to flush the checkpoints to the PFS, including MPI I/O collective operations and POSIX read/write system calls. This allows GIO to be tuned to accommodate many system architectures and persistent storage options. GIO aggregates by default the checkpoints of N processes running on a large number of compute nodes into a smaller subset of M files. We call this N-M aggregation, which is illustrated in Figure 3.1a. M is tunable by the user to better match system composition such as storage targets, I/O servers, or other user-based constraints. For the purpose of this work, we configure GIO to use MPI I/O collective operations, and we let GIO automatically optimize M.

VELOC can capture the checkpoints on a variety of node-local memory hierarchies and storage devices (SSDs, HDDs). We study two POSIX options: in-memory using `tmpfs` (`/dev/shm` mount point) and SSDs using `ext4`. From here, we flush the checkpoints asynchronously to the PFS using a one-file-per process strategy (Figure 3.1b), which was chosen because of its high I/O performance under write concurrency (each file ends up on a single storage sever in its own dedicated stripe) and because it complements the I/O aggregation strategy used by GIO. A single active backend is responsible on each compute node for interacting with the PFS in an asynchronous fashion. We use the default settings, which results in a number of I/O threads on each active backend equal to the number of co-located application processes on each compute node. VELOC is modular and employs a wide array of other resilience strategies: partner replication, peer-to-peer erasure coding, flushing using burst buffers and key-value stores, etc. However, we have chosen to focus on a basic scenario that is the most frequently used in practice.

Compared Aspects The comparison between GIO and VELOC uses a combination of alternative approaches that affect the resilience strategies: synchronous vs. asynchronous, I/O aggregation into a small set of files vs. one file-per-process, MPI collective I/O operations vs. POSIX I/O, direct writes to the PFS vs. leveraging a heterogeneous storage hierarchy. These have different behaviors and implications at scale both with respect to performance overheads and energy consumption, which results in a relevant multi-faceted comparison.

3.3.2 Measurements and Energy Monitoring

In this paper, our main focus is to characterize the energy scalability of asynchronous checkpointing, and different storage devices. To do this, we need the ability to capture the energy consumption of these processes. Thus, we require an energy monitoring tool that polls the hardware counters that exist at the kernel level for energy consumption of the CPU and DRAM. While there are various tools to accomplish this as mentioned in [12] we perform our tests on an Intel CPU and utilize the Running Average Power Limit (RAPL) [15] hardware counters. We measure these results with Performance Application Programming Interface (PAPI) [50].

Powercap and RAPL Based on our cluster's kernel settings, our interface RAPL was through the *powercap* hardware counters. In Linux, these exist in the `/sys/devices/virtual/powercap` path and consists of several levels of telemetry data. These registers are divided into Zones 0 and 1 with a Subzone 0 and 1 in each Zone. These divisions represent different portions of the hardware. For example, Zones 0 and 1 capture the data for all cores of the CPU and the Subzones 0:0, 0:1, 1:0, 1:1 capture data for the DRAM on board [15]. For our case, with two zones and two subzones, if the energy of zones 0 and 1 are E_0, E_1 and subzones 0:0, 0:1, 1:0, 1:1 are $E_{0:0}, E_{0:1}, E_{1:0}, E_{1:1}$ then the total energy of the CPU and DRAM E_{CPU}, E_{DRAM} are:

$$E_{CPU} = E_0 + E_1 \tag{3.1}$$

$$E_{DRAM} = E_{0:0} + E_{0:1} + E_{1:0} + E_{1:1}. \tag{3.2}$$

We note that DRAM and CPU are the only available hardware counters on our system. Therefore, other presentation of total energy is based on these two terms. In this study we define Total Energy (kJ) to be

$$E_{total} = E_{CPU} + E_{DRAM}. \tag{3.3}$$

PAPI To poll RAPL during application runtime, we use PAPI, an interface that allows users to collect program performance metrics via C library calls. This allows a user to determine what they would like to measure and when/where to measure it. In our case, we are interested in the energy consumption of the checkpoint operations of both synchronous and asynchronous C/R libraries. It is important to note that these constitute different methodologies, as C/R synchronicity changes the phases of a program, as touched on in Section 3.2.1. Ignoring the setup of PAPI, in Algorithm 1 we demonstrate that synchronous C/R consists of a blocking checkpoint to persistent memory.

Algorithm 1 PAPI Measurement in C/R

```
Input: data_to_ckpt[], PAPI_EventSet, PAPI_results[]
Output: PAPI_values_arr
    Note: PAPI_EventSet initalized with powercap events
 1: PAPI_start(PAPI_EventSet);
 2: CKPT_LIB.checkpoint(data_to_ckpt[]);
 3: if VELOC_ASYNC then
 4:     Local storage ckpt energy
 5:     PAPI_read(PAPI_EventSet, PAPI_results[0]);
 6:     CKPT_LIB.wait();
 7: end if
 8: Persistent storage ckpt energy
 9: PAPI_stop(PAPI_EventSet, PAPI_results[]);
10: return PAPI_results[]
```

On the other hand, asynchronous multi-level C/R first checkpoints to a local device of the user's choosing (in our case DRAM or an SSD), then flushes said checkpoints to persistent storage (PFS) in the background. For comparison, we are interested in the energy of both of these operations, therefore we read the results at two points as shown in lines 5 and 9 of Algorithm 1.

It is worth noting that we are unable to measure the energy consumption of the SSD and the HDD using PAPI or software available on the Palmetto Cluster. These storage drives are not equipped with hardware counters to poll the energy draw from the kernel-perspective, also we are unable to open the system and perform physical energy monitoring. We consider modeling these missing results by adding the average TDP of the devices multiplied by the runtime of the checkpoint to the total energy to model these results. However, in doing so we would scale everything by almost the same factor. Therefore, for clarity we choose to leave these modeled results out and instead only report the empirical data from PAPI and RAPL.

3.4 Experimental Results and Discussion

In this section, we detail our experimental design for comparing the energy consumption of C/R libraries. We also present our results for energy usage of synchronous and asynchronous C/R writing to different hardware. In our tests, we evaluate these metrics along with throughput of the application in both weak and strong scaling contexts to present a more full evaluation of the merits of the different C/R options.

3.4.1 Experimental Platform

Hardware We carry out our experiments on a multi-node HPC cluster with a 100 Gbps Infiniband interconnect. Each node is equipped with two Intel Xeon(R) Gold 6148 CPUs with a total of 40 cores, a max clock frequency of 2.40GHz, and 370GB of DDR4 DRAM. The system has various storage options, we summarize the storage locations we employ in Table 3.1. Asynchronous C/R requires both a local memory location and a persistent storage location to write to once local checkpointing is complete. The persistent storage was chosen as it uses a PFS. Therefore the Usage column points out how each location is utilized.

Table 3.1: Storage Hardware Summary

Location	**Hardware**	**Size**	**File System**	**Usage**
/dev/shm/	DRAM	370GB	tmpfs	Scratch
/localscratch/	SSD	1.8TB	ext4	Scratch
/scratch1/	HDD	1933TB	beegfs	Persistent

In our testing, we attempt to scale to four nodes using MPI for a total of 160 processes. However, the system in use provides only 24 I/O servers and fixed stripe settings of 4 storage targets and a stripe size of 512kB. The system prioritizes job compactness and throughput of small jobs versus large jobs. Since neither of the C/R libraries we use have any sort of node-level synchronization, it is unlikely that the performance degradation seen in the multi-node configuration as a result of these settings is indicative of the overall scalability of the C/R libraries. Thus, we only present single-node results.

In the following tests, the term local refers to VELOC's blocking write operation to local storage as a part of the multi-level checkpointing heuristic. GenericIO does not write to a scratch memory and is therefore not depicted in said figures.

Software We utilize VELOC (v1.6) and GIO (Git tag 20190417) to perform C/R operations. To monitor energy, we use PAPI (v6.0.0) to read the RAPL counters on the aforementioned Intel CPUs and the cluster's operating system of Rocky Linux (v4.18.0). We interface with these hardware counters through PAPI (v6.0.0) and its C/C++ libraries. We compile our code using GCC 9.5.0 and OpenMPI 4.1.3.

3.4.2 Weak Scalability

Here we present the results for a series of weak scalability tests. For each approach and storage location from Table 3.1, we spawn N ($1 \to 40$) processes. Each rank checkpoints a uniform size of either 1 or 2 Gigabytes (GiBs). We repeat each experiment 50 times and average the results, whose variability is depicted using errorbars.

Note that checkpointing ≥ 2 [GiB] per MPI rank experiences memory overflow errors due to MPI-IO aggregation in the case of GIO. Therefore, we use their POSIX read/write implementation. VELOC is capable of using different I/O flushing methods: (1) *sendfile* system call that internally transfers data and (2) POSIX mmap/write; (3) regular read/write. For the purpose of this work, we use POSIX mmap/writes, since the sendfile system call is not efficiently implemented by our PFS.

Figures 3.2 and 3.3 illustrate how different local storage devices affect throughput and energy consumption when using VELOC. In Figure 3.2 we see that for local checkpoints, DRAM provides $\approx 25\%$ more throughput than the SSDs as processes begin to scale. DRAM supports higher throughput in general, but this actually has a direct impact on the energy use. Only as the throughput begins to differ significantly do we start to see a gap in energy consumption between the two devices. Thus, we conclude throughput dictates energy consumption.

In Figure 3.3 the energy consumption is shown to be linearly correlated with the size of

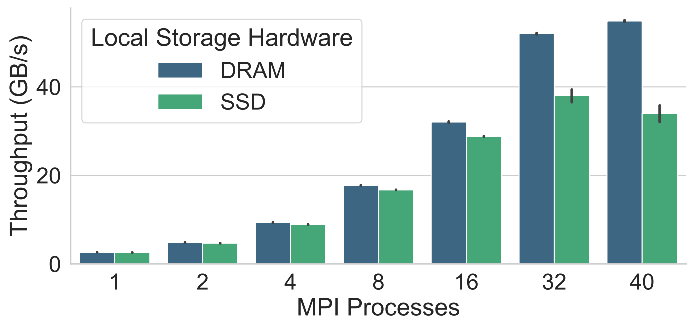


Figure 3.2: Weak scalability of I/O throughput for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Higher is better.

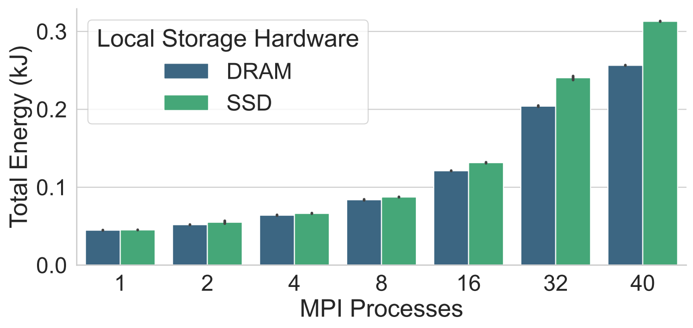
data, as illustrated in the difference between Figures 3.3a and 3.3b. Energy cost is dominated by the magnitude of data movement, as checkpointing 2 GiB per rank requires $\approx 2\times$ more energy than 1 GiB, regardless of the scratch memory location. On average, writing locally to DRAM takes $\approx 20\%$ less energy than to a local SSD.

Therefore, weak scaling reveals that asynchronous checkpoints to DRAM has a higher throughput and lower energy cost, incentivizing its use in C/R. However, this is really only a nominal difference. In a typical scientific application, DRAM is a heavily contested resource which may not be large enough to store both application data structures and checkpoints. Further, reduction techniques (such as compression) are becoming increasingly necessary to address the exponentially growing size of data, meaning it is unlikely that distributed pieces of a global checkpoint will be a uniform size. Therefore, future C/R development should work towards efficiently using both storage tiers by assigning larger data regions to DRAM.

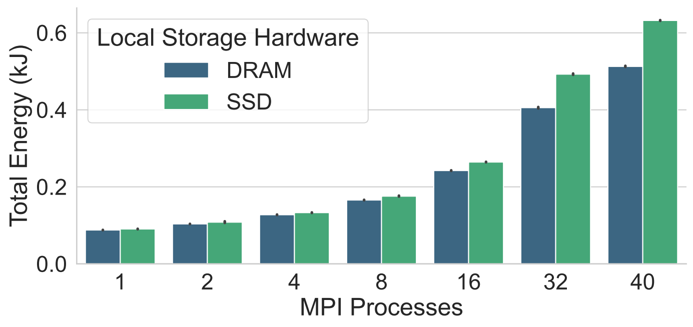
Figures 3.4 and 3.5 compare the overall throughput and energy consumption of both C/R libraries when VELOC writes to DRAM locally. Both strategies use the PFS as their persistent storage tier. These results show that GIO consumes on average $\approx 4\times$ the amount of energy VELOC does. Similar to energy consumption of the local checkpointing phase, we see that as the data assigned per rank doubles, so does the resulting energy consumption.

3.4.3 Strong Scalability

For our strong scalability tests, we evaluate workloads using N $(1 \rightarrow 40)$ processes on aggregate problem sizes of 40 and 80 GiB. Since we are consistently writing over 1 GiB, GIO uses POSIX read/writes and VELOC continues to use mmap/writes.



(a) 1 GB per Rank



(b) 2 GB per Rank

Figure 3.3: Weak scalability of energy consumption for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Lower is better.

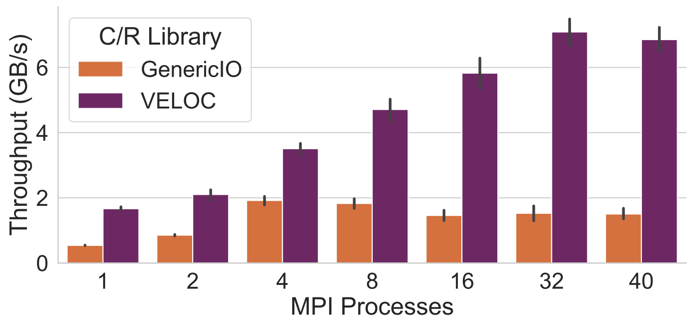
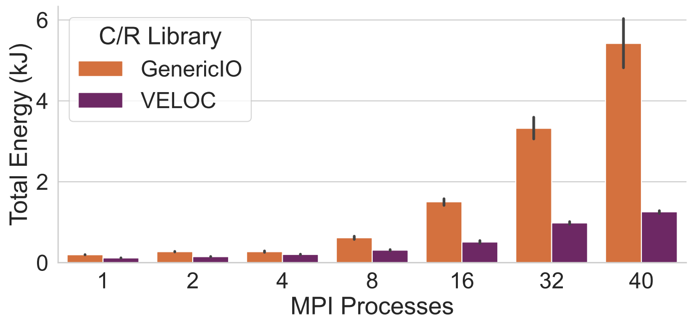


Figure 3.4: Weak scalability of total I/O throughput (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Higher is better.

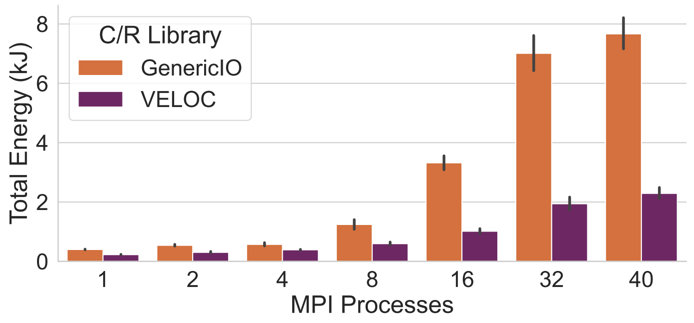
To evaluate how parallelism affects the throughput and energy of the local storage tiers, we analyze these metrics through a series of strong scalability experiments in Figures 3.6 and 3.7, respectively. In these experiments, we see the same overall trends in the weak scaling experiments: DRAM provides (1) higher throughput and (2) lower energy consumption than the SSD during local checkpointing. This similarity further supports the idea that better throughput directly leads to more efficient energy usage. Therefore, future efforts to improve energy efficiency should focus on ensuring high throughput performance rather than throttling.

Figures 3.8 and 3.9 compare the throughput and energy consumption of the C/R libraries, respectively. In these experiments, we show that asynchronous file-per-process strategies continue to provide the highest throughput, without consuming extra energy. Even though the asynchronous strategy requires more operations to (1) write the checkpoint to a local storage device, and (2) buffering the checkpoint to transfer externally, this is more energy efficient than directly interacting with the PFS. There are a few possible explanations for this: (1) writing to a local storage tier and then asynchronously flushing spreads out writes to the PFS such that not all processes (or threads) are competing for I/O servers at the same time, thereby improving throughput and energy consumption; and (2) file aggregation suffers serialization at the I/O server level (in the case of poorly matched I/O strategies and stripe settings) due to false sharing.

In figures 3.9a and 3.9b, after 8 or more processes are used, the energy consumed by GIO begins to rise again, despite the fact that the size per process is shrinking. We conclude that uncoordinated file aggregation, such as the case in POSIX writes, coupled with poor stripe settings causes CPUs to spend time (and therefore energy) waiting for access to the storage servers due to



(a) 1 GB per Rank



(b) 2 GB per Rank

Figure 3.5: Weak scalability of total energy consumption (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Lower is better.

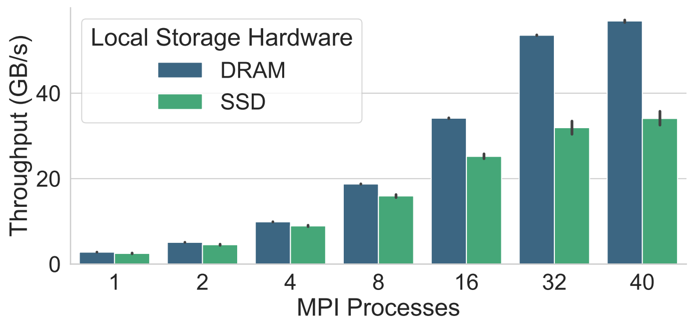
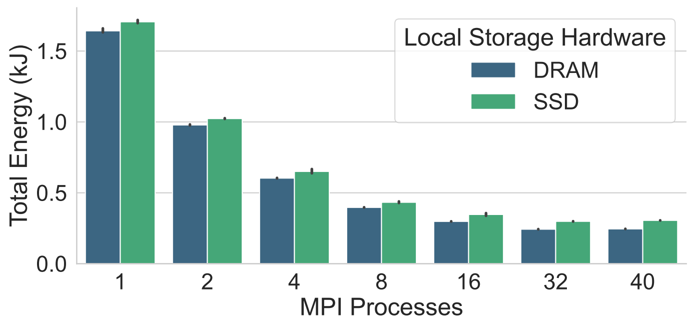
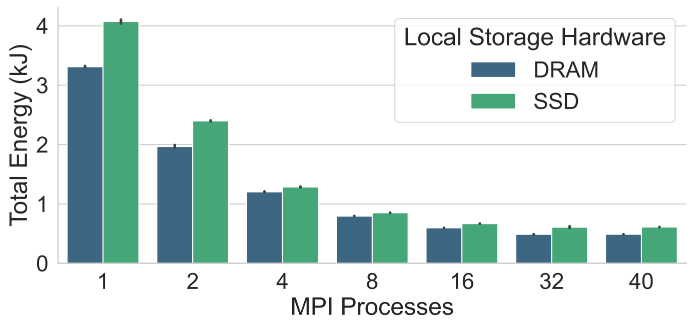


Figure 3.6: Strong scalability of I/O throughput for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Higher is better.



(a) 40 GB Problem Size



(b) 80 GB Problem Size

Figure 3.7: Strong scalability of energy consumption for VELOC local phase: capturing checkpoints to node-local storage using blocking writes. Lower is better.

false sharing. Compared to VELOC, which is using a file-per-process strategy, energy consumption continues to decrease as more processes are added. Therefore, we conclude that the flush strategy has an impacting effect on energy consumption.

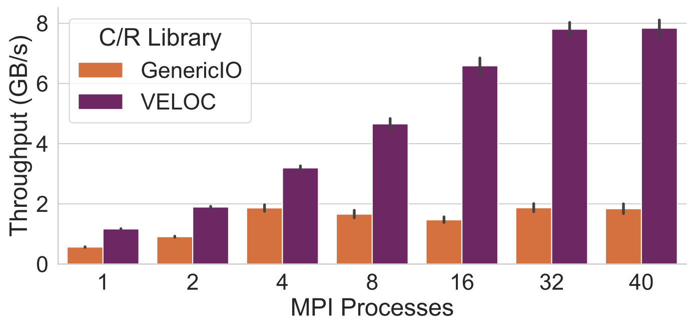


Figure 3.8: Strong scalability of total I/O throughput (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Higher is better.

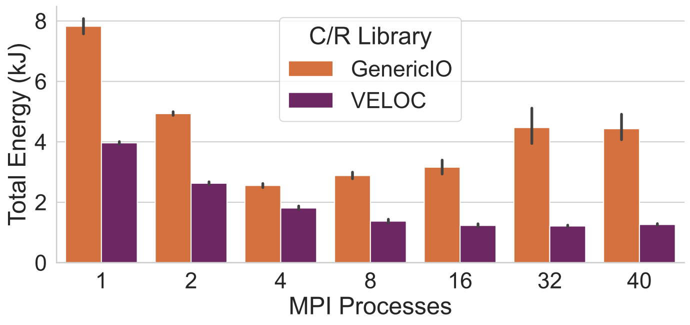
3.5 Future Works and Areas of Improvement

There are numerous ways this work can be expanded and improved upon.

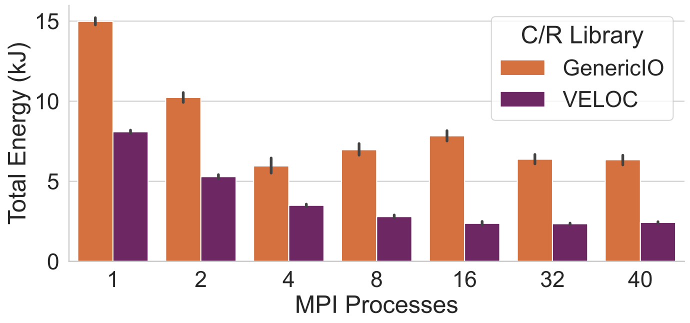
First, the experiments used to characterize the synchronous checkpointing library, GIO, were done using the POSIX implementation of C/R, which relies on uncoordinated file aggregation that issues regular read/write calls. While simple to set up and use, this leads to significant performance degradation as a side effect of file striping. Other techniques based on MPI I/O collectives are available in GIO and perform I/O re-organization and aggregation to better match the underlying stripe settings. This may improve performance and scalability at the expense of more energy consumption. In this context, our own previous work [22] also considers file aggregation in the context of asynchronous checkpointing, which introduces additional trade-offs. Due to time considerations, we did not explore such a comparison in this work, but plan to do so in the future.

Secondly, we plan to expand upon the methods of energy monitoring and recommendations we present. We aim to utilize a local cluster with the ability to physically measure the energy consumption of storage devices, allowing for more detailed recommendations about hardware energy-efficiency. By having this specific data, one could describe how the load of C/R software differences affect the hardware in use.

Finally, in this work our goal was to isolate the energy consumption of checkpointing from any other external influence in synthetic benchmarks. However, in a real-life application, the overlap between asynchronous multi-level checkpointing strategies and the application runtime may lead to interesting interference patterns that affect the energy consumption beyond the individual behaviors in isolation. Thus, in future work, we aim to study such effects in the context of real-life applications.



(a) 40 GB Problem Size



(b) 80 GB Problem Size

Figure 3.9: Strong scalability of total energy consumption (until checkpoints written to PFS): direct writes measured for GIO, local phase followed flush phase (async writes to PFS) measured for VELOC. Lower is better.

3.6 Conclusions

In this paper, we evaluate two different C/R libraries and their various configurations in order to compare the energy consumption of multi-level asynchronous C/R and single-level synchronous C/R. We use this information to better help application and C/R developers identify configurations and areas of C/R that contribute to high energy overheads. Overall, we find that throughput heavily impacts energy consumption. Thus, asynchronous C/R to DRAM using file-per-process flushing strategies utilize the least amount of energy. We summarize our main observations below.

We note that energy consumption is directly tied to throughput. Thus, C/R should focus on providing high throughput in order to achieve low energy costs. In the context of asynchronous checkpointing, using DRAM as the local storage tier is shown to provide the highest throughput as compared to SSDs. However, in real-world scientific computing it is a heavily contended resource. Therefore, asynchronous C/R should focus on *efficiently* utilizing DRAM to balance the use of shared resources that reduce contention between applications and C/R to maximize throughput, thereby lowering energy consumption.

Furthermore, we show that flushing strategies like synchronous file aggregation have a significant impact on energy consumption. Our results at modest scale show that file-per-process strategies provide the highest throughput and consume the least amount of energy. However, file-per-process strategies may encounter I/O bottlenecks at scale due to metadata bottlenecks, which typically is not handled in a scalable fashion by many PFS implementations. Thus, considering the trade-offs between performance and energy efficiency that file aggregation introduces at scale both for synchronous and asynchronous checkpointing are non-trivial and need to be studied in further detail.

Encouraged by these observations, we plan to explore in future work several follow-up directions, as outlined in Section 3.5.

Chapter 4

Conclusion

This Honor's Thesis, titled "Green HPC: Optimizing the Energy Consumption of Large Data Systems," has delved into two critical aspects of high-performance computing (HPC) in the exascale era: energy-efficient resilience strategies and power optimization in data compression. By targeting the software stack, the research has demonstrated the significant potential for energy efficiency improvements in large-scale computing systems, paving the way for a more sustainable future in HPC.

The first part of the thesis focused on the trade-offs and impact of various factors on the performance, scalability, and energy efficiency of checkpoint/restart (C/R) techniques. Through the in-depth study of state-of-the-art C/R libraries, VELOC and GenericIO, the research has provided valuable insights into optimizing the energy consumption of resilience strategies. It has emphasized the importance of high throughput and asynchronous C/R techniques, as well as the need for continuous innovation in resilience methods to address the growing demands of exascale systems.

The second part of the thesis investigated energy consumption optimization for lossy data compressors and data writing in HPC systems. By exploring power consumption patterns of the SZ and ZFP lossy compressors under various conditions and developing a power model for lossy compression, the research has demonstrated energy savings of 13% on average for 512GB I/O. Furthermore, it has presented a tuning methodology that reduces energy overhead by an average of 14.3%, contributing to more energy-efficient lossy data compression and I/O in HPC systems.

These findings contribute to the ongoing efforts to make HPC systems greener and more sustainable, ensuring that the benefits of exascale computing are realized while minimizing the impact on the environment and managing limited power budgets. The research highlights the importance of software stack optimizations in achieving significant energy efficiency gains in HPC systems and underscores the need for a holistic approach to energy optimization that considers various aspects of the computing ecosystem.

As we look towards the future, there are numerous promising directions for further research and innovation in energy-efficient computing. By developing advanced resilience strategies, data compression algorithms, dynamic tuning techniques, cross-layer optimizations, and advocating for sustainable computing policies, we can continue to advance the state-of-the-art in green HPC. Moreover, exploring the impact of emerging heterogeneous architectures on energy efficiency and developing specialized algorithms and methods optimized for new hardware platforms can lead to further breakthroughs in sustainable computing.

In addition, fostering collaboration between researchers, HPC centers, funding agencies, and industry partners is essential to ensure the widespread adoption of energy-efficient computing practices. The development of comprehensive benchmarking suites and standardization guidelines can facilitate the comparison and validation of different approaches, promoting innovation and collaboration in the field.

Ultimately, this thesis demonstrates the potential of software-based solutions to address the energy challenges of large-scale computing and emphasizes the crucial role of the software stack in achieving energy efficiency gains. By continuing to pursue research in this domain, we can contribute to a more sustainable future for high-performance computing systems, unlocking the full potential of exascale computing while minimizing its environmental impact.

Bibliography

[1] top500 results for june 2022, 2022.

[2] Muhammad Alfian Amrizal and Hiroyuki Takizawa. Optimizing energy consumption on hpc systems with a multi-level checkpointing mechanism. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–9, 08 2017.

[3] Ann S. Almgren, John B. Bell, Mike J. Lijewski, Zarija Lukic, and Ethan Van Andel. Nyx: A massively parallel amr code for computational cosmology. *The Astrophysical Journal*, 765(1):39, feb 2013.

[4] Muhammad Alfian Amrizal and Hiroyuki Takizawa. Optimizing energy consumption on hpc systems with a multi-level checkpointing mechanism. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–9, 2017.

[5] Jason Ansel, Kapil Arya, and Gene Cooperman. Dmtcp: Transparent checkpointing for cluster computations and the desktop. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–12, 2009.

[6] John Bent, Bradley W. Settlemyer, and Gary Grider. Serving data to the lunatic fringe: The evolution of hpc storage. *login Usenix Mag.*, 41, 2016.

[7] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 15:181, 2008.

[8] Mohamed-Slim Bouguerra, Thierry Gautier, Denis Trystram, and Jean-Marc Vincent. A flexible checkpoint/restart model in distributed systems. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, pages 206–215, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[9] Raghunath Raja Chandrasekar, Akshay Venkatesh, Khaled Hamidouche, and Dhabaleswar K. Panda. Power-check: An energy-efficient checkpointing framework for hpc clusters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 261–270, 2015.

[10] A. Colin Cameron and Frank A.G. Windmeijer. An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics*, 77(2):329–342, 1997.

[11] Julita Corbalan, Oriol Vidal, Lluis Alonso, and Jordi Aneas. Explicit uncore frequency scaling for energy optimisation policies with ear in intel architectures. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 572–581, 2021.

[12] Pawel Czarnul, Jerzy Proficz, and Adam Krzywaniak. Energy-aware high-performance computing: Survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019:8348791, Apr 2019.

[13] John T Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future generation computer systems*, 22(3):303–312, 2006.

[14] Daniel Dauwe, Rohan Jhaveri, Sudeep Pasricha, Anthony A. Maciejewski, and Howard Jay Siegel. Optimizing checkpoint intervals for reduced energy use in exascale systems. In *2017 Eighth International Green and Sustainable Computing Conference (IGSC)*, pages 1–8, 2017.

[15] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. Rapl: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, page 189–194, New York, NY, USA, 2010. Association for Computing Machinery.

[16] S. Di, D. Tao, X. Liang, and F. Cappello. Efficient lossy compression for scientific data based on pointwise relative error bound. *IEEE Transactions on Parallel and Distributed Systems*, 30:331–345, 2019.

[17] Mohammed el Mehdi Diouri, Olivier Glück, Laurent Lefèvre, and Franck Cappello. Ecofit: A framework to estimate energy consumption of fault tolerance protocols for hpc applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 522–529, 2013.

[18] Nosayba El-Sayed and Bianca Schroeder. To checkpoint or not to checkpoint: Understanding energy-performance-i/o tradeoffs in hpc checkpointing. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 93–102, 2014.

[19] Nosayba El-Sayed and Bianca Schroeder. Understanding practical tradeoffs in hpc checkpoint-scheduling policies. *IEEE Transactions on Dependable and Secure Computing*, 15(2):336–350, 2018.

[20] Daniel Ellsworth, Tapasya Patki, Martin Schulz, Barry Rountree, and Allen Malony. A unified platform for exploring power management strategies. In *Proceedings of the 4th International Workshop on Energy Efficient Supercomputing*, E2SC '16, page 24–30. IEEE Press, 2016.

[21] Dmitry Duplyakin et al. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[22] Mikaila J. Gossman, Bogdan Nicolae, Jon C. Calhoun, Franck Cappello, and Melissa C. Smith. Towards aggregated asynchronous checkpointing. *ArXiv*, abs/2112.02289, 2021.

[23] Salman Habib and et al. HACC: extreme scaling and performance across diverse architectures. *Communications of the ACM*, 60(1):97–104, 2016.

[24] Salman Habib, Adrian Pope, Hal Finkel, Nicholas Frontiere, Katrin Heitmann, David Daniel, Patricia Fasel, Vitali Morozov, George Zagaris, Tom Peterka, Venkatram Vishwanath, Zarija Lukić, Saba Sehrish, and Wei keng Liao. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy*, 42:49–65, jan 2016.

[25] Florin Isaila, Javier García, Jesús Carretero, Robert B. Ross, and Dries Kimpe. Making the case for reforming the i/o software stack of extreme-scale systems. *Adv. Eng. Softw.*, 111:26–31, 2017.

[26] Abhishek Jaiantilal, Yifei Jiang, and Shivakant Mishra. Modeling cpu energy consumption for energy efficient scheduling. In *Proceedings of the 1st Workshop on Green Computing*, GCM '10, page 10–15, 2010.

[27] JE Kay and et al. The community earth system model (CESM), large ensemble project. *Bulletin of the American Meteorological Society*, 96(8):1333–1349, 2015.

[28] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Shaomeng Li, Hanqi Guo, Zizhong Chen, and Franck Cappello. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 438–447, 2018.

[29] Xin Liang, Sheng Di, Dingwen Tao, Sihuan Li, Bogdan Nicolae, Zizhong Chen, and Franck Cappello. Improving performance of data dumping with lossy compression for scientific simulation. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11, 2019.

[30] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, 2014.

[31] Yuanlai Liu, Zhengchun Liu, Rajkumar Kettimuthu, Nageswara Rao, Zizhong Chen, and Ian Foster. Data transfer between scientific facilities – bottleneck analysis, insights and optimizations. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 122–131, 2019.

[32] P. Llopis, M.F. Dolz, J.G. Blas, and et al. Analyzing the energy consumption of the storage data path. *Journal of Supercomputing*, 72:4089–4106, 2016.

[33] Huizhang Luo, Dan Huang, Qing Liu, Zhenbo Qiao, Hong Jiang, Jing Bi, Haitao Yuan, Mengchu Zhou, Jinzhen Wang, and Zhenlu Qin. Identifying latent reduced models to precondition lossy compression. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 293–302, 2019.

[34] MATLAB. *version 9.11.0 (R2021b)*. The MathWorks Inc., Natick, Massachusetts, 2021.

[35] Zheng Miao, Jon Calhoun, and Rong Ge. Energy analysis and optimization for resilient scalable linear systems. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 24–34, 2018.

[36] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, ICS '02, page 35–44, New York, NY, USA, 2002. Association for Computing Machinery.

[37] José Miguel Montañana Aliaga, Alexey Cheptsov, and Antonio Hervás. Towards energy efficient computing based on the estimation of energy consumption. In Michael M. Resch, Manuela Wossough, Wolfgang Bez, Erich Focht, and Hiroaki Kobayashi, editors, *Sustained Simulation Performance 2019 and 2020*, pages 21–33, Cham, 2021. Springer International Publishing.

[38] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC '10: The 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1:1–1:11, New Orleans, USA, 2010.

[39] M. Morán, J. Balladini, D. Rexachs, and E. Luque. Prediction of energy consumption by checkpoint/restart in hpc. *IEEE Access*, 7:71791–71803, 2019.

[40] M. Morán, J. Balladini, D. Rexachs, and E. Luque. Prediction of energy consumption by checkpoint/restart in hpc. *IEEE Access*, 7:71791–71803, 2019.

[41] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. Veloc: Towards high performance adaptive asynchronous checkpointing at large scale. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 911–920, 2019.

[42] Takafumi Saito, Kento Sato, Hitoshi Sato, and Satoshi Matsuoka. Energy-aware i/o optimization for checkpoint and restart on a nand flash memory system. In *Proceedings of the 3rd Workshop on Fault-Tolerance for HPC at Extreme Scale*, FTXS '13, page 41–48, New York, NY, USA, 2013. Association for Computing Machinery.

[43] Martin Schulz, Dieter Kranzlmüller, Laura Brandon Schulz, Carsten Trinitis, and Josef Weidendorfer. On the inevitability of integrated hpc systems and how they will change hpc system operations. In *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, HEART '21, New York, NY, USA, 2021. Association for Computing Machinery.

[44] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. Energy efficiency features of the intel skylake-sp processor and their impact on performance. In *2019 International Conference on High Performance Computing Simulation (HPCS)*, pages 399–406, 2019.

[45] Scientific Data Reduction Benchmark. `https://sdrbench.github.io/`. Online.

[46] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In José M. Laginha M. Palma, Michel Daydé, Osni Marques, and João Correia Lopes, editors, *High Performance Computing for Computational Science – VECPAR 2010*, pages 1–25, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[47] Esteban Stafford and José Luis Bosque. Performance and energy task migration model for heterogeneous clusters. *The Journal of Supercomputing*, 77(9):10053–10064, Sep 2021.

[48] Li Tan, Zizhong Chen, and Shuaiwen Leon Song. Scalable energy efficiency with resilience for high performance computing systems: A quantitative methodology. *ACM Trans. Archit. Code Optim.*, 12(4), nov 2015.

[49] Dingwen Tao, Sheng Di, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1129–1139, 2017.

[50] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 157–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[51] Shu-Mei Tseng, Bogdan Nicolae, George Bosilca, Emmanuel Jeannot, Aparna Chandramowlishwaran, and Franck Cappello. Towards portable online prediction of network utilization using mpi-level monitoring. In *EuroPar'19 : 25th International European Conference on Parallel and Distributed Systems*, pages 47–60, Goettingen, Germany, 2019.

[52] Shu-Mei Tseng, Bogdan Nicolae, Franck Cappello, and Aparna Chandramowlishwaran. Demystifying asynchronous i/o interference in hpc applications. *The International Journal of High Performance Computing Applications*, 35, 2021.

[53] Lipeng Wan, Qing Cao, Feiyi Wang, and Sarp Oral. Optimizing checkpoint data placement with guaranteed burst buffer endurance in large-scale hierarchical storage systems. *Journal of Parallel and Distributed Computing*, 100:16–29, 2017.

[54] Grant Wilkins and Jon C. Calhoun. Modeling power consumption of lossy compressed i/o for exascale hpc systems. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1118–1126, 2022.

[55] Grant Wilkins and Jon C. Calhoun. Modeling power consumption of lossy compressed i/o for exascale hpc systems. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1118–1126, 2022.

[56] Grant Wilkins, Mikaila J. Gossman, Bogdan Nicolae, Melissa C. Smith, and Jon C. Calhoun. Analyzing the energy consumption of synchronous and asynchronous checkpointing strategies. In *2022 IEEE/ACM Third International Symposium on Checkpointing for Supercomputing (SuperCheck)*, pages 1–9, 2022.

[57] Kai Zhao, Sheng Di, Xin Liang, Sihuan Li, Dingwen Tao, Zizhong Chen, and Franck Cappello. Significantly improving lossy compression for hpc datasets with second-order prediction and parameter optimization. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, page 89–100, New York, NY, USA, 2020. Association for Computing Machinery.

[58] Ziliang Zong, Rong Ge, and Qijun Gu. Marcher: A heterogeneous system supporting energy-aware high performance computing and big data analytics. *Big Data Research*, 8:27–38, 2017. Tutorials on Tools and Methods using High Performance Computing resources for Big Data.