OPTIMIZING VISUALIZATION PERFORMANCE ON POWER-CONSTRAINED SUPERCOMPUTERS

by

STEPHANIE ALYSSA LABASAN

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 2019

DISSERTATION APPROVAL PAGE

Student: Stephanie Alyssa Labasan

Title: Optimizing Visualization Performance on Power-Constrained Supercomputers

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Henry Childs	Chair
Allen Malony	Core Member
Boyana Norris	Core Member
Nicholas Proudfoot	Institutional Representative
Barry Rountree	Outside Member

and

Janet Woodruff-Borden	Vice Provost and Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded March 2019

© 2019 Stephanie Alyssa Labasan
All rights reserved.

DISSERTATION ABSTRACT

Stephanie Alyssa Labasan

Doctor of Philosophy

Department of Computer and Information Science

March 2019

Title: Optimizing Visualization Performance on Power-Constrained Supercomputers

Power consumption is widely regarded as one of the biggest challenges to reaching the next generation of high performance computing (HPC). On future supercomputers, power will be a limited resource. This constraint will affect the performance of both simulation and visualization workloads. Understanding how a particular application behaves under a power limit is critical to making better use of the limited power. In this research, we focus specifically on visualization and analysis applications, which are an important component in HPC. Visualization algorithms merit special consideration, since they are more data intensive in nature than traditional HPC programs, such as simulation codes. We explore the power and performance tradeoffs for several common algorithms under different configurations, and understand how power constraints will affect execution behaviors. We then demonstrate that we can gain additional performance by redistributing power based on performance predictions provided by the visualization algorithm.

This dissertation includes previously published co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: Stephanie Alyssa Labasan

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
University of the Pacific, Stockton, CA, USA

DEGREES AWARDED:

Doctor of Philosophy, Computer and Information Science, 2019, University of Oregon
Master of Science, Computer and Information Science, 2016, University of Oregon
Bachelor of Science, Computer Engineering, 2013, University of the Pacific

AREAS OF SPECIAL INTEREST:

Power-Constrained Supercomputing
High Performance Computing
Scientific Visualization

PROFESSIONAL EXPERIENCE:

Lawrence Scholar, Lawrence Livermore National Laboratory, 2016–Present
HPC Power Management Researcher, Intel Corporation, 2015–2016
Graduate Research Assistant, University of Oregon, 2013–2015
Computation Student Intern, Lawrence Livermore National Laboratory, Summer 2013

GRANTS, AWARDS AND HONORS:

Area Exam, Passed With Distiction, University of Oregon, 2016

Livermore Graduate Scholar Fellowship, Lawrence Livermore National Laboratory, 2016

PUBLICATIONS:

S. Labasan, M. Larsen, H. Childs, and B. Rountree. "Evaluating Predictive and Adaptive Power Strategies for Optimizing Visualization Performance," In IEEE Transactions on Parallel and Distributed Systems (TPDS), 2019. (In Preparation)

S. Labasan, M. Larsen, H. Childs, and B. Rountree. "Power and Performance Tradeoffs for Visualization Algorithms," In IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, 2019. (To Appear)

S. Labasan, M. Larsen, H. Childs, and B. Rountree. "PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance," In Eurographics Symposium on Parallel Graphics and Visualization (EGPGV), Barcelona, Spain, 2016.

S. Labasan, M. Larsen, and H. Childs. "Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm," In IEEE Symposium on Large Data Analysis and Visualization (LDAV), Chicago, IL, 2015.

M. Larsen, **S. Labasan**, P. Navrátil, J.S. Meredith, and H. Childs. "Volume Rendering Via Data-Parallel Primitives," In Eurographics Symposium on Parallel Graphics and Visualization (EGPGV), Cagliari, Sardinia, Italy, 2015.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Hank Childs for his technical guidance and unwavering support and encouragement throughout the Ph.D. process. Hank taught me how to write readable research papers and give effective presentations, and helped me find simplicity in difficult challenges. Thank you for giving me the freedom to explore different research ideas and for providing constructive feedback along the way.

I would like to thank my committee members, Dr. Allen Malony, Dr. Boyana Norris, and Dr. Nicholas Proudfoot. Your timely feedback and suggestions have greatly improved this research.

I wish to thank Dr. Barry Rountree for introducing me to academic research in high performance computing and the idea of pursuing a graduate degree. Thank you for pushing me beyond my comfort zone and for providing opportunities for my own professional development and programming skills.

I am grateful to my CDUX colleagues who contributed to these memorable years of graduate school. Thank you all for the late night bunker sessions, paper feedback, technical discussions, and fun-filled activities.

The author wrote this dissertation in support of requirements for the degree Doctor of Philosophy in Computer and Information Science at the University of Oregon in Eugene, OR. The research is funded in part by the LLNL Graduate Scholars Program, and is not a deliverable for any United States government agency. The views and opinions expressed are those of the author, and do not state or reflect those of the United States government or Lawrence Livermore National Security, LLC.

Some of this research was funded in part by the DOE Early Career Award, Contract No. DE-SC0010652, Program Manager Lucy Nowell. Some of this work was supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, Program Manager Lucy Nowell. Some of this research was performed under the auspices of the U.S. Department of Energy by by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-689101, LLNL-CONF-727082, and LLNL-CONF-753659).

Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

I dedicate this to my parents and my sister, who have been my biggest supporters throughout my life. You instilled in me the importance of diligence and hard work, giving me the confidence to pursue my dreams. Thank you to my husband for your unfailing love, support and understanding during this process. You kept me grounded and helped keep things in perspective. This achievement would not have been possible without all of you.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1. Motivation 1
1.2. Research Goals and Approaches 3
1.3. Dissertation Outline 5
1.4. Co-Authored Material 6

II. POWER AND PERFORMANCE TRADEOFFS UNDER REDUCED CLOCK FREQUENCIES 7

2.1. Motivation 7
2.2. Related Work 9
2.3. Benchmark Tests 11
2.4. Experimental Setup 12
2.5. Results 20
2.6. Summary 33

III.POWER AND PERFORMANCE TRADEOFFS UNDER POWER LIMITS 35

3.1. Motivation 36
3.2. Related Work 38
3.3. Overview of Visualization Algorithms 39
3.4. Experimental Overview 41
3.5. Definition of Metrics 44

Chapter Page

3.6. Results 46

3.7. Summary of Findings 59

3.8. Conclusion 61

IV.POWER-AWARE RUNTIME SYSTEM FOR VISUALIZATION 63

4.1. Motivation 63

4.2. HPC and Power Overview 64

4.3. Related Work 66

4.4. Our Approach for Adaptive Power Scheduling 69

4.5. Study Overview 73

4.6. Results 77

4.7. Conclusion 84

V. EVALUATING TECHNIQUES FOR SCHEDULING POWER 87

5.1. Introduction 87

5.2. Background and Related Work 89

5.3. Power-Aware Runtime Systems 91

5.4. Experimental Overview 94

5.5. Results 100

5.6. Conclusion and Future Work 108

VI.CONCLUSION AND FUTURE DIRECTIONS 112

6.1. Synthesis 112

6.2. Future Directions 115

REFERENCES CITED 118

LIST OF FIGURES

Figure Page

1. Performance results for compute-bound and memory-bound micro-benchmarks under reduced CPU clock frequencies. 12

2. Images of the Enzo and Nek5000 data sets used in the initial variation study in Chapter II. 16

3. Performance slowdown and energy savings under reduced CPU clock frequencies for data sets of varying sizes and layouts. 25

4. Performance slowdown and energy savings under reduced CPU clock frequencies for the isosurfacing MPI implementation. 25

5. Performance slowdown and energy savings under reduced CPU clock frequencies for varying levels of OpenMP thread concurrency. 26

6. Performance slowdown and energy savings under reduced CPU clock frequencies for varying levels of MPI task concurrency. . . . 26

7. Renderings of the representative set of visualization algorithms explored in the thorough study in Chapter III. 38

8. CPU frequency, instructions per cycle, and last level cache miss rate for the eight algorithms explored in Chapter III. 50

9. Number of data elements processed per second under varying power limits for cell-centered algorithms. 54

10. Algorithms that increase in instructions per cycle with increasing data set size: slice, contour, isovolume, threshold, and spherical clip. 56

11. Algorithms that increase in instructions per cycle with decreasing data set size: volume rendering. 57

12. Algorithms that have no change in instructions per cycle with changes in data set size: particle advection and ray tracing. 57

13. Rendered images of the CloverLeaf data set from three different camera positions. 75

Figure Page

14. Speedup results for a single rendering workload using the Min predictive power scheduling strategy defined in Chapter IV. 79

15. Comparing speedup results for a single rendering workload using five different predictive power scheduling strategies. 81

16. Comparing speedup results when varying across the different rendering workloads and predictive power scheduling strategies outlined in Chapter IV. 82

17. Comparing speedups when varying across rendering workloads and predictive power scheduling strategies at higher node concurrency. 85

18. Rendered images of a contour of the pressure at the 200th cycle for the CloverLeaf hydrodynamics proxy application. 94

19. Total rendering times across all visualization cycles for each rank. . . . 97

20. Comparing the performance of adaptive and predictive power scheduling strategies for Workload A defined in Chapter V. . . . 102

21. Comparing the performance of adaptive and predictive power scheduling strategies for Workload B defined in Chapter V. . . . 105

22. Comparing the performance of adaptive and predictive power scheduling strategies for Workload C defined in Chapter V. . . . 106

23. Comparing the performance of adaptive and predictive power scheduling strategies for Workload D defined in Chapter V. . . . 107

24. Comparing the performance of adaptive and predictive power scheduling strategies for a single workload at high concurrencies. . 109

25. Difference in performance between the predictive and adaptive power scheduling strategies at varying power limits and concurrencies. 110

LIST OF TABLES

Table Page

1. Performance slowdown for compute-bound and memory-bound micro-benchmarks as the CPU clock frequency is reduced. 13

2. Results for the baseline isosurfacing OpenMP implementation on the Enzo data set as the CPU clock frequency is reduced. 23

3. Results for the baseline isosurfacing OpenMP implementation on the randomized Enzo data set as the clock frequency is reduced. 24

4. Comparing performance metrics at 1.6 GHz clock frequency for the baseline isosurfacing OpenMP implementation across all data sets defined in Chapter II. 24

5. Results from the baseline isosurfacing MPI implementation on the Enzo data set. 27

6. Results from the baseline isosurfacing MPI implementation on the randomized Enzo data set. 27

7. Comparing performance metrics for the baseline isosurfacing OpenMP and MPI implementations at varying concurrency levels. . . . 30

8. Results for the VTK isosurfacing OpenMP implementation on the Enzo data set as the CPU clock frequency is reduced. 31

9. Results for the baseline isosurfacing OpenMP implementation on a hardware architecture with a shared clock frequency between the memory and compute units. 32

10. Performance of the VTK-m contour algorithm with a data set size of 128^3 under reduced processor power limits. 47

11. Performance of all algorithms explored in Chapter III using a data set size of 128^3 and reduced power limits. 49

12. Performance of all algorithms explored in Chapter III using a data set size of 256^3 and reduced power limits. 55

Table Page

13. Power scheduling strategy parameters used in the predictive approach in PaViz. 71

14. Rendering workloads used by the predictive approach implemented in PaViz. 76

15. Node-level and processor-level power budgets used by the adaptive and predictive runtime systems. 96

16. Rendering workloads used in the adaptive and predictive power scheduling study in Chapter V. 98

17. Comparing power allocation decisions by the adaptive and predictive runtime systems for a single rendering workload. The per-node power decisions assume a processor power cap of 60W. 100

CHAPTER I

INTRODUCTION

1.1 Motivation

Power is a key challenge in achieving the next generation of high performance computing (HPC). At the start of this decade, scaling current technologies to higher concurrency would have produced a machine requiring gigawatts or more [12, 14, 63]. Such a power-hungry machine results in unfeasible operational costs. As a result, power has become a severely limited resource moving forward. One major change in response to this limitation was a shift towards many-core architectures, which have higher efficiency. That said, the premise of this research is that hardware changes are not enough, i.e., achieving an exaflop within specified power constraints will require innovation across all aspects of HPC, including runtime infrastructures, simulation, and visualization codes.

Current supercomputer designs assume sufficient power will be available to run all compute nodes concurrently at their maximum thermal design point (TDP). Said another way, TDP is the theoretical maximum power a given node will ever consume. As power requirements for supercomputers move into the range of dozens of megawatts, the strategy of allocating power for all nodes to run at TDP becomes untenable. Very few applications run at TDP, and provisioning very large systems as if most did both wastes power capacity and unnecessarily constrains the size of the supercomputer.

Overprovisioning [57], short for hardware overprovisioning, is one solution to improve power utilization. In such a design, we increase the compute capacity (i.e., number of nodes) of the system, but, in order to not exceed the system power allocation, not every node will be able to run at TDP simultaneously.

Consider the example of the Vulcan supercomputer at Lawrence Livermore National Laboratory. It was allocated for 2.4 MW assuming all 24,576 nodes consumed TDP, but the vast majority of applications running on that machine did not exceed 60% of the allocated power (1.47 MW average system power consumption) [58]. Thus, the strategy of allocating TDP to every node failed to take advantage of nearly 1 MW of power on average. An overprovisioned approach system would contain about 34,000 nodes (40% increase) consuming some amount of power less than TDP. With such a system, the machine consumes all allocated power and reduces trapped capacity [69].

For overprovisioning to be a success, it must be complemented with a scheme to limit nodes' power usage, to ensure the total provisioned power is never exceeded. One way to accomplish this is to uniformly cap the power available to each node, e.g., each node can use only up to 60% of its TDP. The result of applying such a power cap is that the processor operating frequency is reduced. The effect of reduced CPU frequency is variable; applications dominated by computation will slow down proportionally, while applications dominated by memory accesses may be unaffected. Despite the performance degradation for individual jobs, this strategy would lead to better power utilization and greater overall throughput.

Uniform power allocations per node, however, is a sub-optimal strategy. The runtime behaviors of distributed applications can be highly variable across nodes. The nodes assigned the largest amount of work become a bottleneck and determine the overall performance of the application. On the other hand, nodes that are assigned the smallest amount of work finish quickly and sit idle until the other nodes have completed execution. A better strategy is to dynamically assign power to where it will do the most good. This is a non-trivial problem. In an ideal scenario, we would assign the power such that all nodes finish executing at the same time despite varying workloads.

Overprovisioned systems lend themselves to multiple levels of optimization. At the job scheduling level, individual job power bounds are allocated to optimize throughput and/or turn-around time [58]. Alternatively, there are dynamic optimizations to individual jobs that may be realized by rebalancing power and changing node configuration at runtime [2, 22, 47].

While much previous research has studied adapting power usage for simulations, this dissertation considers how visualization routines will need to adapt to a power-limited environment, where compute nodes will be limited in their power usage. The motivation for studying visualization is two-fold. First, visualization is a critical component in the scientific discovery process. Additionally, visualization is moving to an in situ workflow, where the visualization will run concurrently with the simulation. Therefore, optimizing the performance of the visualization component is beneficial, since the visualization can use significant resources on the HPC system. Second, visualization workloads are different than the typically HPC applications like simulation codes, since they are more data intensive. For both of these reasons, visualization workloads merit special considering in this power-limited environment.

1.2 Research Goals and Approaches

The central question that this dissertation addresses is: *How can we optimize the performance of scientific visualization algorithms in a power-constrained environment?"* This main idea can be further broken down into the following research goals.

The first research goal is to explore how tunable input parameters within visualization algorithms impact performance, energy, and power usage. On future supercomputers, visualization routines will need to adapt to the power-limited environment, where compute nodes will be limited in their power usage. Current strategies for power management are not aware of what phase (e.g., computational, memory, I/O, etc.) the application is in. Understanding the impacts of particular input parameters is critical in making a decision between optimizing performance and staying under a specified power limit. Since the space of tunable parameters can grow exponentially large, this information forms the basis for generating performance models relating execution time, energy usage, and power consumption. Understanding the specific execution behaviors of visualization algorithms will enable us to develop better power-aware strategies for optimal performance and power usage.

The second research goal is to improve performance by exploiting the variation in visualization workloads when dynamically reallocating power The scientific visualization community is moving towards in situ, where data is processed alongside the simulation (as opposed to processed post hoc). For in situ strategies, power will need to be shared in some fashion between the scientific simulation and visualization [61]. Exploring power management strategies for visualization routines will be critical in improving the overall turn-around time for in situ workflows. Visualization workloads can be imbalanced, and allocating power relative to the amount of work each compute node has is a good strategy for improving performance. We use an existing visualization-specific performance model to determine how best to share the power across nodes in order to optimize performance. This shows the need for additional performance models to improve performance of other visualization algorithms.

1.3 Dissertation Outline

Putting it all together, scientific visualization is a key component of the scientific discovery process. It enables the exploration and analysis of scientific data, and the ability to communicate findings through a comprehensible image. Visualization at exascale will be challenging due to constraints in power usage. Under this power-limited environment, visualization algorithms merit special consideration, since they are more data intensive in nature than typical HPC applications like simulation codes.

At present, there is a very limited set of work addressing the challenges of visualization and analysis with respect to power constraints on future architectures. The goal of my dissertation is to explore this field uniquely positioned at the intersection of power-constrained HPC and visualization in order to provide understanding of the tradeoffs between power and performance specifically for visualization routines. Understanding the specific execution behaviors of visualization algorithms will enable us to develop better power-aware strategies for optimal performance and power usage.

This dissertation is organized as follows:

- Chapters II and III: We evaluate the tradeoffs between performance and power usage when varying different input parameters for a representative set of visualization algorithms.

- Chapter IV: We develop a power-aware visualization framework that incorporates prediction to dynamically reallocate power within a job and improve performance of a visualization algorithm.

- Chapter V: We evaluate a predictive and adaptive scheduling strategy for dynamically reallocating power within a job.

- Chapter VI: We discuss ideas for future research directions in the area of visualization and power usage.

1.4 Co-Authored Material

Much of the work in this dissertation is from previously published co-authored material. Below is a listing connecting the chapters with the publications and authors that contributed. Further detail on the division of labor is provided at the beginning of each chapter. That said, for each of these publications, I was not only the first author of the paper, but also the primary contributor for implementing systems, conducting studies, and writing manuscripts.

- Chapter II: [35] was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), and myself.

- Chapter III: [36] was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself.

- Chapter IV: [38] was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself.

- Chapter V: [37] was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself.

CHAPTER II

POWER AND PERFORMANCE TRADEOFFS UNDER REDUCED CLOCK FREQUENCIES

Most of the text in this chapter comes from [35], which was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), and myself. The writing of this paper was a collaboration between Hank Childs and myself, and I performed the lead role on all writing, especially including the description of experiments and results. Matthew Larsen and I wrote the benchmark tests. Hank Childs and Matthew Larsen provided the data sets in different sizes and layouts. Hank Childs and I developed the algorithm implementations. I developed the performance monitoring infrastructure, designed and executed the study, and contributed to the majority of the other sections.

This chapter explores the power and performance tradeoffs for one common visualization routines. In this chapter, we focus specifically on isosurfacing, a canonical visualization algorithm, where the output is a three-dimensional surface containing points of a constant value. We ran an extensive study to understand changes in execution behaviors when the CPU clock frequency is reduced. Findings from this study begin to inform how we can tune algorithmic-level knobs to optimize energy and power usage.

2.1 Motivation

Power is a central issue for achieving future breakthroughs in high performance computing (HPC). As today's leading edge supercomputers require between 5 and 18 MegaWatts to power, and as the cost of one MegaWatt-year is approximately one million US dollars, supercomputing centers regularly spend over five million US dollars per year, and sometimes exceed ten million US dollars per year. Worse, power usage is proportional to the size of the machine; scaling up to even larger machines will cause power usage (and associated costs) to grow even larger. Applying today's designs to exascale computing would cost hundreds of millions of US dollars to power. As a result, the HPC community has made power efficiency a central issue, and all parts of the HPC ecosystem are being re-evaluated in the search for power savings.

Supercomputers require a varying amount of power. When running programs that stay within the machine's normal operating limits, the amount of power often matches the usage for when the machine is idle. However, programs that engage more of the hardware — whether it is caches, additional floating point units, etc. — use more power. HPL (High Performance Linpack), a benchmark program that is computationally intensive, has been known to triple power usage, since HPL has been highly optimized and makes intense use of the hardware. However, many visualization programs have not undergone the same level of optimization, and thus only require power near the machine's idle rate. That said, alternate approaches exist that do create opportunities for data-intensive programs — i.e., visualization programs — to achieve energy savings.

As power usage is highly dependent on clock frequency, reductions in frequency can lead to significant power savings. On the face of it, reducing the clock frequency seems like, at best, a break-even strategy — i.e., running at half the speed should take twice as long to execute. However, visualization programs are different than traditional HPC workloads, since many visualization algorithms are data-intensive. So, while HPC workloads engage floating point units (and thus drive up power), visualization workloads make heavier use of cache.

The data-intensive nature of visualization algorithms creates an opportunity: newer architectures have controls for slowing down the clock frequency, but keeping the cache operating at a normal speed. In this case, power is being devoted to the cache at the same rate (which is good because cache is often a bottleneck), but power is devoted to the CPU at a lower rate (which is also good because the CPU is being under-utilized). As the extreme outcome, then, it is conceivable that slowing down the clock frequency (and keeping the caches operating at full speed) could lead to a scenario where the execution time is the same (since the cache performance dominates), but the power usage drops.

With this study, we explore the efficacy of varying clock speed to achieve energy savings. The achieved effects vary based on myriad factors, and we endeavor to understand those factors and their impacts. Our study focuses on a representative visualization algorithm (isosurfacing), and looks at how that algorithm performs under a variety of configurations seen in HPC settings. We find that these configurations have real impacts on power-performance tradeoffs, and that some approaches lend themselves to better energy efficiency than others.

2.2 Related Work

2.2.1 Power. Power is widely viewed as the central challenge for exascale computing, and that challenge is expected to impact exascale software [21], including visualization software [8]. Outside of visualization, many HPC researchers have looked at how to reduce energy usage at different levels ranging from the processor to the full system, including tradeoffs between power and performance. Porterfield et al. [59] looked into the variability in the performance to energy usage at the processor level using OpenMP and MPI. Other research has been dedicated to reduce the total power usage of the system [25, 26, 42, 59]. Ge et al. [28]

developed compute-bound and memory-bound synthetic workload to demonstrate that power-performance characteristics are determined by various characteristics in the application.

The study closest to the one described in this chapter was by Gamell et al. [27]. In this work, they investigated the power-performance relationship for visualization workloads. However, our studies are complementary, as they looked at behaviors across nodes and we aim to illuminate the behavior within a node. A second noteworthy study was by Johnsson et al. [31]. In this study, the authors studied power usage on a GPU when different rendering features were enabled and disabled. Our study is different in nature, as we are studying a visualization algorithm, and also we are studying impacts of programming model, data set, and architectural features, rather than changing the (OpenGL) rendering algorithm itself.

2.2.2 Visualization.

Our study focuses on a traditional isosurfacing algorithm using Marching Cubes [44]. While our study does not preclude using an acceleration structure to quickly identify only the cells that contain the isosurface [18], we did not employ this optimization since we wanted the data loads and stores to follow a more controlled pattern.

Most of our experiments were run using our own implementation of an isosurface algorithm. However, some experiments were run using the isosurfacing algorithm in the Visualization ToolKit (VTK) [64]. Further, we believe that the corresponding results are representative of the class of general frameworks, e.g., OpenDX [6] and AVS [66], and of the end-user tools built on top of such frameworks (i.e., VTK-based ParaView [10] and VisIt [17]).

2.3 Benchmark Tests

One goal for this study is to identify situations where the clock frequency can be reduced, but the execution time does not increase proportionally. In such cases, energy savings are possible. However, we should only expect the execution time to be maintained if the computation is data-bound. This situation occurs when data load and store requests exceed what the memory infrastructure can deliver.

To get a sense of when programs are data-bound and when they are compute-bound, we created four micro-benchmarks. These benchmarks will also inform the spectrum of outcomes we can expect. The benchmarks are:

- *computeBound*: A compute-bound workload performing several multiplication operations on one variable.
- *computeBoundILP*: The above compute-bound benchmark with instruction-level parallelism. This enables pipelining of multiple instructions.
- *memoryBound*: A memory-bound benchmark that accesses an element in an array and then writes it to another array based on an index.
- *memoryBoundCacheThrash*: The above memory-bound benchmark, but the indices that map the output value have been randomized, removing any benefit of locality.

Figure 1 shows the performance results of our micro-benchmarks with varying CPU clock frequencies. Our original hypothesis was that the *computeBound* workload would double in execution time if run at half the speed, the *memoryBoundCacheThrash* application would have the most consistent runtime across all frequencies, and the *computeBoundILP* and *memoryBound* workloads

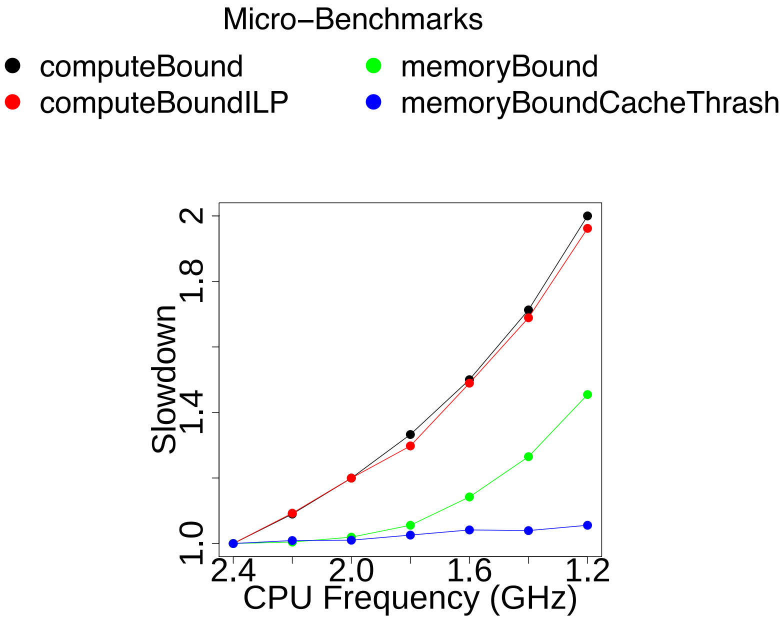


Figure 1. Performance results of our micro-benchmarks with varying frequencies. The *computeBound* workload is directly proportional to the clock speed, while the *memoryBoundCacheThrash* is independent of the change in clock frequency.

would have changes in runtime that fall between the two extremes. From the figure, we find that the *computeBound* workload follows our initial premise. The *memoryBoundCacheThrash* workload stays relatively consistent, but there is a slight increase in runtime when run at the lowest frequency. Even with a synthetic data-bound workload, we are not able to achieve perfect consistency in runtime over varying frequencies. This means that we should not expect visualization workloads to achieve perfect consistency in runtime, since they have significantly more computations than the synthetic workload, and since they use cache in a more coherent way.

2.4 Experimental Setup

The following section details the various parameters in our experiments.

2.4.1 Factors Studied. We considered the following factors:

Benchmark	Time		Ratio
	2.4 GHz	1.2 GHz	
computeBound	24.59s	49.17s	2X
computeBoundILP	1.32s	2.58s	2X
memoryBound	5.25s	7.63s	1.4X
memoryBoundCacheThrash	79.78s	84.22s	1.1X

Table 1. Increase in runtime for the four micro-benchmarks when slowing down the clock frequency by a factor of 2X. Though *memoryBoundCacheThrash* is synthetically designed to be the most data-intensive workload, it still does not hold constant in runtime across frequencies, i.e., achieve a ratio of exactly 1X.

- **Hardware architecture.** Architecture is important, since each architecture has different power requirements at varying clock frequencies, and also different caching characteristics.

- **CPU clock frequency.** As the clock speed slows down, the data-intensive workloads may not slow down proportionally, creating opportunities for power savings.

- **Data set.** Data set dictates how the algorithm must traverse memory. While structured data accesses memory in a regular pattern, unstructured data may have more random arrangements in memory, increasing the data intensity.

- **Parallel programming model.** The programming model affects how multi-core nodes access data and the necessary memory bandwidth for the algorithm. Specifically, coordinated accesses across cores can reduce cache thrashing, while uncoordinated accesses can increase cache thrashing.

- **Concurrency.** Concurrency affects the demands placed on the cache: more cores are more likely to hit the memory infrastructure's bandwidth limit, while fewer cores are less likely.

- **Algorithm implementation.** The algorithm implementation dictates the balance of computations and data loads and stores. Across implementations, the total number of instructions and the ratios of instruction types will change, which in turn could affect power-performance tradeoffs.

2.4.2 Configurations.

Our study consisted of six phases and 277 total tests. It varied six factors:

- Hardware architecture: 2 options
- CPU clock frequency: 7 or 11 options, depending on hardware architecture
- Data set: 8 options
- Parallel programming model (OpenMP vs. MPI): 2 options
- Concurrency: 4 options
- Algorithm implementation: 2 options

2.4.2.1 Hardware Architecture.

We studied two architectures:

- **CPU1**: A Haswell Intel i7 4770K with 4 hyper-threaded cores running at 3.5 GHz, and 32 GB of memory operating at 1600 MHz. Each core has a private L1 and L2 cache running with a bandwidth of 25.6 Gbytes/s.
- **CPU2**: A single node of NERSC's Edison machine. Each node contains two Intel Ivy Bridge processors, and each processor contains 12 cores, running at 2.4 GHz. Each node contains 64 GB of memory operating at 1866 MHz. Each core has a private L1 and L2 cache, with 64 KB and 256 KB, respectively. A 30 MB L3 cache is shared between the 12 cores. The cache bandwidth for L1/L2/L3 is 100/40/23 Gbytes/s.

Both CPUs enable users to set a fixed CPU clock frequency as part of launching the application. CPU1 uses the Linux `cpufreq-utils` tool, while CPU2 uses an `aprun` command line argument to specify the frequency of a submitted job. Both CPUs are also capable of reporting total energy usage and power consumed (see Section 2.4.3).

Finally, it is important to note that Intel Haswell processors (i.e., CPU1) do not tie cache speeds to clock frequency, but Intel Ivy Bridge processors (i.e., CPU2) do force their caches to match clock frequency, and thus their caches slow down when clock frequency is reduced.

2.4.2.2 CPU Clock Frequency. For CPU1, we were able to set the clock frequency at 11 different options, from 1.6 GHz to 3.5 GHz (nominal frequency). For CPU2, the hardware controls only allowed for 7 options, from 1.2 GHz to 2.6 GHz (nominal frequency).

2.4.2.3 Data Set. In this study, we consider only unstructured meshes, although we consider different sizes, and different cache coherency characteristics. Our study used the following eight data sets:

- **Enzo-1M**: a cosmology data set from the Enzo [55] simulation code originally mapped on a rectilinear grid. We decomposed the data set into 1.13 million tetrahedrons.
- **Enzo-10M**: a 10.5 million tetrahedron version of Enzo-1M.
- **Enzo-80M**: an 83.9 million tetrahedron version of Enzo-1M.
- **Nek5000**: a 50 million tetrahedron unstructured mesh from a Nek5000 [24] thermal hydraulics simulation. Nek5000's native mesh is unstructured, but

composed of hexahedrons. For this study, we divided these hexahedrons into tetrahedrons.

- **REnzo-1M, REnzo-10M, REnzo-80M, RNek5000**: Altered versions of the above data sets. We randomize the point indices such that accesses are irregular and locality is not maintained.

We selected an isovalue of 170 for the Enzo data sets, and 0.3 for Nek5000. While "isovalue" could have served as another parameter for our study, we found that varying it did not significantly affect results.

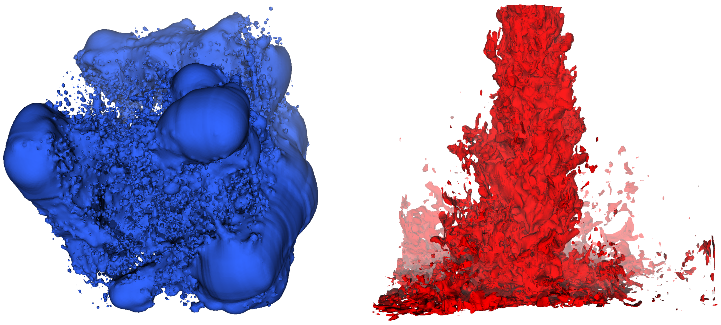


Figure 2. Visualizations of the two data sets used in this study. The Enzo data set is on the left and Nek5000 is on the right.

2.4.2.4 **Parallel Programming Model.**

We implemented our algorithms using both the OpenMP [16] and MPI [65] parallelism approaches within a node.

With the OpenMP approach, all cores operated on a common data set, meaning that cache accesses can be done in a coordinated way. Our OpenMP implementation used default thread scheduling. With this method, the chunk size is determined by the number of iterations in the loop divided by the number of OpenMP threads. That said, we experimented with many chunking strategies and were not able to locate any that significantly outperformed the default.

With the MPI approach, each core operated on an exact local copy of the data in its own space. As a result, the cores were operating independently, creating uncoordinated cache accesses.

Whatever the parallel programming model, the tests operated on the same data size. Given C cores and N cells total, the OpenMP approach would have all cores operate on the N cells together, and perform the operation C times, while the MPI approach would have each core operate on $N * C$ cells. Further, in order to obtain reliable measurements, we had each algorithm execute ten times, and the reported measurements are for all ten executions.

2.4.2.5 Concurrency. For CPU1, we ran tests using 1, 2, 3, and 4 cores. For CPU2, no tests of this nature were run.

2.4.2.6 Algorithm Implementation. We implemented two different versions of isosurfacing for our study:

- **BaselineImplementation**: We implemented our own isosurfacing algorithm. This algorithm could only perform tetrahedral Marching Cubes (Marching Tets), and so it was efficient for that purpose, especially in terms of minimal numbers of instructions. We implemented versions of our code to work with both MPI and OpenMP.

- **GeneralImplementation**: Isosurface implemented using a general-purpose visualization software library (VTK), specifically the vtkContourFilter. Generalized frameworks like VTK sacrifice performance to ensure that their code works on a wide variety of configurations and data types. As a result, the performance characteristics of such a framework are different (specifically having an increased number of instructions), and thus the opportunities for power-performance tradeoffs may also be different. The vtkContourFilter does not work with OpenMP, so our only supported programming model for this implementation was with MPI (via our own custom MPI program that incorporated this module).

2.4.3 Performance Measurements.

We enabled PAPI [52] performance counters to gather measurements for each phase of the algorithm. Specifically, we capture `PAPI_TOT_INS`, `PAPI_TOT_CYC`, `PAPI_L3_TCM`, `PAPI_L3_TCA`, and `PAPI_STL_ICY`. (Note that `PAPI_TOT_CYC` counts all instructions executed, which can vary from run to run due to CPU branch speculation. Unfortunately, we were not able to count instructions retired, which would be consistent across runs.)

We then derive additional metrics from the PAPI counters:

- instructions executed per cycle (IPC) = `PAPI_TOT_INS` / `PAPI_TOT_CYC`
- L3 cache miss rate = `PAPI_L3_TCM` / `PAPI_TOT_CYC`

On CPU1, we used Intel's Running Average Power Limit (RAPL) [19] to obtain access to energy measurements. This instrumentation provides a per socket measurement, aggregating across cores. On CPU2, we took power and energy measurements using the Cray XC30 power management system [48]. This instrumentation provides a per node measurement, again aggregating across cores.

2.4.4 Methodology.

Our study consisted of six phases. The first phase studied a base case, and the subsequent phases varied additional dimensions from our test factors, and analyzed the impacts of those factors.

2.4.4.1 Phase 1: Base Case.

Our base case varied the CPU clock frequency. It held the remaining factors constant: CPU1, Enzo-10M, four cores (maximum concurrency available on CPU1), the BaselineImplementation, and the

OpenMP parallel programming model. The motivation for this phase was to build a baseline understanding of performance.

Configuration: (CPU1, 4 cores, Enzo-10M, BaselineImplementation, OpenMP) × 11 CPU clock frequencies

2.4.4.2 Phase 2: Data Set. In this phase, we continued varying clock frequency and added variation in data set. It consisted of 88 tests, of which 11 were studied in Phase 1 (the 11 tests for Enzo-10M).

Configuration: (CPU1, 4 cores, BaselineImplementation, OpenMP) × 11 CPU clock frequencies × 8 data sets

2.4.4.3 Phase 3: Parallel Programming Models. In this phase, we continued varying clock frequency and data set and added variation in parallel programming model. It consisted of 176 tests, of which 88 were studied in Phase 2 (the OpenMP tests).

Configuration: (CPU1, 4 cores, BaselineImplementation) × 11 CPU clock frequencies × 8 data sets × 2 parallel programming models

2.4.4.4 Phase 4: Concurrency. In this phase, we went back to the Phase 1 configuration, and added variation in concurrency and programming model. It consisted of 88 tests, of which 11 were studied in Phase 1 (the OpenMP configurations using all 4 cores) and 11 were studied in Phase 3 (the MPI configurations using all 4 cores).

Configuration: (CPU1, Enzo-10M, BaselineImplementation) × 11 CPU clock frequencies × 4 concurrency levels × 2 parallel programming models

2.4.4.5 Phase 5: Algorithm Implementation. In this phase, we studied variation in algorithm implementation. Since the GeneralImplementation was only available with the MPI parallel programming model, we could not go back to Phase 1. Instead, we compared 11 new tests with 11 tests first performed in Phase 3.

Configuration: (CPU1, 4 cores, Enzo-10M, MPI) × 11 CPU clock frequencies × 2 algorithm implementations

2.4.4.6 Phase 6: Hardware Architecture.

With this test, we went to a new hardware architecture, CPU2. We kept many factors constant — BaselineImplementation, Enzo-10M, 24 cores — and varied CPU clock frequency and parallel programming model. All 14 tests for this phase were new.

Configuration: (CPU2, 24 cores, Enzo-10M, BaselineImplementation) × 7 CPU clock frequencies × 2 parallel programming models

2.5 Results

In this section, we describe results from the six phases detailed in Section 2.4.4. Before doing so, we consider an abstract case, as the analysis of this abstract case is common to the analysis of each phase. We also define terms that we use throughout this section.

Assume a visualization algorithm, when running at the default clock frequency of F_D, takes time T_D seconds to run, consumes a total energy of E_D Joules, and requires an average power of P_D Watts (with $P_D = E_D/T_D$). Further assume that same visualization algorithm, when reducing the clock frequency to F_R, takes T_R seconds, consumes a total of E_R Joules, and requires an average of P_R Watts (once again with $P_R = E_R/T_R$). We then define the following terms:

- $F_{rat} = F_D/F_R$. This is the ratio of the clock frequencies. If the clock frequency was slowed down by a factor of two, then $F_{rat} = 2$.

- $T_{rat} = T_R/T_D$. This is the ratio of elapsed time. If the algorithm runs twice as slow, then $T_{rat} = 2$.

- $E_{rat} = E_D/E_R$. This is the ratio of energy consumed. If the energy consumed is reduced by a factor of two, then $E_{rat} = 2$.
- $P_{rat} = P_D/P_R$. This is the ratio of power usage. If the power usage is reduced by a factor of two, then $P_{rat} = 2$.

Note that three of the terms have the value for the default clock frequency in the numerator and the value for the reduced clock frequency in the denominator, but that T_{rat} flips them. This flip simplifies comparison across terms, since it makes all ratios be greater than 1.

We then find these three pairs of terms noteworthy:

- F_{rat} and T_{rat}: When T_{rat} is less than F_{rat}, the data-intensive nature of the visualization algorithm enabled the program to slow down at a rate less than the reduction in clock frequency.
- T_{rat} and E_{rat}: This pair represents a proposition for visualization consumers (i.e., visualization scientists or simulation scientists who use visualization software): "if you are willing to run the visualization (T_{rat}) times slower, then you can use (E_{rat}) times less energy."
- T_{rat} and P_{rat}: This pair represents a related proposition for visualization consumers: "if you are willing to run the visualization (T_{rat}) times slower, then you can use (P_{rat}) times less power when doing so." This power proposition would be useful for those that want to run a computing cluster at a fixed power rate.

2.5.1 Phase 1: Base Case.

Phase 1 fixed all factors except clock frequency, to provide a baseline for future phases. The factors held constant were:

BaselineImplementation, OpenMP, 4 cores on CPU1, and the Enzo-10M data set. Table 2 contains the results.

In terms of our three ratios:

- F_{rat} and T_{rat}: At the slowest clock speed (1.6 GHz), F_{rat} was 2.2X, but T_{rat} was 1.84X, meaning that the program was not slowing down proportionally. A purely compute-intensive program that took 1.29s at 3.5 GHz would have taken 2.82s at 1.6 GHz, while our isosurfacing program took 2.40s (i.e., 17% faster).

- T_{rat} and E_{rat}: Energy savings of up to 1.44X can be gained by accepting slowdowns of up to 1.84X. Clock frequencies in between the extremes offer propositions with less energy savings, but also less impact on runtime.

- T_{rat} and P_{rat}: Power savings of up to 2.7X can be gained by accepting slowdowns of up to 1.84X. The power savings are greater than the energy savings since the energy accounts for reduced runtime, while the power only speaks to instantaneous usage. Regardless, such power savings could be useful when running complex systems with a fixed power budget.

2.5.2 Phase 2: Data Set. Phase 2 extended Phase 1 by varying over data set. Table 3 shows specific results for the REnzo-10M data set (which compares with the Enzo-10M data set in Table 2 of Section 2.5.1), and Figure 3 shows aggregate results over all data sets.

In terms of our three ratios:

- F_{rat} and T_{rat}: The right sub-figure of Figure 3 shows that the slowdown factor varies over data set. In the worst case, for the Enzo-1M data set, the slowdown factor is at 2.2X — i.e., exactly 3.5 GHz over 1.6 GHz — meaning that it is performing like a computationally-intensive workload. This makes sense, however, since Enzo-1M is our smallest data set, and it has a regular data access pattern.

- T_{rat} and E_{rat}: This tradeoff varies based on data set. The data sets with randomized access patterns (REnzo, RNek) have better propositions, as do large data sets. Also, when comparing Table 3 and Table 2, we can see that the tradeoffs got more favorable with REnzo-10M, with energy savings of 1.7X against slowdowns of 1.4X (where it was 1.44X and 1.84X for Enzo-10M).
- T_{rat} and P_{rat}: Table 3 shows us that the power tradeoff for REnzo-10M is slightly worse than Enzo-10M. We attribute the increase in instantaneous power to increased data intensity (see Table 4).

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	1.29s	1X	74.3J	1X	57.4W	1X
3.3GHz	1.1X	1.32s	1X	69.4J	1.1X	52.6W	1.1X
3.1GHz	1.1X	1.38s	1.1X	66.7J	1.1X	48.2W	1.2X
2.9GHz	1.2X	1.42s	1.1X	63.4J	1.2X	44.8W	1.3X
2.7GHz	1.3X	1.50s	1.2X	61.5J	1.2X	40.9W	1.4X
2.5GHz	1.4X	1.62s	1.3X	60.9J	1.2X	37.5W	1.6X
2.3GHz	1.5X	1.78s	1.4X	53.7J	1.4X	30.1W	1.9X
2.1GHz	1.7X	1.93s	1.5X	53.8J	1.4X	27.9W	2.1X
2.0GHz	1.8X	1.95s	1.5X	52.1J	1.4X	26.8W	2.2X
1.8GHz	1.9X	2.13s	1.7X	51.1J	1.4X	24.1W	2.4X
1.6GHz	2.2X	2.40s	1.9X	51.4J	1.4X	21.4W	2.7X

Table 2. Experiment results for Phase 1, which uses OpenMP and the BaselineImplementation.

The performance measurements listed in Table 4 help explain the differences between the data sets. Specifically, the L3 miss rate (unsurprisingly) goes up when data sets get larger and their accesses become randomized. This in turn pushes

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	2.95s	1X	142.7J	1X	48.4W	1X
3.3GHz	1.1X	3.05s	1X	134.8J	1.1X	44.2W	1.1X
3.1GHz	1.1X	3.01s	1X	124.3J	1.1X	41.3W	1.2X
2.9GHz	1.2X	3.33s	1.1X	122.3J	1.2X	36.8W	1.3X
2.7GHz	1.3X	3.23s	1.1X	109.3J	1.3X	33.8W	1.4X
2.5GHz	1.4X	3.22s	1.1X	99.6J	1.4X	30.9W	1.6X
2.3GHz	1.5X	3.48s	1.3X	93.4J	1.5X	26.8W	1.8X
2.1GHz	1.7X	3.49s	1.3X	88.0J	1.6X	25.2W	1.9X
2.0GHz	1.8X	3.79s	1.3X	88.3J	1.6X	23.3W	2.1X
1.8GHz	1.9X	3.79s	1.3X	82.2J	1.7X	21.7W	2.2X
1.6GHz	2.2X	4.19s	1.4X	82.1J	1.7X	19.6W	2.5X

Table 3. Experiment results for the REnzo-10M data set in Phase 2, which uses OpenMP and the BaselineImplementation.

down the number of instructions per cycle (a surrogate for capturing how many stalls are occurring in the pipeline, which is difficult to measure).

Data Set	Time	Cycles	IPC	L3 Miss Rate
Enzo-1M	0.39s	614M	1.42	597
Enzo-10M	2.40s	3.0B	1.89	1027
Enzo-80M	13.2s	18B	2.24	1422
Nek5000	14.3s	20B	1.54	949
REnzo-1M	0.44s	700M	1.17	5420
REnzo-10M	4.2s	6.0B	0.94	10913
REnzo-80M	33.9s	51B	0.78	12543
RNek5000	27.2s	38B	0.81	11593

Table 4. Performance measurements for the 1.6 GHz experiments from Phase 2. IPC is short for Instructions Per Cycle, and the L3 Miss Rate is the number of L3 cache misses per one million cycles.

2.5.3 Phase 3: Parallel Programming Models.

Phase 3 extended Phase 2 by varying over parallel programming model. Figure 4 shows the overall results for all eight data sets using MPI; it can be compared with Figure 3 of Section 2.5.2, which did the same analysis with OpenMP. Tables 5 and 6 show the results using MPI on the Enzo-10M and REnzo-10M data sets, respectively.

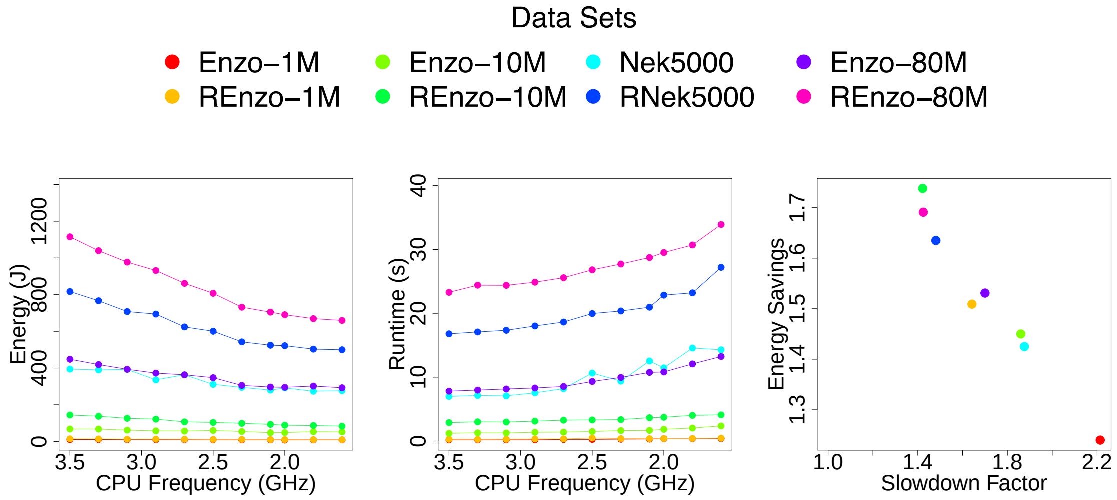


Figure 3. Results from Phase 2, which uses four cores of CPU1 with OpenMP and the BaselineImplementation and varies over data set and clock frequency. The plots are of energy (left) and runtime (middle), as a function of CPU clock frequency. The right figure is a scatter plot of the 1.6GHz slowdown factor versus energy savings for the eight data sets.

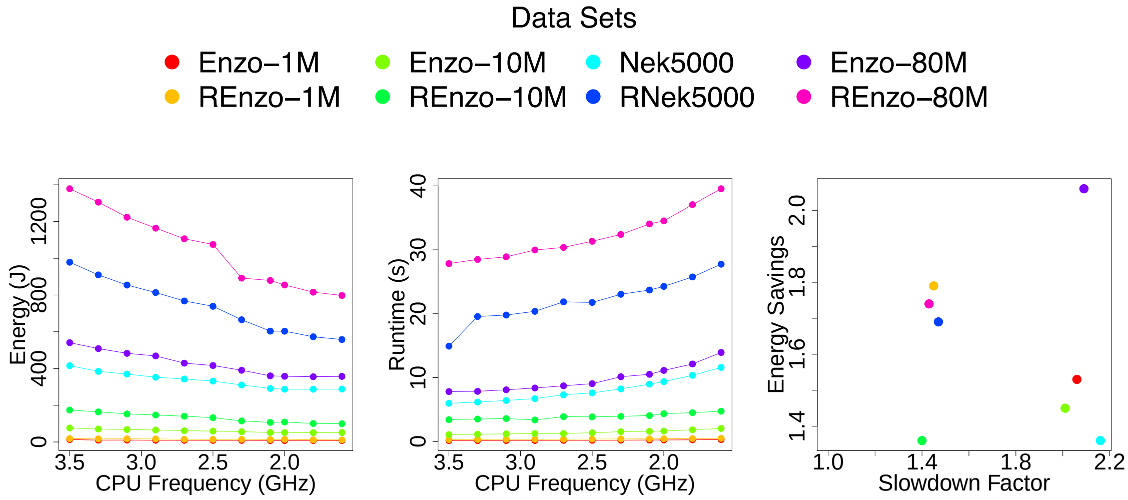


Figure 4. Results from Phase 3, which uses four cores of CPU1 with MPI and the BaselineImplementation and varies over data set and clock frequency. The plots are of energy (left) and runtime (middle), as a function of CPU clock frequency. The right figure is a scatter plot of the 1.6GHz slowdown factor versus energy savings for the eight data sets.

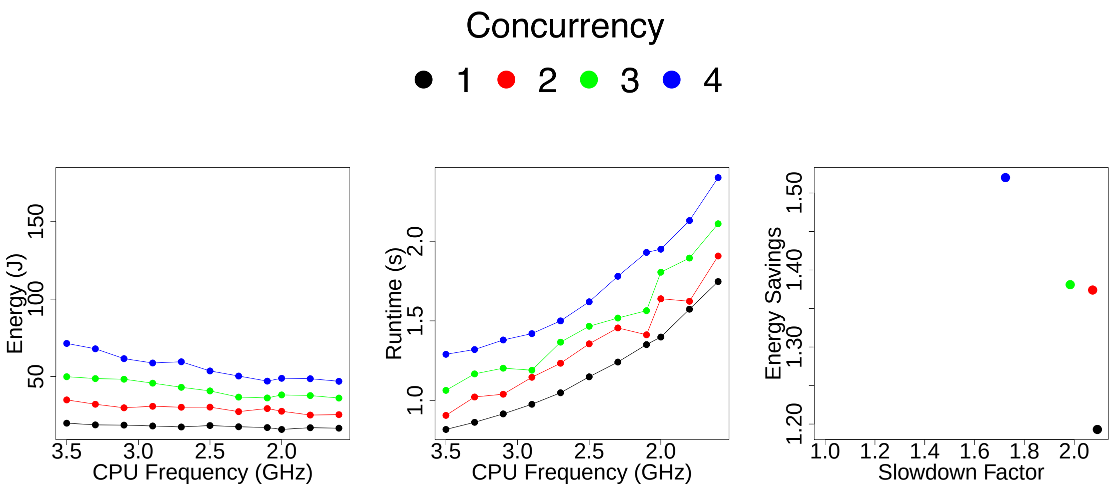


Figure 5. Results from Phase 4's tests with OpenMP, studying Enzo-10M using CPU1 and the BaselineImplementation and varying over clock frequency and concurrency. The plots are of energy (left) and runtime (middle), as a function of CPU clock frequency. The right figure is a scatter plot of the 1.6GHz slowdown factor versus energy savings for the four concurrencies.

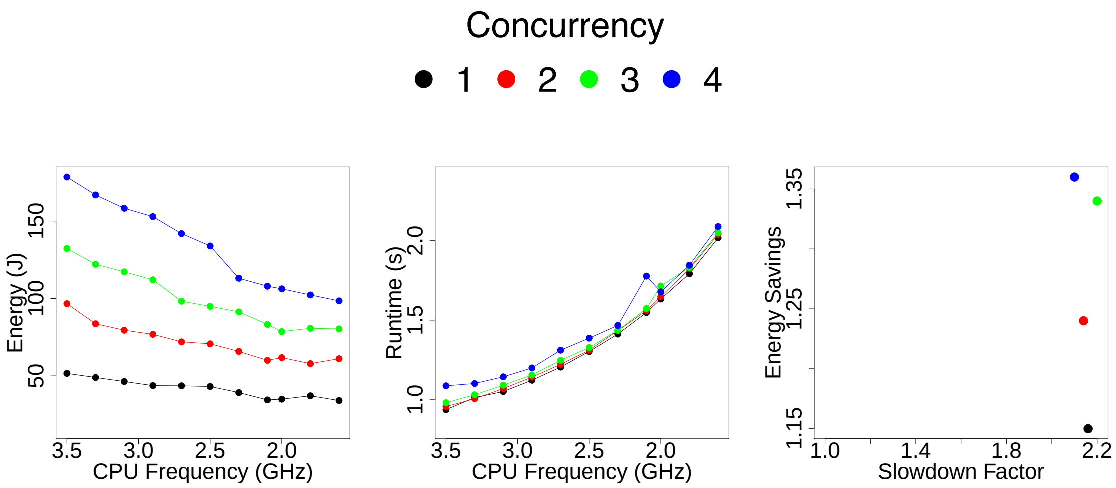


Figure 6. Results from Phase 4's tests with MPI, studying Enzo-10M using CPU1 and the BaselineImplementation and varying over clock frequency and concurrency. The plots are of energy (left) and runtime (middle), as a function of CPU clock frequency. The right figure is a scatter plot of the 1.6GHz slowdown factor versus energy savings for the four concurrencies.

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	1.08s	1X	74.5J	1X	69.2W	1X
3.3GHz	1.1X	1.12s	1X	70.4J	1.1X	62.7W	1.1X
3.1GHz	1.1X	1.18s	1.1X	67.3J	1.1X	57.0W	1.2X
2.9GHz	1.2X	1.20s	1.1X	66.2J	1.1X	55.0W	1.3X
2.7GHz	1.3X	1.35s	1.3X	63.5J	1.2X	47.1W	1.5X
2.5GHz	1.4X	1.36s	1.3X	59.8J	1.2X	43.9W	1.6X
2.3GHz	1.5X	1.46s	1.4X	55.3J	1.3X	37.8W	1.8X
2.1GHz	1.7X	1.59s	1.4X	51.7J	1.4X	32.6W	2.1X
2.0GHz	1.8X	1.80s	1.7X	55.4J	1.3X	30.8W	2.2X
1.8GHz	1.9X	1.92s	1.7X	52.6J	1.4X	27.4W	2.5X
1.6GHz	2.2X	2.08s	2X	51.8J	1.4X	24.9W	2.8X

Table 5. Experiment results from Phase 3 for the Enzo-10M data set, which uses MPI and the BaselineImplementation.

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	3.46s	1X	179.5J	1X	51.9W	1X
3.3GHz	1.1X	3.48s	1X	166.8J	1.1X	47.9W	1.1X
3.1GHz	1.1X	3.59s	1X	158.9J	1.1X	44.2W	1.2X
2.9GHz	1.2X	3.62s	1X	147.7J	1.2X	40.8W	1.3X
2.7GHz	1.3X	3.78s	1.1X	143.0J	1.3X	37.9W	1.4X
2.5GHz	1.4X	3.88s	1.1X	135.4J	1.3X	34.9W	1.5X
2.3GHz	1.5X	4.00s	1.1X	116.2J	1.5X	29.1W	1.8X
2.1GHz	1.7X	4.18s	1.3X	108.0J	1.7X	25.8W	2X
2.0GHz	1.8X	4.29s	1.3X	109.8J	1.6X	25.6W	2X
1.8GHz	1.9X	4.52s	1.3X	105.0J	1.7X	23.2W	2.2X
1.6GHz	2.2X	4.62s	1.4X	95.5J	1.9X	20.7W	2.5X

Table 6. Experiment results from Phase 3 for the REnzo-10M data set, which uses MPI and the BaselineImplementation.

Table 2 of Section 2.5.1 and Table 3 of Section 2.5.2 are also useful for comparison, as they showed the results for these same data sets using OpenMP.

In terms of our three ratios:

- F_{rat} and T_{rat}: The right sub-figure of Figure 4 shows two clusters: one grouping (made up of the randomized data sets) slows down only by a factor of ˜1.4, while the other grouping (made up of the non-randomized data sets) slows down in near proportion with the clock frequency reduction. This contrasts with the OpenMP tests seen in Figure 3, which showed more spread over these two extremes. We conclude that the randomized data sets create significantly more memory activity for MPI than for OpenMP, which is supported by our performance measurements. Taking REnzo-80M as an example, the MPI test had over 48,000 L3 cache misses per million cycles, while the OpenMP test had less than 12,500.

- T_{rat} and E_{rat}: Renzo-10M with MPI gave the largest energy savings of any test we ran, going from 179.5J to 95.5J. That said, its starting point was higher than OpenMP, which went from 142.7J to 82.1J. Overall, energy savings were harder to predict with MPI, but were generally better than the savings with OpenMP (again because it was using more energy to start with).
- T_{rat} and P_{rat}: the MPI tests used more power, but saw greater reduction when dropping the clock frequency. For the Enzo-10M data set, the MPI test dropped from 72.2W (3.5GHz) to 25.3W (1.6GHz), while OpenMP dropped from 57.4W to 21.4W. MPI's increased power usage likely derives from activity with the memory system, and increased L3 cache misses.

Summarizing, our performance measurements show that the MPI approach uses the memory infrastructure less efficiently, leading to increased energy and power usage, but also creating improved propositions for reducing energy and power when reducing clock frequencies.

2.5.4 Phase 4: Concurrency.

Phase 4 did not build on Phase 3, but rather went back to Phase 1 and extended it by considering multiple concurrency levels and programming models. Figure 5 shows plots of our results with OpenMP and Figure 6 shows the results with MPI. Table 7 contains data that complements the figures.

Higher levels of concurrency issue more memory requests, leading to saturation of the memory infrastructure. As a result, runtimes steadily increase. Because some processes may have to wait for the memory infrastructure to satisfy requests, we observed energy savings by slowing down the clock frequency, making waiting processes consume less power and having their memory requests satisfied more quickly (relative to the number of cycles, since cycles lasted longer). Table 7 shows this trend, in particular that the four core configurations overwhelm their memory (as seen in the increase in L3 cache misses and in the reduction in instructions per cycle), while the one core configurations fare better.

In terms of our three ratios:

- F_{rat} and T_{rat}: The right sub-figures of Figures 5 and 6 show that the slowdown factor varies over concurrency. Lower concurrencies (which have faster runtimes) have higher slowdowns, because their memory accesses are being supported by the caching system. Higher concurrencies (which have longer runtimes) have lower slowdowns, because the cache was not keeping up as well at high clock frequencies (since more processors were issuing competing requests).

- T_{rat} and E_{rat}: The tradeoff between slowdown and energy varies quite a bit over concurrency. With OpenMP, a single core suffers over a 2X slowdown to receive a 1.2X energy savings. But, with four cores, the slowdown improves to 1.86X and the energy savings improve to 1.45X. With MPI, the trends are similar, but less pronounced.

- T_{rat} and P_{rat}: As seen in Table 7, the power savings get better as more cores are used, but not dramatically so. With one core, both OpenMP and MPI provide 2.5X power improvements by dropping the clock frequency. Going up to four cores raises this power improvement to 2.85 (MPI) and 2.68 (OpenMP).

Configuration	Time	Energy	Power	IPC	L3 Miss Rate
MPI/1/3.5 GHz	0.90s	24.4J	26.9W	2.37	5652
MPI/1/1.6 GHz	2.0s	21.1J	10.8W	2.41	3788
OpenMP/1/3.5 GHz	0.83s	19.9J	23.9W	2.06	1697
OpenMP/1/1.6 GHz	1.74s	16.7J	9.6W	2.11	931
MPI/4/3.5 GHz	0.96s	69.5J	72.2W	2.07	10476
MPI/4/1.6 GHz	2.02s	51.2J	25.3W	2.37	3456
OpenMP/4/3.5 GHz	1.29s	74.3J	57.4W	1.51	3351
OpenMP/4/1.6 GHz	2.40s	51.1J	21.4W	1.89	1027

Table 7. Experiment results from Phase 4. IPC is short for Instructions Per Cycle, and L3 is the number of L3 cache misses per one million cycles.

2.5.5 Phase 5: Algorithm Implementation.

Phase 5 once again went back to Phase 1 as a starting point, this time extending the experiments to consider multiple algorithm implementations. The factors held constant were: OpenMP, 4 cores on CPU1, and the Enzo-10M data set. Table 8 contains the results.

With GeneralImplementation, runtime and clock frequency are highly correlated, i.e., reducing the clock frequency by 2.2 causes the workload to take 2.1X longer to run. This relationship between frequency and runtime is characteristic of a compute-intensive workload, depicted by our *computeBound* micro-benchmark. In contrast, the BaselineImplementation exhibited behavior closer to data-intensive in our previous phases.

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
3.5GHz	1X	16.06s	1X	1056J	1X	65.8W	1X
3.3GHz	1.1X	16.57s	1X	992J	1.1X	59.9W	1.1X
3.1GHz	1.1X	17.64s	1.1X	950J	1.1X	53.9W	1.2X
2.9GHz	1.2X	19.00s	1.3X	928J	1.1X	48.8W	1.3X
2.7GHz	1.3X	20.85s	1.3X	914J	1.2X	43.9W	1.5X
2.5GHz	1.4X	21.82s	1.4X	876J	1.2X	40.1W	1.6X
2.3GHz	1.5X	24.01s	1.4X	784J	1.3X	32.7W	2X
2.1GHz	1.7X	26.09s	1.7X	763J	1.4X	29.3W	2.2X
2.0GHz	1.8X	27.43s	1.7X	768J	1.4X	28.0W	2.4X
1.8GHz	1.9X	30.67s	1.9X	764J	1.4X	24.9W	2.6X
1.6GHz	2.2X	34.17s	2.1X	756J	1.4X	22.1W	3X

Table 8. Experiment results for GeneralImplementation of Phase 5. These results compare with BaselineImplementation, whose corresponding results are in Table 2.

The explanation for the difference between the two implementations is in the number of instructions. While both issue the same number of loads and stores, the GeneralImplementation issues 102 billion instructions, while the BaselineImplementation issues only 7 billion. These additional instructions change the nature of the computation (from somewhat data-intensive to almost entirely compute intensive), as well as making the overall runtimes and energy consumption much higher. Of course, these instructions add value for general toolkits, in terms of supporting more data models and algorithms. The takeaway from this study is that the approach from general toolkits appears to tilt the instruction mix (at least for isosurfacing).

Interestingly, the E_{rat} and P_{rat} ratios are still favorable, at 1.4X and 3X, respectively. This is because the relationship between clock frequency and energy consumed is not strictly linear. As a result, even compute-intensive workloads can benefit from clock frequency reductions, although their T_{rat}'s will still match the clock frequency reduction.

2.5.6 Phase 6: Architecture.

Phase 6 did not build on any previous phases. Instead, it explored CPU2, whose results do not translate to any of the previous CPU1 experiments. The factors held constant were: MPI, 24 cores on CPU2, Enzo-10M, and the BaselineImplementation. Table 9 contains the results.

F	F_{rat}	T	T_{rat}	E	E_{rat}	P	P_{rat}
2.4GHz	1X	2.265s	1X	549J	1X	242.4W	1X
2.2GHz	1.1X	2.479s	1.1X	558J	1X	225W	1.1X
2.0GHz	1.2X	2.695s	1.2X	571J	1X	211.9W	1.1X
1.8GHz	1.3X	3.024s	1.3X	573J	1X	189.5W	1.3X
1.6GHz	1.5X	3.385s	1.5X	631J	0.9X	186.4W	1.3X
1.4GHz	1.7X	3.836s	1.7X	668J	0.8X	174.1W	1.4X
1.2GHz	2X	4.466s	2X	697J	0.8X	156W	1.6X

Table 9. Experiment results from Phase 6, which uses CPU2 with MPI and the BaselineImplementation.

CPU2 is significantly different than CPU1 in that it contains Ivy Bridge processors, while CPU1 contains Haswell processors. On Haswells, the core (compute units, private L1 and L2 caches) and uncore (shared L3 cache) are on separate clock domains, so slowing down the frequency only applies to the speed of the executing instructions and accessing L1 and L2 caches. On Ivy Bridge, core and uncore share the same clock frequency, and so data-intensive workloads cannot benefit with respect to T_{rat}.

Table 9 shows that, while P_{rat} is better at lower clock frequencies, E_{rat} is worse. Restated, while the power dropped, its drop was not steep enough to offset the increases in runtime, and so overall energy usage goes up. This does not match the results in Phase 5, where a compute-bound workload created a similar "T_{rat} equals F_{rat}" situation. As explanation, we again note the non-linear relationship between power and clock frequency (which varies over architecture).

2.6 Summary

We conducted a study exploring the tradeoffs between power and performance when reducing clock frequencies. As a result of this initial study, we identified that favorable power and performance tradeoffs are possible for a common visualization algorithm when the CPU clock frequency is reduced. More specifically, the isosurfacing algorithm exposes variation, and can be sufficiently data intensive to avoid a performance slowdown with energy savings. We summarize the results of our findings by phase:

- Phase 1 confirmed our basic hypotheses about reducing clock frequencies: (i) Isosurfacing is sufficiently data-intensive to slow the impact from reduced clock frequencies. (ii) Clock frequency reductions can create options for visualization consumers to choose between finishing an algorithm quickly using more energy, or slowly using less energy. (iii) Clock frequency reductions decrease power usage, creating options for visualization consumers wanting to balance system-wide power usage.

- Phase 2 showed that the tradeoffs between energy and runtime get increasingly favorable as data complexity goes up (either due to size of increased irregularity in data access).

- Phase 3 showed that MPI's less coordinated memory accesses affect energy and power tradeoffs compared to OpenMP.

- Phase 4 showed that the tradeoffs between execution time and energy are most favorable when the memory infrastructure is being stressed, and that this scenario exists at higher concurrencies (or, alternatively, is less likely to exist when some of a node's cores are not being used).

- Phase 5 showed that general-purpose implementations of visualization algorithms shift the instruction mix such that the tradeoffs between execution time and energy are less favorable.

- Phase 6 showed the importance of having an architecture where the memory infrastructure can be controlled separately from the CPU.

In terms of future work, we want to explore the variation in a wider set of visualization workloads. While we feel isosurfacing is representative of many visualization algorithms — i.e., those characterized by iterating on cells one-by-one and producing a new output — other algorithms have different properties. In particular, particle advection problems perform data-dependent memory accesses, which may produce even more favorable propositions for energy and power savings. Further, algorithms like volume rendering require both significant computation and irregular memory accesses (especially for unstructured grids), making it unclear how it would be affected by changes in clock frequency. We investigate the power and performance tradeoffs of other visualization workloads in Chapter III.

CHAPTER III

POWER AND PERFORMANCE TRADEOFFS UNDER POWER LIMITS

Most of the text in this chapter comes from [36], which was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself. This paper has been accepted for publication at the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS).

The writing of this paper was a collaboration between Hank Childs, Matthew Larsen, Barry Rountree, and myself. I was the primary contributor to the writing of the overall paper. Additionally, I developed the performance monitoring infrastructure, and designed and executed the study. The overview of power usage in high performance computing was a collaboration between Barry Rountree and myself. Hank Childs and Matthew Larsen provided guidance on organization and also provided edits to the final paper.

This chapter explores the power and performance tradeoffs for a representative set of visualization algorithms. An important premise of this work is that power will be a limited resource on future supercomputers, necessitating an understanding of how applications behave under a power limit. This is a follow-up study to the work presented in Chapter II, which demonstrated opportunities for power and energy savings in a visualization workload. With this more thorough study, we investigate the power and performance tradeoffs for a larger set of visualization algorithms. Whereas Chapter II explored the variation under reduced clock frequencies, this chapter explores the variation under reduced power limits (ultimately, reducing the CPU frequency). The result of this study provides the core of the research presented in this dissertation. We provide a set of recipes that can better inform scientists and tool developers on the effects of visualization-specific input parameters on power usage and performance.

3.1 Motivation

Power is one of the major challenges in reaching the next generation of supercomputers. Scaling current technologies to exascale may result in untenable power costs. Thus, the entire HPC ecosystem, including hardware and software, is being re-designed with power efficiency in mind.

The premise of this research is that simulations and visualization routines (and other components of the HPC ecosystem) will operate in a power-limited environment. The Tokyo Institute of Technology in Japan is one example of a facility that has deployed power-limited production systems [45]. Two of their systems — TSUBAME2 and TSUBAME3 — must share the facility-level power budget (i.e., inter-system power capping). Additionally, due to extreme heat during the summer months, the resource manager may dynamically turn off nodes to stay under a specified power cap.

At exascale, it is expected that visualization routines will need to be run simultaneously with simulations (i.e., *in situ* processing), due to decreasing I/O performance relative to floating point operations. Further, power-limited environments will greatly impact the overall time-to-solution. Efforts to optimize performance under a power bound has typically focused on traditional HPC workloads rather than visualization, which can be a significant portion of the overall execution time. Furter, visualization applications are more data intensive than traditional HPC workloads.

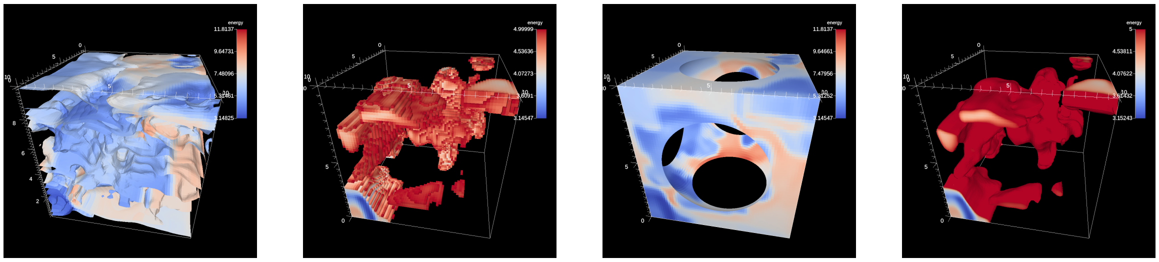
For any simulation, the amount of time dedicated to in situ visualization can vary. It is dependent on a myriad of factors including the type of analysis to be completed and the number of operations in the visualization pipeline. From experience, visualization may account for 10%–20% of the overall execution time spent running the simulation and the visualization.

The main contribution of this work is providing the foundation for future research in this area, which has very few efforts exploring the performance behaviors of visualization algorithms in a power-limited environment. We believe a study focusing on visualization applications is needed for three main reasons. First, visualization is a key phase in the scientific discovery process, transforming abstract data into a comprehensible image useful for communication and exploration. Second, the time to do visualization is often a significant portion of the overall execution time. Third, visualization algorithms are more data intensive than HPC applications.

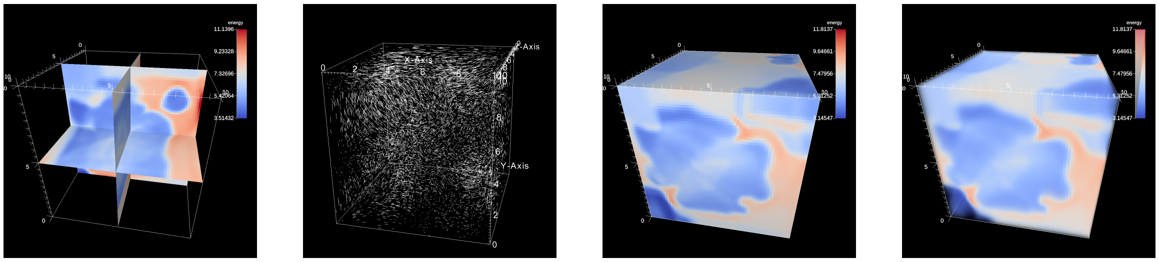
We selected eight common visualization algorithms, which we believe are representative of the execution behaviors of the hundreds of existing visualization algorithms. We also selected four data set sizes and varied the processor-level power cap to understand how the changes affect power and performance properties.

The results of this study identify two classes of algorithms. The first class contains compute-bound algorithms (**power sensitive**). The performance of these algorithms is sensitive to the processor-level power cap, so limiting its available power significantly degrades the performance. The second class contains memory-bound algorithms, which provide a unique opportunity for power savings without sacrificing execution time (**power opportunity**). Our findings may be integrated into a runtime system that assigns power between a simulation and visualization application running concurrently under a power budget, such that overall performance is maximized.

The rest of this chapter is organized as follows. Section 3.2 discusses previous work. Section 3.2 provides an overview of power in HPC and the algorithms explored. The details of the experimental setup and methodology are presented in Section 3.4. We define the metrics and variables used in Section 3.5. Results are discussed in Section 3.6. We summarize our findings in Section 3.7 and identify ideas for future work in Section 3.8.



(a) Contour (b) Threshold (c) Spherical Clip (d) Isovolume



(e) Slice (f) Particle Advection (g) Ray Tracing (h) Volume Rendering

Figure 7. Renderings of the eight visualization algorithms explored in this study. We believe this set of algorithms is representative of the execution behaviors of the hundreds of existing visualization algorithms. The images show the energy field at the 200th time step of the CloverLeaf hydrodynamics proxy application.

3.2 Related Work

Prior works at the intersection of power and performance can be seen in Section 2.2. This chapter builds on the work described in Chapter II [35], where we studied a single visualization algorithm (isosurfacing) and considered explicit setting of the CPU frequency (which is less favorable for managing power usage on exascale systems than more recent power capping technologies such as Intel's Running Average Power Limit (RAPL) [19], AMD's TDP PowerCap [20], and IBM's EnergyScale [30]). In our current study, we consider eight algorithms — chosen to be representative of most visualization algorithms — and use the more current technique of power capping. Therefore, while the initial study [35] showed that a visualization algorithm has unique power and performance tradeoffs, the current study is considerably more comprehensive and also more relevant to exascale computing (i.e., power capping versus setting CPU frequencies). Further, this study contains a series of findings that allow us to extrapolate behavior to other visualization algorithms.

3.3 Overview of Visualization Algorithms

We explored eight algorithms for this study. We believe this set of algorithms is representative of the behaviors and characteristics commonly found across all visualization algorithms. We provide a brief description of each of the eight algorithms in the following subsections (see Figure 7 for a rendered image of each algorithm).

3.3.1 Contour. For a three-dimensional scalar volume, the output of a contour is a surface representing points of a constant value (i.e., isovalue). For this study, the data set consisted of hexahedrons and the algorithm used was Marching Cubes [44]. The contour algorithm iterates over each cell in the data set, identifying cells that contain the constant value. The algorithm uses pre-computed lookup tables in combination with interpolation to generate triangles that represent the surface, and the resulting geometry is combined into the output data set. We used 10 different isovalues for a single visualization cycle.

3.3.2 Threshold. The threshold algorithm iterates over every cell in the data set and compares it to a specified value or range of values. Cells containing the value are included in the output data set, while cells not containing the value are removed.

3.3.3 Spherical Clip. Spherical clip culls geometry within a sphere specified by an origin and a radius. The algorithm iterates over each cell and finds the distance of that cell from the center of the sphere. Cells completely inside the sphere are omitted from the output data set, while cells completely outside the sphere are retained in entirety, and passed directly to the output. If the cell contains the surface of the sphere, then the cell is subdivided into two parts, with one part inside the sphere and the other part outside the sphere, and each part is handled as before.

3.3.4 Isovolume. Isovolume and clip are similar algorithms. Instead of an implicit function (e.g., sphere), an isovolume evaluates each cell within a scalar range. Cells completely inside the scalar range are passed directly to the output, and cells completely outside the scalar range are removed from the output. If the cell lies partially inside and outside the scalar range, the cell is subdivided and the part outside the range is removed.

3.3.5 Slice. A slice cuts the data set on a plane, resulting in a two-dimensional data set. In order to create the slice, a new field is created on the data set representing the signed distance field from the plane (e.g., if the signed distance is 0, then the point is on the plane). Then, the contour algorithm evaluates the field at an isovalue of 0, resulting in a topologically two-dimensional plane. In this study, we evaluated three slices on the x-y, y-z, and x-z planes, resulting in a three-dimensional data set.

3.3.6 Particle Advection. The particle advection algorithm advects massless particles through a vector field. Particles are seeded throughout the data set, and advected for a user-specified number of steps. For this study, we advected the particles through a steady state (i.e., a single time step). The algorithm outputs a data set representing the path of each particle through the number of steps in the form of lines (i.e., streamlines).

3.3.7 Ray Tracing. Ray tracing is a rendering method that iterates over pixels in the image. Rays are intersected with the data set to find the nearest intersection. Ray tracing uses a spatial acceleration structure to minimize the amount of intersection tests that are performed on the data set. If an intersection is found, then a color is determined by the scalar field. The output of the ray tracing algorithm is an image. For this study, we created an image database consisting of 50 images per visualization cycle generated from different camera positions around the data set.

3.3.8 Volume Rendering. Volume rendering is another rendering method that iterates over pixels in the image. Rays step through the volume and sample scalar values at regular intervals. Each sample is mapped to a color containing a transparency component, and all samples along the ray are blended together to form the final color. For this study, we created an image database consisting of 50 images per visualization cycle generated from different camera positions around the data set.

3.4 Experimental Overview

In the following subsections, we discuss the study overview and methodology for our experiments.

3.4.1 Software Framework.

Our software infrastructure included VTK-m and Ascent. VTK-m [49] is an open-source library of scientific visualization algorithms designed for shared-memory parallelism. Its algorithms are implemented using a layer of abstraction enabling portable performance across different architectures. It is an extension of the Visualization ToolKit (VTK) [64], a well-established open-source library of visualization algorithms that form the basis of VisIt [17] and ParaView [10]. For this study, we configured VTK-m with Intel's Thread Building Blocks (TBB) [60] for thread-level parallelism.

The Ascent [39, 40] in situ framework is part of the multi-institutional project known as ALPINE. Ascent is a flyweight, open-source in situ visualization framework designed to support VisIt's LibSim [68] and ParaView's Catalyst [23]. Of the three included multi-physics proxy applications, we used CloverLeaf [4, 46], a hydrodynamics simulation, tightly coupled with the visualization. That is to say, the simulation and visualization alternate while using the same resources.

3.4.2 Hardware Architecture.

We used the *RZTopaz* supercomputer at Lawrence Livermore National Laboratory to conduct our experiments. Each node contains 128 GB of memory and two Intel Xeon E5-2695 v4 dual-socket processors executing at a base clock frequency of 2.1 GHz (120W thermal design power, or TDP). The Turbo Boost clock frequencies range from 2.6 GHz to 3.3 GHz. Each hyper-threaded processor has 18 physical cores.

On LLNL systems, the `msr-safe` [3] driver provides an interface for sampling and controlling processor power usage, among other performance counters, via 64-bit model-specific registers. On this Broadwell processor, the power can be capped from 120W (TDP) down to 40W using Intel's Running Average

Power Limit technology (RAPL) [19]. Then, the processor adjusts the operating frequency to guarantee the desired power cap.

3.4.3 Study Factors.

Our study consisted of three phases and 288 total test configurations. Each test was launched using a single node and a single MPI process for maximum memory allocation. Shared-memory parallelism was enabled with VTK-m. We varied the following parameters for this study:

- **Processor power cap** (9 options): Enforce a processor-level (cores, cache) power cap ranging from 120W (TDP) down to 40W in increments of 10W using Intel's RAPL.
- **Visualization algorithm** (8 options): The representative set of algorithms explored are contour, threshold, spherical clip, isovolume, slice, particle advection, ray tracing, and volume rendering.
- **Data set size** (4 options): The CloverLeaf data set sizes used per node are 32^3, 64^3, 128^3, and 256^3. The total number of cells ranged from $32,768$ to $16,777,216$.

3.4.4 Methodology.

This study consisted of three phases. Phase 1 studied a base case, and subsequent phases studied the impacts of varying one of the study factors listed in Subsection 3.4.3.

3.4.4.1 Phase 1: Processor-Level Power Cap.

Phase 1 varied the processor-level power caps and studied the behavior of the contour algorithm implemented in VTK-m. With this phase, we extended a previous finding [35], which determined baseline performance for isosurfacing by explicitly setting CPU frequencies. This phase consisted of nine tests.

Test Configuration: (Contour algorithm, 128^3 data set size) × 9 processor power caps

3.4.4.2 Phase 2: Visualization Algorithm.

In this phase, we continued varying processor-level power caps, and added variation in visualization algorithm. It consisted of 72 tests, nine of which were studied in Phase 1.

Test Configuration: (128^3 data set size) × 9 processor power caps × 8 visualization algorithms

3.4.4.3 Phase 3: Data Set Size.

In this phase, we add variation in data set size. It consisted of 288 tests, of which nine were studied in Phase 1 and 63 were studied in Phase 2.

Test Configuration: 9 processor power caps × 8 visualization algorithms × 4 data set sizes

3.5 Definition of Metrics

This section defines the variables and metrics that will be used in the following results section.

3.5.1 Abstract Case.

Assume a visualization algorithm takes T_D seconds to run at the default power (TDP) of P_D watts. As the power cap is reduced, the same visualization algorithm now takes T_R seconds to run with a power cap of P_R watts. The following derived terms are used to explain our results:

- $P_{ratio} = P_D/P_R$: This is the ratio of power caps. If the processor-level power cap is reduced by a factor of 2, then $P_{ratio} = 2$.
- $T_{ratio} = T_R/T_D$: This is the ratio of execution times. If the algorithm takes twice as long to run, then $T_{ratio} = 2$.

- $F_{ratio} = F_D/F_R$: This is the ratio of CPU frequencies. If the frequency was twice as slow, then $F_{ratio} = 2$.

These terms have been defined such that all ratios will be greater than 1. To accomplish this, P_{ratio} and F_{ratio} have the default value in the numerator and the reduced value in the denominator, while T_{ratio} has them reversed. Inverting the ratio simplifies our comparisons.

Using our three ratios, we can make the following conclusions. First, if T_{ratio} is less than P_{ratio}, then the algorithm was sufficiently data intensive to avoid a slowdown equal to the reduction in power cap. In addition, users can make a tradeoff between running their algorithm T_{ratio} times slower and using P_{ratio} less times power. Alternatively, this ratio enables us to optimize performance under a given power cap. Second, the relationships between F_{ratio} and P_{ratio} and T_{ratio} and F_{ratio} will be architecture-specific. Enforcing a power cap will lower the CPU frequency, however, the reduction in frequency will be determined by the processor itself. The reduction in clock frequency may slowdown the application proportionally (if the application is compute-bound) or not at all (if the application is memory-bound). The results in Section 3.6 present the ratios for a particular Intel processor (i.e., Broadwell), but this relationship may change across other architectures.

3.5.2 Performance Measurements. To collect power usage information, the energy usage of each processor in the node is sampled every 100 ms throughout the application (i.e., simulation and visualization) execution. The power usage for each processor is calculated by dividing the energy usage (contained in a 64-bit register) by the elapsed time between samples. In addition to energy and power counters, we also sample fixed counters, frequency-related counters, and two programmable counters — last level cache misses and references. From these counters, we can derive the following metrics. We show the derivation of these metrics using the Intel-specific performance counter event names [5], where applicable.

- Effective CPU frequency = `APERF/MPERF`
- Instructions per cycle (IPC) = `INST_RET.ANY/CPU_CLK_UNHALT.REF_TSC`
- Last level cache miss rate = `LONG_LAT_CACHE.MISS/LONG_LAT_CACHE.REF`

3.5.3 Efficiency Metric. We leverage a rate in terms of the size of the input (i.e., data set size) rather than speedup for comparing the efficiency of one visualization algorithm to another. If the speedup of a parallel algorithm is defined as $\frac{T_{n,1}}{T_{n,p}}$, then one must know the serial execution time of the algorithm. This is challenging with increasingly complex simulations running at higher concurrency levels. Instead, we assess speedup using a rate originally proposed by Moreland and Oldfield [32, 51]. They express the rate in terms of the data set size, n, as follows: $\frac{n}{T_{n,p}}$.

The higher the resulting rate, the more efficient the algorithm. Because the rate is computed using the size of the data set, we only compare those algorithms that iterate over each cell in the data set (e.g., contour, spherical clip, isovolume, threshold, and slice). At higher concurrencies, an algorithm with good scaling will show an upward incline, then will gradually flatten from the perfect efficiency curve.

3.6 Results

In this section, we describe the results from the phases detailed in Section 3.4.4.

Contour					
P	P_{ratio}	T	T_{ratio}	F	F_{ratio}
120W	1.0X	33.477s	1.00X	2.55GHz	1.00X
110W	1.1X	33.543s	1.00X	2.41GHz	1.06X
100W	1.2X	33.579s	1.00X	2.55GHz	1.00X
90W	1.3X	33.519s	1.00X	2.55GHz	1.00X
80W	1.5X	33.617s	1.00X	2.54GHz	1.01X
70W	1.7X	30.371s	0.91X	2.54GHz	1.00X
60W	2.0X	30.394s	0.91X	2.50GHz	1.02X
50W	2.4X	31.066s	0.93X	2.52GHz	1.01X
40W	3.0X	39.198s	1.17X	2.07GHz	1.23X

Table 10. The slowdown for the contour algorithm as the processor power cap is reduced. The configuration used for this algorithm is a data set size of 128^3. P is the enforced processor power cap. T is the total execution time in seconds for the contour algorithm over all visualization cycles. F is the effective CPU frequency given the power cap P. A 10% slowdown (denoted in red) does not occur for this algorithm until the lowest power cap.

3.6.1 Phase 1: Processor-Level Power Cap.

In this phase, we fix all study factors while varying the power cap in order to achieve a baseline performance for subsequent phases. Specifically, we use the following configuration: contour algorithm and a data set size of 128^3. We present the results in Table 10.

When the default power cap of 120W is applied to each processor, the simulation spends a total of 33.477 seconds executing a contour filter and the total power usage of both processors is 120W (88% of total node power). As we gradually reduce the processor-level power cap, the execution time remains constant (e.g., T_{ratio} is 1X). Since the algorithm is data intensive, it does not use a lot of power. Applying a more stringent power cap does not affect the overall performance as it is not using power equivalent to the desired power cap, so the underlying frequency does not need to slowdown.

Once the power cap is reduced by a factor of 3X (from 120W down to 40W), we see a change in the execution time and CPU frequency by a factor of 1.17X and

1.23X, respectively. At 40W, the algorithm takes longer to run (since the frequency is also reduced to maintain the desired power usage), but the algorithm did not slowdown proportionally to the reduction in power by a factor of 3. This confirms our finding in [35], where we determined that the contour algorithm was sufficiently data intensive to avoid slowing down proportional to the CPU clock frequency.

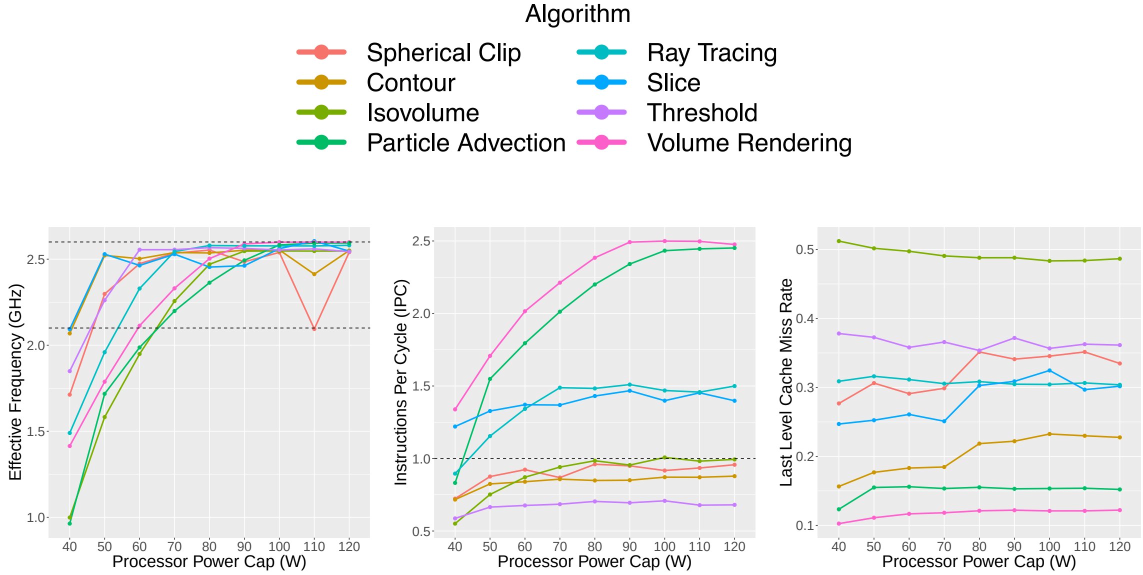
Running with the lowest power cap does not impact the performance for contour. If doing a contour post hoc, the user can request the lowest power, leaving power for other applications that are competing for the same compute resources. If doing a contour in situ, the runtime system may leverage the low power characteristic and dynamically allocate less power to the visualization phase, allowing more power to be dedicated to the simulation.

3.6.2 Phase 2: Visualization Algorithm. In Phase 1, we determined that the contour algorithm is sufficiently memory-bound to avoid a change in execution time until a severe power cap. In Phase 2, we want to explore if this data intensive trend is common across other algorithms, so we extend the previous phase and vary the visualization algorithm. We continue to focus on a data set size of 128^3. We identify two clear groupings: those algorithms that are insensitive to changes in power (power opportunity), and those algorithms that are sensitive to changes in power (power sensitive). We will discuss the two categories in more detail below.

3.6.2.1 Power Opportunity Algorithms. The algorithms that fall into the power opportunity category are contour (discussed in the previous section), spherical clip, isovolume, threshold, slice, and ray tracing. Table 11 shows the slowdown in execution time and CPU frequency for all algorithms. The power opportunity algorithms do not see a significant slowdown (of 10%, denoted in

	P	120W	110W	100W	90W	80W	70W	60W	50W	40W
	P_{rat}	1X	1.1X	1.2X	1.3X	1.5X	1.7X	2X	2.4X	3X
1	T_{rat}	1X	1X	1X	1X	1X	0.91X	0.91X	0.93X	1.17X
	F_{rat}	1X	1.06X	1X	1X	1.01X	1X	1.02X	1.01X	1.23X
2	T_{rat}	1X	1.01X	1.03X	1.02X	1X	1.05X	1.02X	1.18X	1.48X
	F_{rat}	1X	1.21X	1X	1.02X	1X	1X	1.03X	1.11X	1.48X
3	T_{rat}	1X	1.01X	0.99X	1.04X	1.02X	1.06X	1.14X	1.30X	1.81X
	F_{rat}	1X	1X	1X	1X	1.03X	1.13X	1.31X	1.61X	2.55X
4	T_{rat}	1X	0.98X	0.98X	1X	0.99X	0.99X	1.02X	1.08X	1.31X
	F_{rat}	1X	0.99X	1X	0.99X	0.99X	1X	1X	1.12X	1.38X
5	T_{rat}	1X	0.98X	1X	0.99X	0.98X	1.02X	1.04X	1.03X	1.26X
	F_{rat}	1X	0.98X	0.99X	1.03X	1.04X	1.01X	1.03X	1.01X	1.22X
6	T_{rat}	1X	1X	0.99X	0.99X	1X	1.01X	1.10X	1.31X	1.75X
	F_{rat}	1X	1X	1X	1X	1X	1.01X	1.11X	1.32X	1.73X
7	T_{rat}	1X	1X	1.01X	1.05X	1.11X	1.21X	1.34X	1.57X	3.12X
	F_{rat}	1X	1X	1X	1.04X	1.10X	1.18X	1.31X	1.51X	2.69X
8	T_{rat}	1X	1X	0.99X	1X	1.04X	1.12X	1.23X	1.46X	1.86X
	F_{rat}	1X	1X	1X	1X	1.04X	1.12X	1.23X	1.45X	1.84X

Table 11. Slowdown factor for all algorithms with a data set size of 128^3. The mapping between number and algorithm is defined as follows: (1) Contour, (2) Spherical Clip, (3) Isovolume, (4) Threshold, (5) Slice, (6) Ray Tracing, (7) Particle Advection, and (8) Volume Rendering. Slowdown is calculated by dividing execution time at 40W by execution time at 120W. Numbers highlighted in red indicate the first time a 10% slowdown in execution time or frequency occurs due to the processor power cap P.



(a) Effective frequency

(b) IPC

(c) Last level cache miss rate

Figure 8. Effective frequency (GHz), instructions per cycle (IPC), and last level cache miss rate for all algorithms as the processor power cap is reduced. For each algorithm, we use a data set size of 128^3.

red) until P_{ratio} is at least 2X or higher. These algorithms are data-bound — the bottleneck is the memory subsystem, not the processor — so reducing the power cap does not significantly impact the overall performance. This is confirmed since T_{ratio} is less than P_{ratio}.

When looking at the CPU operating frequency in Figure 8a, we see that all algorithms, regardless of whether it is in the power opportunity or power sensitive class, run at the same frequency of 2.6 GHz at a 120W power cap, which is the maximum turbo frequency for this architecture when all cores are active. The differences across the algorithms are seen in the rate at which the frequency declines because of the enforced power cap and the power usage of the algorithms.

The default power usage varies across visualization algorithms, ranging from as low as 55W up to 90W per processor. For algorithms that do not consume

TDP, the processor decides it can run in turbo mode (i.e., above 2.1 GHz base clock frequency) to maximize performance. Once the power cap is at or below the power usage of the algorithm, the operating frequency begins to drop because the processor can no longer maintain a high frequency without exceeding the power cap. For algorithms with a high power usage, the frequency will start dropping at power caps close to TDP. For algorithms with a low power usage (e.g., contour, described previously), the processor runs in turbo mode for most power caps to maximize performance. It is not until the lowest power cap of 40W that we see a reduction in the clock frequency for contour.

Figure 8b shows the average instructions per cycle (IPC) for all algorithms. The dotted line drawn at an IPC of 1 shows the divide between compute-bound algorithms (IPC $>$ 1) and memory-bound algorithms (IPC $<$ 1). Spherical clip, contour, isovolume, and threshold make up one class of algorithms. Their IPC is characteristic of a data-bound algorithm, and their power usage is also very low, so the decrease in IPC is not seen until the lowest power cap of 40W. Threshold is dominated by loads and stores of the data, so it has a low IPC value. Contour and isovolume have higher IPC values (out of this group of algorithms) because it calculates interpolations.

Another class of algorithms (with respect to IPC) consists of ray tracing and slice, which have an IPC that falls into compute-bound range. Although they have an IPC larger than 1, they have low power usage and their performance remains unchanged until low power caps. For this study, we created an image database of 50 rendered images (either with volume rendering or ray tracing) per visualization cycle to increase algorithm time. Investigating ray tracing further, we discover that the execution time includes the time to gather triangles and find external faces, build a spatial acceleration structure, and trace the rays. Tracing the rays is the most compute intensive operation within ray tracing, but it is being dominated by the data intensive operations of gathering triangles and building the spatial acceleration structure. As such, ray tracing behaves similarly to the cell-centered algorithms in this category: spherical clip, threshold, contour, isovolume, and slice. It also has the best slowdown factor.

Slice has a higher IPC than contour, which is expected since it is doing a contour three times. Three-slice creates three slice planes on x-y, y-z, and z-x intersecting the origin. Consequently, the output size is fixed for any given time step. Three-slice under the hood uses contour, but differs in the fact that each slice plane calculates the signed distance field for each node on the mesh, which is compute intensive.

Figure 8c shows the last level cache miss rate for all algorithms, and is the inverse of Figure 8b. Isovolume has the highest last level cache miss rate, indicating that a high percentage of its instruction mix is memory-related. Because of the high miss rate, the isovolume algorithm spends a lot of time waiting for memory requests to be satisfied. Memory access instructions have a longer latency than compute instructions. Therefore, it cannot issue as many instructions per cycle, and has a low IPC.

Another interesting metric to investigate is shown in Figure 9, which is the number of elements (in millions) processed per second. Because the power usage of these algorithms is low, the denominator (e.g., seconds) stays constant for most power caps, yielding a near constant rate for each algorithm. At severe power caps, the number of elements processed per second declines because the algorithm incurs slowdown. Algorithms with very fast execution times will have a high rate, while algorithms with a longer execution time will have a low rate.

3.6.2.2 Power Sensitive Algorithms. The power sensitive algorithms are volume rendering and particle advection. They consume the most power at roughly 85W per processor. When the power cap drops below 85W, the frequency starts dropping as it can no longer maintain the desired power cap at the 2.6 GHz frequency. Thus, there are slowdowns of 10% at 70W and 80W, respectively, which is at a higher power cap than the power opportunity algorithms. These algorithms not only have the highest IPC values overall as shown in Figure 8b (peak IPC of 2.68, highly compute-bound), but also have the biggest change in IPC as the power cap is reduced. Such algorithms are dominated by the CPU, so a reduction in power greatly impacts the number of cycles it takes to issue the same set of instructions (i.e., slows down the algorithm).

Figure 8b coupled with Figure 8c shows volume rendering and particle advection with a high IPC because they have the lowest last level cache miss rate (i.e., better memory performance). Additionally, more instructions can be retired per cycle because the processor is not stalled waiting on memory requests to be satisfied (i.e., high IPC). Everything fit into cache, and IPC was changing drastically with changing power caps, so we can infer that IPC behavior was dominated by compute instructions.

3.6.2.3 Key Takeaways. For most of the algorithms explored in this chapter, the power cap has little effect on performance. This is because the power usage of visualization algorithms is low compared to typical HPC applications. For similar algorithms, we can run them with the lowest power cap without impacting performance. In a larger scheme where we are running the simulation

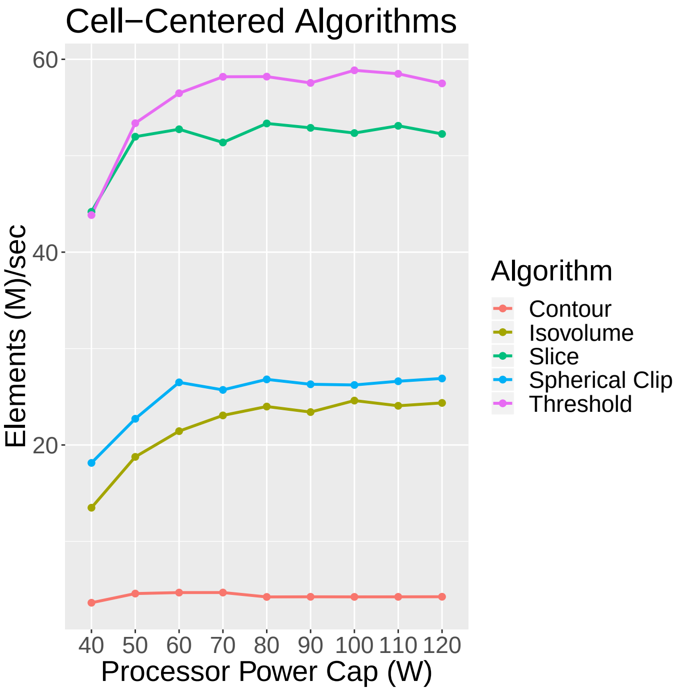


Figure 9. Elements processed per second for cell-centered algorithms using 128^3 data set size.

and visualization on the same resources, we can more intelligently allocate power between the two, rather than using a naïve scheme of evenly distributing the power. Said another way, we can allocate most of the power to the power-hungry simulation, leaving minimal power to the visualization, since it does not need it. Additionally, we find two of the algorithms explored (volume rendering and particle advection) have high power usage, consistent with typical HPC applications. These algorithms have a poor tradeoff between power and performance. There may be other algorithms that behave similarly.

3.6.3 Phase 3: Data Set Size. Phase 3 extended Phase 2 by varying over data set size. Table 12 shows the results for all algorithms using a data set size of 256^3. This table can be compared to Table 11 in Section 3.6.2.

	P	120W	110W	100W	90W	80W	70W	60W	50W	40W
	P_{rat}	1X	1.1X	1.2X	1.3X	1.5X	1.7X	2X	2.4X	3X
Contour	T_{rat}	1X	1X	1X	1X	1X	1X	1.05X	1.19X	1.71X
	F_{rat}	1X	1X	1X	1X	1.01X	0.99X	1.07X	1.18X	1.52X
Spherical Clip	T_{rat}	1X	1.01X	1.01X	1.05X	1.01X	1.10X	1.16X	1.41X	2.13X
	F_{rat}	1X	1X	1X	1X	1.01X	1.05X	1.17X	1.42X	1.95X
Isovolume	T_{rat}	1X	0.98X	0.97X	1.01X	1.01X	1.01X	1.17X	1.33X	1.76X
	F_{rat}	1X	1X	0.97X	1X	1X	1.05X	1.11X	1.32X	1.79X
Threshold	T_{rat}	1X	1.02X	0.99X	0.99X	0.98X	1.09X	1.16X	1.30X	1.53X
	F_{rat}	1X	1.01X	1.02X	1.02X	1.02X	1.05X	1.17X	1.38X	1.66X
Slice	T_{rat}	1X	1X	0.99X	0.99X	1X	1X	0.99X	1.33X	1.69X
	F_{rat}	1X	0.98X	1.01X	0.93X	1.01X	0.98X	1.01X	1.24X	1.44X
Ray Tracing	T_{rat}	1X	1X	1X	1.01X	1X	1.02X	1.10X	1.28X	2X
	F_{rat}	1X	1X	1X	1X	1X	1.01X	1.10X	1.29X	2.05X
Particle Advection	T_{rat}	1X	1X	1.03X	1.07X	1.14X	1.39X	1.64X	2.13X	2.67X
	F_{rat}	1X	1X	1.02X	1.06X	1.13X	1.35X	1.57X	2.05X	2.56X
Volume Rendering	T_{rat}	1X	1X	1X	1X	1.06X	1.13X	1.24X	1.45X	1.81X
	F_{rat}	1X	1X	1X	1X	1.06X	1.13X	1.23X	1.45X	1.82X

Table 12. Slowdown factor for all algorithms with a data set size of 256^3. Slowdown is calculated by dividing execution time at 40W by execution time at 120W. Numbers highlighted in red indicate the first time a 10% slowdown in execution time or frequency occurs due to the processor power cap P.

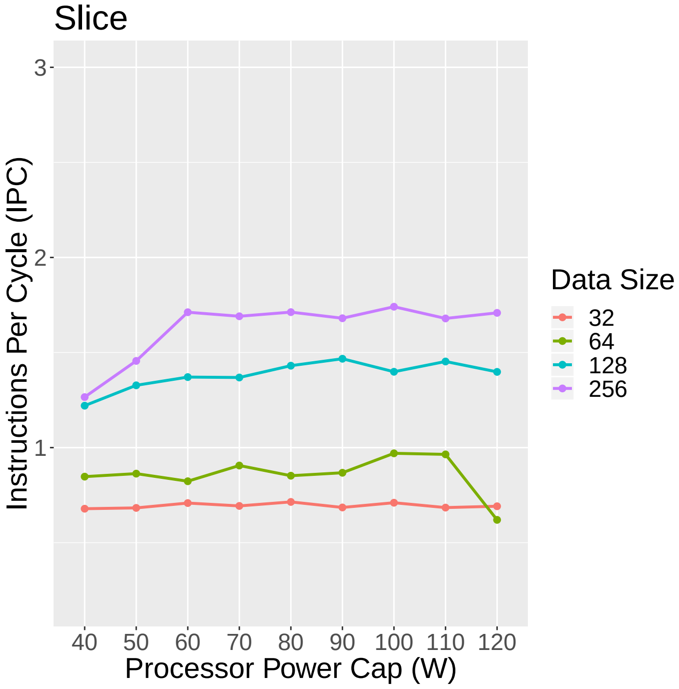


Figure 10. This category of algorithms sees an increase in IPC as the data set size increases. Algorithms that fall into this category are slice, contour, isovolume, threshold, and spherical clip.

As the data set size is increased from 128^3 in Table 11 to 256^3 in Table 12, T_{ratio} changes across algorithms. For the power opportunity algorithms identified in Phase 2, T_{ratio} exceeds 1.1X at higher power caps with larger data set sizes. As an example, spherical clip did not have significant slowdowns until 50W with a data set size of 128^3, but now has similar slowdowns at 70W. Other algorithms in this category, such as contour, threshold, slice, and ray tracing, now slowdown at 60W and 50W with a data set size of 256^3 instead of slowing down at 40W with a data set size of 128^3.

Depending on the algorithm, the IPC may increase or decrease as the data set size is increased. Figure 10, Figure 11, and Figure 12 show the IPC for three different algorithms over all power caps and data set sizes. The IPC of the three different algorithms shown in the figures represent three categories.

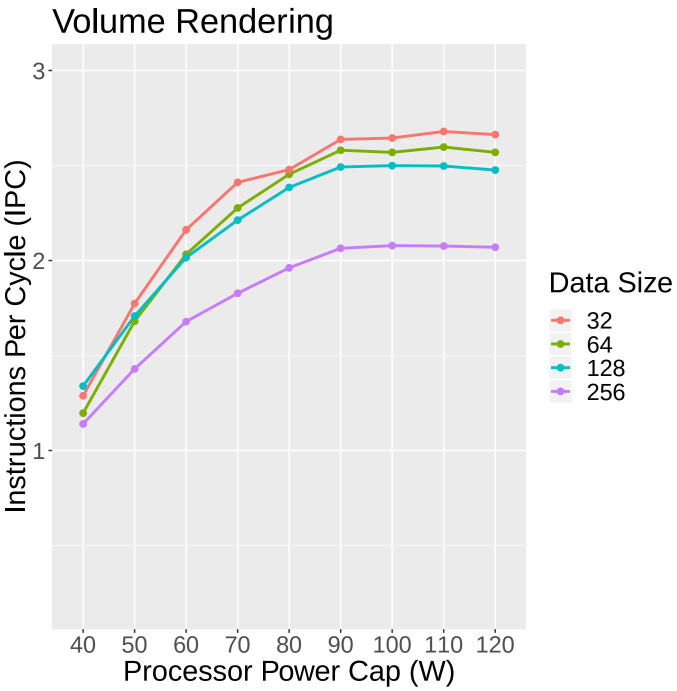


Figure 11. This category of algorithms sees an increase in IPC as the data set size decreases. Volume rendering is the only algorithm exhibiting this behavior.

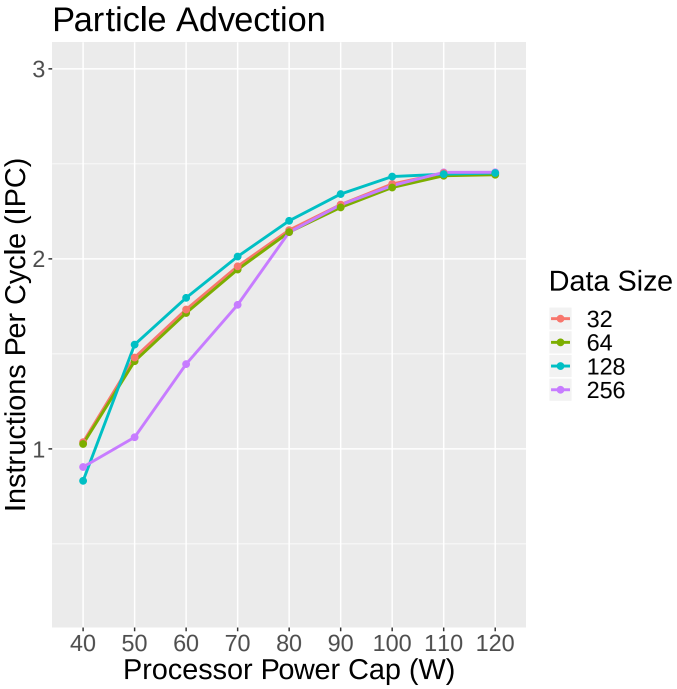


Figure 12. This category of algorithms see no change in IPC as the data set size changes. Algorithms exhibiting this behavior are particle advection and ray tracing.

The first category consists of slice, contour, isovolume, threshold, and spherical clip. As the data set size increases, the IPC also increases for these algorithms as shown in Figure 10. Particularly for slice and spherical clip, the number of instructions increases with a larger number of elements (i.e., bigger data set size) because for each cell, the algorithm computes the signed distance. The other algorithms in this category — contour, isovolume, and threshold — iterate over each cell, so the number of comparisons will also increase (i.e., for threshold, keep this cell if it meets some criteria, else discard). Algorithms in this category tend to have lower IPC values. These algorithms contain simple computations, so the loads and stores of the data (i.e., memory instructions) dominate the execution time.

The second category contains volume rendering, which shows an inverse relationship between data set size and IPC as shown in Figure 11. Here, the IPC increases as the data set size decreases. As an example, as the data set size increases from 128^3 to 256^3 (8X bigger), the IPC only drops by 20% going from 2.5 down to 2. On average, the IPC of volume rendering is higher than any of the other algorithms explored in this chapter. Volume rendering is an image-order algorithm and has a high number of floating point instructions resulting in high power and high IPC.

The third category consists of algorithms whose IPC does not change with increases in data set sizes as illustrated in Figure 12. The algorithms identified here are particle advection and ray tracing. For particle advection, we held the following constant regardless of the data set size: the same number of seed particles, step length, and number of steps. Because we chose to keep these parameters consistent, particles may get displaced outside the bounding box depending on the data set size. When particles are displaced outside the bounding box, they terminate, and there is no more work to do for that particle.

Particle advection has a high IPC value, and a high power consumption. The advection implementation uses the Runge-Kutta, which is the 4th order method to solve ordinary differential equations. This method is computationally very efficient and has a large number of high power instructions.

The ray tracing algorithm consists of three steps: building a spatial acceleration structure, triangulation, and tracing the rays. The amount of computation does not scale at the same rate as the data set size. An increase in the data set size by a factor of 8 (going from 128^3 to 256^3) results in only a 4X increase in the number of faces encountered.

3.7 Summary of Findings

One of the key goals of this chapter was to identify the impacts of various factors on power usage and performance of visualization algorithms in order to better inform scientists and tool developers. This chapter is an extension of the work in Chapter II, which performed an initial exploration of the feasibility of achieving energy and power savings with reduced clock frequencies. With this chapter, we performed a more in-depth study of the power and performance tradeoffs for a representative set of visualization algorithms. Specifically, this chapter identifies the environment in which variation occurs in visualization. We summarize the findings from the previous sections here.

On varying processor power caps (Section 3.6.1):

- The VTK-m implementation of contour is sufficiently data intensive to avoid a significant slowdown from reducing the power cap. This extends a previous finding [35] which set CPU frequencies and used a custom implementation, and is additionally noteworthy since our study uses a general toolkit designed to support a wide variety of algorithms and data types.

- The execution time remains unaffected until an extreme power cap of 40W, creating opportunities for redistributing power throughout the system to more critical phases or applications.

On comparing different visualization algorithms (Section 3.6.2):

- Most of the visualization algorithms studied in this chapter consume low amounts of power, so they can be run under a low power cap without impacting performance. These algorithms have lower IPC values, characteristic of data-bound workloads.

- Two of the explored algorithms consume higher power, similar to what we commonly see of traditional compute-bound benchmarks, such as Linpack. These algorithms will see significant slowdown from being run at a lower power cap, up to 3.2X. As such, the slowdown begins around 80W, roughly 67% of TDP. These algorithms have high IPC values, which are characteristic of compute-bound workloads.

On varying the input data set size (Section 3.6.3):

- Larger data set sizes result in poorer tradeoffs for performance. With a higher data set size, these algorithms start to slowdown at higher power caps. So instead of seeing a 10% slowdown at 50W with a data set size of 128^3, the slowdown begins at 70W for a data set size of 256^3.

- For the algorithms that were significantly compute-bound (and consuming high amounts of power), the change in data set size does not impact the power usage.

These recipes can be applied to two use cases in the context of a power-constrained environment. First, when doing post hoc visualization and data analysis on a shared cluster, requesting the lowest amount of power will leave more for other power-hungry applications. Second, when doing in situ visualization, appropriately provisioning power for visualization can either leave more power for the simulation or improve turn-around time for the visualization pipeline. These results can be integrated into a job-level runtime system, like PaViz [38] or GEOPM [2, 22], to dynamically reallocate the power to the various components within the job. By providing more tailored information about the particular visualization routine, the runtime system may result in better overall performance.

3.8 Conclusion

Our study explored the impacts of power constraints on scientific visualization algorithms. We considered a set of eight representative algorithms, nine different processor-level power caps, and four data set sizes, totaling 288 total test configurations. We believe the results of the study provide insights on the behavior of visualization algorithms on future exascale supercomputers. In particular, this study showed that visualization algorithms use little power, so applying an extremely low power cap will not impact the performance. (Refer back to Section 3.7 for specific findings.) We believe these findings can be used to dynamically reallocate power between competing applications (i.e., simulation and visualization) when operating under a power budget. The runtime system would identify visualization workflows that are compute- or data-bound and allocate power accordingly, such that the scarce power is used wisely.

This study suggests several interesting directions for future work. Our results identified two different classes of algorithms. These findings can be applied to other visualization algorithms in making informed decisions about how to allocate power for visualization workflows. While most of the algorithms explored in this chapter consumed low power and were data-bound, we did find two algorithms (particle advection and volume rendering) that did not fall into this category. This indicates there may be other visualization algorithms that might fall into the category of high power usage and compute intensity. Another extension of this work is to explore how the power and performance tradeoffs for visualization algorithms compare across other architectures that provide power capping mechanisms. Other architectures may exhibit different responses to power limits, and so it is unclear how the underlying architecture will affect the algorithms.

CHAPTER IV

POWER-AWARE RUNTIME SYSTEM FOR VISUALIZATION

Most of the text in this chapter comes from [38], which was a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself. Barry Rountree primarily wrote the overview of power in high performance computing, although I provided some contributions as well. The related work section on volume rendering was written by Matthew Larsen, while I wrote the sections on power and the intersection of scientific visualization and power. Matthew Larsen provided guidance on the software integration between performance predictions and the runtime system. I developed the power-aware runtime system, decision strategies, and necessary monitoring infrastructure, designed and executed the study, and was the primary contributor to the writing of the overall paper.

This chapter demonstrates how we can exploit the variation in visualization workloads by dynamically reallocating power. By exposing application-level information from the visualization routine to a central manager, additional performance can be realized. We implemented a runtime system that uses an existing performance model to dynamically reallocate power across nodes within a job. This study shows that using prediction to adapt power across nodes is an effective strategy for optimizing performance for visualization applications, which can be more unpredictable and irregular in nature.

4.1 Motivation

Power is a critical challenge in achieving the next generation of high performance computing (HPC) systems. Specifically, scaling today's technologies to higher concurrency may lead to excessive power consumption costs. As a result, the entire HPC environment — from processors to software applications to runtime systems — is being re-evaluated for power efficiency.

For this research, an important premise is that simulations and visualization routines need to adapt to a power-limited environment, meaning nodes will have their power usage capped. Motivation for this premise can be found in Section 4.2. The main contribution of this chapter is PaViz, a power-adaptive visualization framework that enables performance improvements when power is a scarce resource. To understand its efficacy, we ran PaViz on a rendering algorithm that incorporated runtime predictions based on an accurate performance model. Our study focuses on rendering for two reasons. First, rendering is a ubiquitous operation for visualization. Second, rendering is a particularly interesting algorithm to study, since its workloads are highly variable depending on input parameters. In terms of findings, we found that, in limited power budget environments, adapting power based on performance model predictions led to speedups of up to 33% while using the same power overall.

The rest of this chapter is organized as follows. We present an overview of power with respect to HPC in Section 4.2. The related work is detailed in Section 4.3. The contributions of the PaViz framework and power scheduling strategies are discussed in Section 4.4. Section 4.5 identifies the study parameters. We evaluate the benefits of PaViz in Section 4.6. In Section 4.7, we summarize our findings and present some ideas for future exploration in this space.

4.2 HPC and Power Overview

Current cluster designs assume sufficient power will be available to simultaneously run all compute nodes at their maximum thermal design point (TDP). Said another way, TDP is the maximum power a given node will ever consume. As power requirements for clusters move into the range of dozens of MegaWatts, the strategy of allocating power for all nodes to run at TDP becomes untenable. Very few applications run at TDP, and provisioning very large systems as if most did both wastes power capacity and unnecessarily constrains the size of the cluster.

Overprovisioning [57], short for hardware overprovisioning, is one solution to improve power utilization. In such a design, we increase the compute capacity (i.e., number of nodes) of the system, but, in order to not exceed the system power allocation, not every node will be able to run at TDP simultaneously. For example, the Vulcan supercomputer at Lawrence Livermore National Laboratory was allocated for 2.4 MW at TDP, but the vast majority of applications that run on that machine did not exceed 60% of the allocated power (1.47 MW average power consumption). Thus, the strategy of allocating TDP to every node fails to take advantage of nearly 1 MW of power on average. An overprovisioned approach would use 40% more nodes, consuming all allocated power and reducing trapped capacity [69].

For overprovisioning to be successful, it must be complemented with a scheme to limit nodes' power usage, to ensure the total allocated power is never exceeded. One way to accomplish this is to uniformly cap the power available to each node, e.g., each node can use only up to 60% of its TDP. The result of applying such a power cap is that the processor operating frequency is reduced. The effect of reduced CPU frequency is variable; programs dominated by compute will slow down proportionally, while programs dominated by memory accesses may be unaffected altogether. Despite the slowdown in execution time for individual jobs, this strategy would lead to better power utilization and greater overall throughput.

The strategy of allocating power uniformly across nodes, however, is suboptimal. This is because the runtime behaviors of distributed applications can be highly variable across nodes. The nodes assigned the largest amount of work become a bottleneck and determine the overall performance of the application. On the other hand, nodes that are assigned the smallest amount of work finish quickly and sit idle until the other nodes have completed execution.

A better strategy is to actively assign power to where it will do the most good. This is the direction we pursue with this study. In an ideal scenario, we can assign the power such that all nodes finish executing at the same time despite varying workloads.

Overprovisioned systems lend themselves to multiple levels of optimization. At the job scheduling level, per job power bounds are allocated to optimize throughput and/or turn-around time [58]. Alternatively, there are dynamic optimizations to individual jobs that may be realized by rebalancing power and changing node configuration at runtime [2, 22, 47]. While our work fits into this runtime level, our primary contribution is demonstrating that additional performance may be realized by giving the runtime system deeper knowledge of the unique execution behaviors for visualization routines. Specifically, we use a performance model for volume rendering to better estimate the runtime behaviors, and show that we can improve performance on less predictable workloads.

4.3 Related Work

In the following subsections, we survey related work.

4.3.1 Volume Rendering. Volume rendering is an important set of rendering algorithms that enables visualization of an entire three dimensional scalar field, and volume rendering is widely used because it is capable of analyzing a large amount of data from many scientific disciplines. Volumetric ray casting [43] traces rays from the camera through a scalar field, sampling the volume at regular intervals, and accumulating color and opacity via a transfer function. This algorithm is embarrassingly parallel and is used in community driven visualization tools (e.g., VisIt [17] and ParaView [10]) and packages from hardware vendors (e.g., Intel's OSPRay [67] and NVIDIA's IndeX [1]). For our study, we chose volumetric ray casting because of its widespread use, and also because of its existing performance model [41].

4.3.2 Power. Some of the earliest solutions in addressing energy use in HPC were CPU MISER [29], Jitter [25], and Adagio [62]. These approaches used dynamic voltage and frequency scaling (DVFS) to make decisions between performance and energy at varying granularities. All three approaches were aimed at minimizing energy use with varying tolerances for increases in runtime. CPU MISER made CPU frequency decisions based on time intervals, and did not perform well when application behavior was less predictable. Jitter used iterations to identify the processor with the most work in order to slow down remaining processors, and this led to sub-optimal performance on applications where the critical path moved across processors within an iteration. Adagio's solution used task-based granularity to identify the critical path, thus minimizing performance degradation. For our study, we focus on the rendering work prior to MPI (i.e., prior to compositing), so we make decisions at an iteration-based granularity.

Processor manufacturer technologies for enforcing power caps (Intel's Running Average Power Limit (RAPL) [19], AMD TDP PowerCap [20], and IBM EnergyScale [30]) enable more recent efforts to focus on optimizing performance under a power bound. Conductor [47] used initial iterations to determine an ideal schedule of per-node power caps, thread concurrency, and per-core operating frequency. GEOPM [2, 22] is a production-grade runtime framework for optimizing performance under resource constraints. GEOPM, Conductor, and Adagio share similar goals and are collaborating to integrate technologies. GEOPM supports manual application markup as well as automated phase detection to dynamically reallocate power. Its architecture supports multiple plugins, but currently it does not support any particular policy targeted at in situ visualization.

Neither Conductor nor GEOPM can use application inputs to optimize performance. To the best of our knowledge, PaViz is the first runtime system to use visualization workloads, which behave differently than typical benchmarks than those used by Conductor and GEOPM. Specifically, we use rendering workload parameters — number of active pixels, camera position, image resolution — to predict execution time and optimize performance under a power bound.

4.3.3 Scientific Visualization and Power. Due to the I/O bandwidth limitations at exascale, visualization is moving away from a traditional post-processing method to in situ. In the post-processing method, the simulation writes out data to disk at regular time steps. Once the simulation has completed, the data is read back from disk for post-processing analysis and visualization using tools, such as VisIt [17] or ParaView [10]. As simulations increase in complexity, the amount of data they can write out increases exponentially, making it unsustainable to write out data with high temporal frequency. The critical challenge is saving enough data from the simulation without impacting fidelity or losing notable areas of interest.

In the in situ model, visualization and analysis occur alongside the simulation to mitigate the impacts of reduced I/O bandwidth. The data from the simulation is analyzed and visualized and the resulting images are written to disk, vastly reducing the total amount of data written to disk. Since power is a critical challenge to reaching the next generation of computing, research efforts have been dedicated to understanding the power profiles of this new analysis strategy, particularly with respect to how data is moved through the storage hierarchy [7, 61]. These works compared the power profiles of each pipeline, concluding that in situ drastically reduces energy usage by reducing the total runtime to complete the simulation and analysis.

Labasan et al. [35] provided an initial exploration of the various factors that may impact power and performance trade-offs for an isosurfacing algorithm implemented in two frameworks. This work studied the performance impacts of various parameters as the CPU operating frequency was gradually reduced. Similarly, work by Gamell et al. [27] also looked at the power-performance trade-offs of various parameters at scale.

4.4 Our Approach for Adaptive Power Scheduling

In an overprovisioned environment, the total power allocated to the machine is not enough to run all nodes at peak power simultaneously. The default strategy for handling this reduction in power is to uniformly assigned reduced power — if the total power is 50% of peak, then each node would be capped at 50% of its maximum power. The performance effects of this power cap will vary from node to node. In cases where the node was approaching maximum power, the slowdown would be greater, while in cases where the node was already using less than the power cap, there would be no effect in runtime. Therefore, since the rendering task is only as fast as the slowest processor, the choice of uniform reduction is poor. A better choice is to adapt the power assigned to each node based on how much work it has to do — nodes with lots of work get higher power caps and nodes with less work get lower power caps. In an ideal scenario, each node would complete rendering at the same time.

Adaptively assigning power is a non-trivial task. At its essence, it involves assessing how much work each processor needs to do. For our approach, we incorporate an existing rendering performance model [41]. When the performance model predicts a high rendering time, we assign more power, and when it predicts low rendering time, we assign less. In terms of specifics, we consider a family of strategies, detailed in the next subsection.

For volume rendering, the performance model predicts the rendering time by considering the camera configuration and data set size. The model is based on the observation that there are two distinct types of operations in volumetric ray casting, each with a cost determined by the hardware architectural factors. Operations that are associated with entering a new cell (e.g., loading nodal scalar values) occur with a frequency influenced by the size of the data set and the distance between samples relative to cell size, and operations that are associated with each sample (e.g., interpolating scalars and compositing colors) occur with a frequency influenced by the total data set spatial extents and sample distance. The combination of these operations represent the total amount of work per ray, and with an estimate of the number of rays that intersect the volume using camera parameters, a total amount of work for the entire image can be predicted.

Parameter	**Description**
n	Number of MPI tasks
pow_{node_min}	Minimum node power needed to execute job
pow_{avail}	Available power to allocate
ren_i	Predicted render time for task i
ren_{min}	Global minimum predicted render time among all n tasks
ren_{mean}	Global mean predicted render time among all n tasks
ren_{med}	Global median predicted render time among all n tasks
ren_{max}	Global maximum predicted render time among all n tasks

Table 13. Power scheduling strategy parameters.

Architectural costs for each of the two operations were calculated using multiple linear regression data gathered on the architecture on which the model was used [41]. In all, given a camera position, data set size, and sampling parameters, the time to render on a node could be predicted with high accuracy.

4.4.1 Power Scheduling Strategies.

In this section, we describe the power scheduling strategies used in this study. For this exploratory work, we implemented a handful of simple strategies, and evaluate their ability to improve performance.

Each scheduling strategy produces a scalar factor, which we use to assign a portion of the "available power" (denoted as pow_{avail}) to each node. This guarantees that the allocated job power budget is not exceeded as per-node power assignments are being made. The pow_{avail} is calculated by taking the difference between the specified power budget and the minimum power required to execute the job reliably (i.e., the minimum power needed to operate all nodes sufficiently).

4.4.1.1 **Min *Scheduling Strategy.***

This strategy uses the difference from the minimum estimated render time to determine the power allocation. For each predicted rendering runtime, the node power cap is determined as follows:

$$pow_{node\_min} + \frac{|ren_{min} - ren_i|}{\sum_{i=0}^{n-1} |ren_{min} - ren_i|} * pow_{avail}$$

We speculate that this strategy will produce the best speedups of all the strategies described in this section. Nodes that are furthest away from the minimum (i.e., highest render time, most work to be done) will be allocated a high amount of power, and this should produce the highest speedups in a balanced and imbalanced workload configuration, since the rendering task is only as fast as the slowest processor.

4.4.1.2 **Normalized *Scheduling Strategy.*** This strategy calculates node power assignments by the value of the predicted render time:

$$pow_{node\_min} + \frac{ren_i}{\sum_{i=0}^{n-1} ren_i} * pow_{avail}$$

This strategy behaves similarly to *Min*, since power assignments correlate with the render times. It assigns less aggressive power caps, since the computation does not take into account the global minimum of the predicted render times.

4.4.1.3 **Mean *Scheduling Strategy.*** This strategy uses the distance from the average estimated render time to assign power allocations.

$$pow_{node\_min} + \frac{|ren_{mean} - ren_i|}{\sum_{i=0}^{n-1} |ren_{mean} - ren_i|} * pow_{avail}$$

The intuition is that this strategy has no impact on performance when the rendering workload is evenly balanced. If the rendering workload is imbalanced, the *Mean* strategy may provide some benefits, but we speculate it will not produce as aggressive of a power schedule as the *Min* strategy, since the mean falls between all estimates.

4.4.1.4 **Median *Scheduling Strategy.*** This strategy uses the distance from the median estimated render time in making its power decision.

$$pow_{node\_min} + \frac{|ren_{med} - ren_i|}{\sum_{i=0}^{n-1} |ren_{med} - ren_i|} * pow_{avail}$$

We speculate that the *Median* strategy will perform similarly to the *Mean* strategy, since the median and mean will not differ significantly in our rendering configurations. We envision cases where the median is better for assigning more aggressive power caps than the mean, producing better speedups than the *Mean* strategy.

4.4.1.5* Max *Scheduling Strategy. This scheduling strategy uses the difference from the maximum estimated render time to determine the power allocation. For each predicted rendering runtime, the node power cap is determined as follows:

$$pow_{node\_min} + \frac{|ren_{max} - ren_i|}{\sum_{i=0}^{n-1} |ren_{max} - ren_i|} * pow_{avail}$$

Intuitively, this scheduling strategy will perform the worst of all implemented strategies as it rebalances power in a sub-optimal manner. It allocates higher power to nodes with only a small amount of work to complete (i.e., fast render time), and low power to nodes with long render times, only increasing the overall runtime.

4.5 Study Overview

The following section provides an overview of the methodology.

4.5.1 Software Infrastructure. For our software infrastructure, we used Strawman [40], an open-source in situ framework containing three physics proxy applications. Of these three proxy applications, we used Cloverleaf3D [4, 46], a hydrodynamics mini-app on a three-dimensional structured grid. Strawman also includes a rendering infrastructure, which combines node-level rendering using VTK-m [49], configured with Intel's Thread Building Blocks [60], and distributed memory image compositing using IceT [50]. For our study, we integrated PaViz into Strawman and added infrastructure to calculate per node rendering estimates based on the performance model.

4.5.2 Hardware Architecture.

We ran tests on Catalyst, an Intel Ivy Bridge cluster at Lawrence Livermore National Laboratory. Each node contains two hyper-threaded Intel Xeon E5-2695 v2 CPUs containing 12 physical cores. The processor operates at a base frequency of 2.4 GHz, and has a maximum TurboBoost frequency of 3.2 GHz. Each node contains 128 GB of memory. Access to socket-level power capping and monitoring is done through model-specific registers (MSRs), specifically through the msr-safe kernel driver [3]. Using Intel's Running Average Power Limit (RAPL) technology [19], we can power cap each processor in the node between 115W (i.e., thermal design power (TDP)) and 64W, and the processor will adjust the CPU operating frequency to guarantee the specified power cap.

4.5.3 Study Parameters.

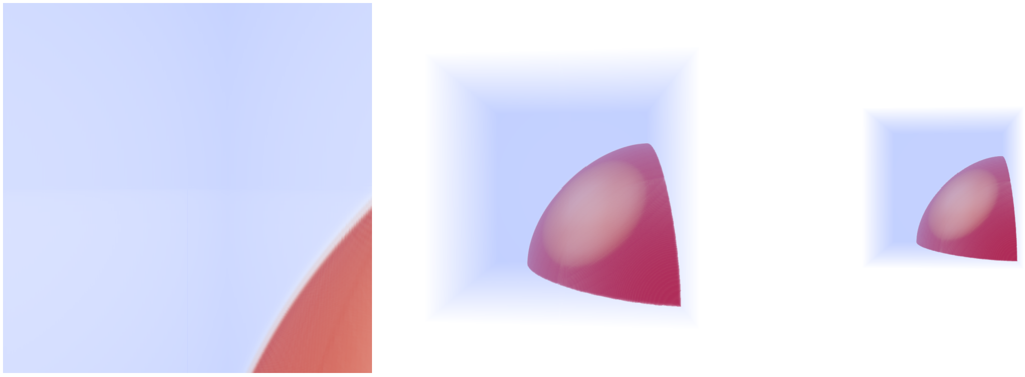
We varied the following parameters as part of this study:

- Rendering Workload (4 options)
- MPI Task Concurrency (2 options)
- Power Scheduling Strategy (5 options)
- Job Power Budget (12 options)

We ran the cross product of the study parameters for a total of 480 tests. We detail each of the parameters listed above in the following subsections.

4.5.3.1 Rendering Workload.

We selected four representative configurations varying in the size of the data set, image resolution, and camera position. These configurations spanned commonly used values for each parameter, and yet each configuration differed in terms of the amount of work per task. The configurations used in this study are enumerated in Table 16. Images of the data



(a) Rendering Workloads A and C (camera positioned inside the data set)

(b) Rendering Workload B (camera positioned near the data set)

(c) Rendering Workload D (camera positioned far from the data set)

Figure 13. Rendered images of the data set from the three camera positions used in this study — inside, near, and far. The renderings show a pressure wave expanding from the corner of the data set where the initial energy was deposited.

from the three camera positions used — inside the data set, near the data set, and far away from the data set — are shown in Figure 18.

4.5.3.2 MPI Task Concurrency. We varied the number of MPI tasks to explore the effects of concurrency on the number of active pixels per task. For the architecture previously described in Section 4.5.2, the number of MPI tasks used were 8 and 64, mapping one task to each node. On this hardware architecture, there are 24 cores per node, so our experiments used a total of 192 and 1,472 cores, for 8 and 64 nodes, respectively. We ran this as a weak scaling study, i.e., the data set size per task was held constant with each level of concurrency.

4.5.3.3 Power Scheduling Strategy. We explore the performance improvements of the five power scheduling strategies defined in Section 4.4.1 to rebalance power based on need. Some strategies are more aggressive in terms of assigning socket power caps, while others are less aggressive, but risk leaving further performance to be reclaimed. In addition to exploring the benefit of

Wkld	Data Set	Image Res	Cam Pos	IFact
A	240^3	2880^2	inside	1.57
B	470^3	1080^2	near	1.16
C	128^3	1920^2	inside	1.58
D	320^3	2048^2	far	1.12

Table 14. Selected rendering workloads for this study. The configurations use 1000 samples per ray and render 100 images per cycle. IFact is a quantitative representation of the work imbalance, derived by taking the maximum estimated render time over the average of all estimates.

rebalancing power based on a performance model, we wanted to explore the benefits of using different power scheduling strategies.

4.5.3.4 Job Power Budget. We vary the power budget by assuming a uniform node power cap (ranging from 230W down to 128W per node, 115W to 64W per processor). In this study, we only consider the power consumption of the socket domain. For example, assume there are eight nodes in the job. The range of job power budgets ranges from 1840W (per node power cap of 230W) down to 1024W (per node power cap of 128W). By enforcing lower node power caps, we arbitrarily limit the job power budget, and can compare the performance of PaViz to a uniform power distribution under a power-limited environment.

4.5.4 Efficacy Metrics. We define two efficacy metrics quantifying the benefits of adaptively rebalancing power based on a performance model. We explain these metrics in more detail in the following subsections.

4.5.4.1 Speedup Over Uniform Power Caps. The speedup metric compares the runtime of PaViz to a uniform power distribution computed by $\frac{job\_power\_budget}{nnodes}$. This scenario is currently implemented in practice. With PaViz, each node may be running at a different power cap, but the aggregate sum of the power caps is less than or equal to the job power budget. A speedup greater than 1 indicates better performance with PaViz in adapting power caps relative to the predicted render time. A speedup less than 1 indicates degraded performance with PaViz, and a speedup equal to 1 indicates no change in performance.

To see the impacts of RAPL's power capping mechanism, the workload must be long enough to overcome the delay between when the new power cap is set and when the processor recognizes (and begins operating at) the new cap. Rendering a single image can be a very quick operation, less than a fraction of a second. However, it is not uncommon to create several images per time step, and on the extreme side, image-based in situ [11], where hundreds of images are rendered per time step. This use case increases overall render time and amortizes out the RAPL delay.

4.5.4.2 Unused Job Power. The second metric compares the job power allocated by PaViz to the original power budget. PaViz rebalances power based on a predicted rendering estimate generated by a performance model, such that the original power budget is not exceeded. The percentage of unused job power is computed by taking the difference between the job power budget and the job power allocated by PaViz divided by the job power budget. We observed the best performance when the entire job power budget is allocated by PaViz, particularly in a power-constrained environment, where some nodes may not be able to execute at TDP.

4.6 Results

We organized our study into four phases. We first look at a base case, which uses eight nodes and a single power scheduling strategy detailed in Section 4.4.1. Subsequent sections evaluate the benefits of PaViz when varying the power scheduling strategy, workload configuration, and concurrency under different power budgets. These factors were previously described in Section 4.5.3. In each phase, we vary the job power budget, and analyzed its impact.

4.6.1 Phase 1: Base Case. In this phase, we compare the speedups of the *Min* scheduling strategy under various job power budgets. The x-axis has been reversed, such that the job power budgets are decreasing, as would be the case with a power-constrained environment. The results are for an imbalanced rendering workload configuration (labeled as "A" and defined in Table 16) and are shown in Figure 14.

The performance degradation is minimal for job power budgets between 1800W and 1600W. This is because the allocated power exceeds the observed (i.e., actual) power consumption of the application. The dotted line, which represents the unused job power, shows the same execution time can be achieved by using up to $12\% - 20\%$ less than the allocated job power budget. If the job power budget is extremely constrained to 1024W, we similarly see no benefit as there is a small amount of job power available to reallocate between the nodes.

In this configuration, PaViz produces up to 10% speedup over the current practice for job power budgets between 1400W and 1100W. For these speedups, PaViz reallocates all of the job power budget, such that there is 0% unused job power. At a job power budget of 1500W, PaViz achieves about 4% speedup in this configuration by using 3% less than the job power budget. If we assume the job power budget is the actual power consumption of the job, then PaViz can also save energy by having a faster runtime than the current practice. For example, with a job power budget of 1360W, PaViz produces a speedup of 10% by using the entire power budget (i.e., 0% unused power). This produces an energy savings of about 9% as compared to the current practice.

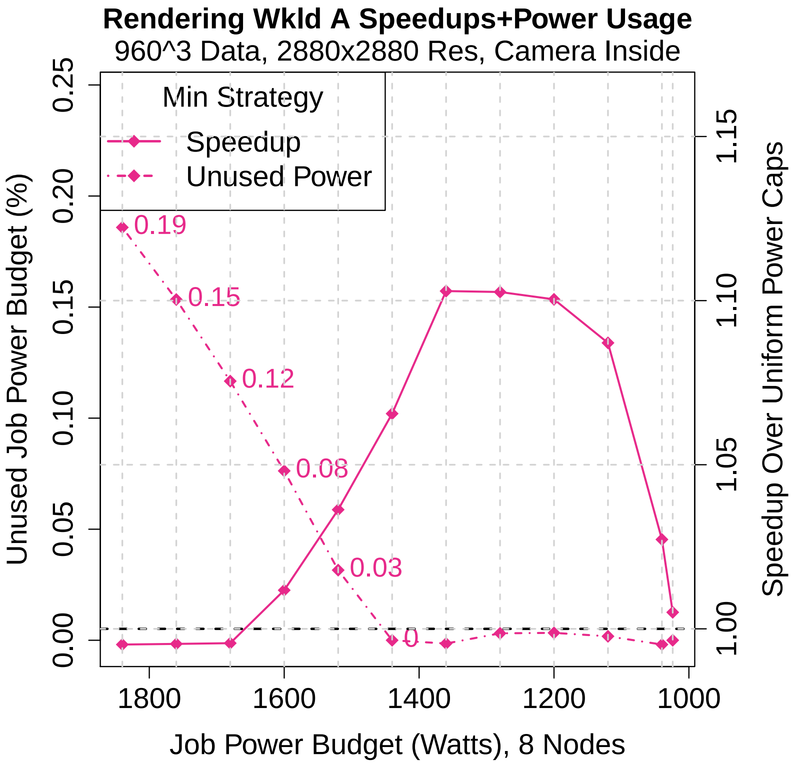


Figure 14. Speedups and allocated power for Rendering Workload A using the Min power scheduling strategy. The solid line shows the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup is 1, indicating no change in performance. The dotted line shows the percentage of unused job power budget that resulted in a particular speedup (left y-axis). A percentage of 0% means that the entire job power budget was reallocated across the nodes.

4.6.2 Phase 2: Vary Scheduling Strategy.

In this phase, we compare the speedups and unused job power of the five power scheduling strategies (see Figure 15). The *Min* strategy performed the best of all strategies in this configuration, since it assigns the highest power limit to those nodes with high estimated render times and vice versa.

On the other hand, the *Max* strategy performed the worst of all strategies. This strategy assigns high power caps to those nodes with low predicted render times, while assigning low power caps to those with high predicted render times. With this strategy, performance degrades significantly since the node with the most work to do (e.g., the bottleneck node) will execute at a lower power cap.

The *Normalized* strategy has a similar behavior to the *Min* strategy. This is because power caps are scaled directly by the estimated render time, which will assign high power caps to high render times and low power caps to low render times. For this configuration, the *Normalized* strategy does not achieve as high a speedup as the *Min* strategy because it is unaware of the fastest render time, and will assign a less aggressive power cap.

The *Mean* strategy performs as well as the current practice for this configuration with the camera positioned inside the data set. With an imbalanced workload, some nodes will be higher than the mean and others will be lower than the mean, and will average out to the same performance as running all nodes at the same power cap.

The *Median* strategy degrades performance slightly. Depending on the distribution of estimated render times, the median may cause the non-ideal assignment of power caps to predicted render times.

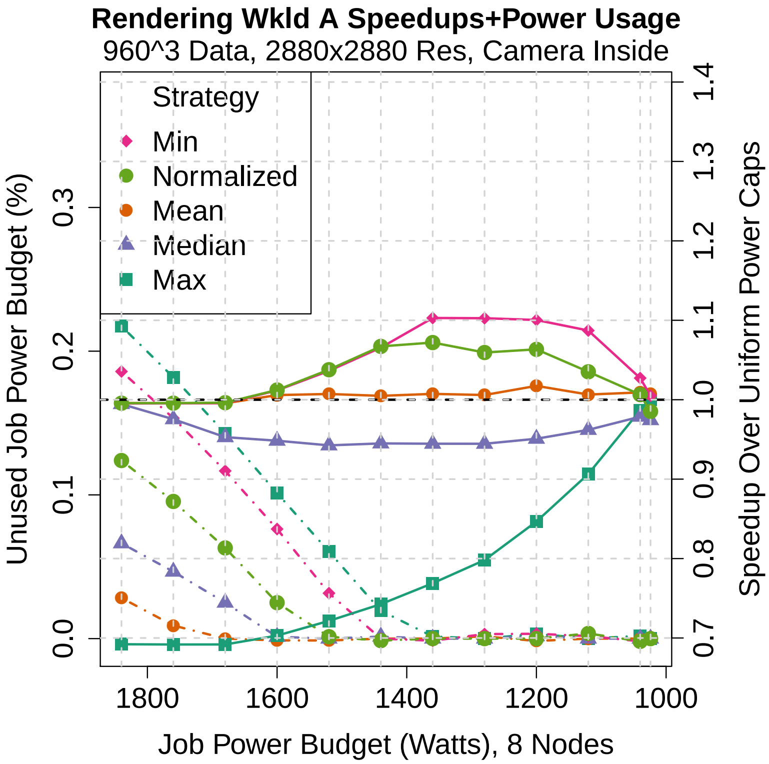


Figure 15. Comparing speedups and unused job power budget for Rendering Workload A across all five power scheduling strategies.

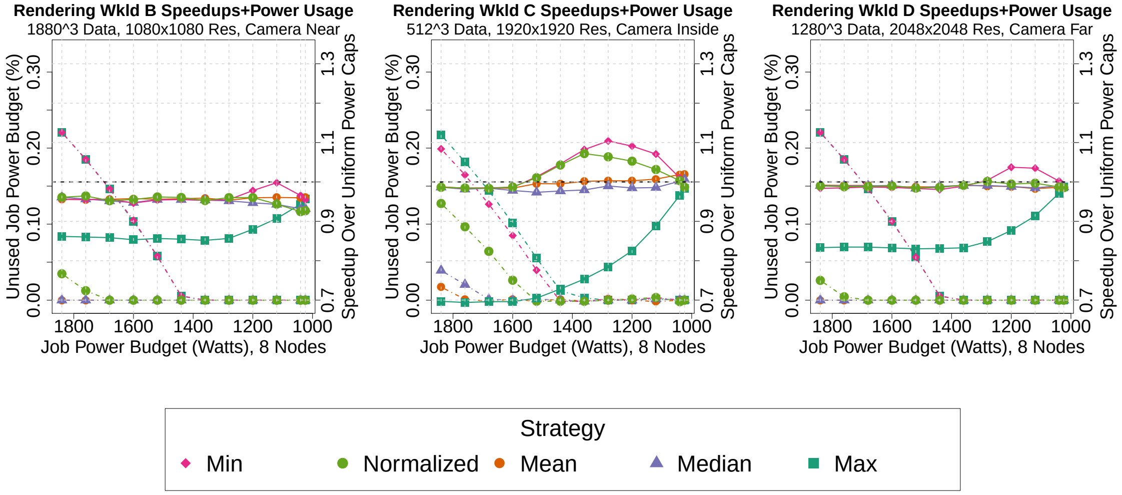


Figure 16. Comparing speedups and unused job power for Rendering Workloads B, C, and D at 8 node concurrency using all five power scheduling strategies. The solid lines show the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup is 1, indicating no change in performance. The dotted lines show the percentage of unused job power that resulted in a particular speedup (left y-axis).

4.6.3 Phase 3: Vary Workload Configuration.

We vary the workload configuration to demonstrate how rendering parameters may impact the potential for performance improvements. With the camera positioned inside the data set (Rendering Workloads A and C), there is greater work imbalance (i.e., wider distribution of predicted render times) between the nodes because some nodes will have no geometry in the field of view of their cameras, and thus will perform no rendering. Moving the camera position far away from the data set, such as in Rendering Workloads B and D, creates a more even distribution of predicted render times, and this balance limits the ability of PaViz to achieve significant speedups.

Figure 16 shows the speedups and unused job power for the remaining workload configurations — B, C, and D. Rendering Workload C has a maximum speedup of 10% that is comparable to Rendering Workload A, which was previously shown in Figure 15. This is because there is significant imbalance when the camera is positioned inside the data set, providing more benefit from adaptively rebalancing power. However, we note that Rendering Workload A provides speedups with the *Min* and *Normalized* strategies over more job power budgets than Rendering Workload C. The range of raw render estimates is far greater in Rendering Workload A (0.3 sec to 1.4 sec) than Rendering Workload C (0.15 sec to 0.64 sec) due to the data set size per node. The increased distance between the minimum and maximum predicted runtime gives Rendering Workload A more opportunity for benefits with PaViz.

We achieve little to no speedup on Rendering Workloads B and D because the render estimates are balanced when the camera is positioned further away from the data set, which matched our initial intuition. For these configurations, the render estimates ranged from 0.10 sec to 0.14 sec for workload B, and 0.12 sec to 0.16 sec for workload D, which did not provide much room for adapting power (in several cases, all nodes were assigned the same uniform power cap). The *Min* strategy results in a 4% speedup on Rendering Workload D, but we attribute this to performance variability when the processor is under a power cap.

4.6.4 Phase 4: Vary Concurrency. In this phase, we increase the node concurrency from 8 nodes to 64 nodes to understand the potential benefits at scale. The initial intuition was that a higher concurrency would lead to better performance since there would be a larger work imbalance per node and a larger job power budget that can be reallocated between nodes. Figure 17 shows the speedups and unused job power for all rendering configurations enumerated in

Table 16 using 64 nodes. We weak scale the data size to maintain the same work per node.

For these configurations, PaViz achieves up to 33% speedup over uniform distribution of power. At 64 nodes, we see the render predictions change in two ways. First, the range of predictions between the minimum and maximum render value is much smaller. Secondly, a larger percentage of nodes have very little, or even no, geometry to render. In the imbalanced configurations A and C, the scheduling strategies in PaViz assign these nodes low power caps, enabling nodes with lots of work to operate at a high power cap. We suspect that imbalanced workloads at even higher concurrencies will achieve even greater speedups. In the balanced configurations B and D, the performance estimates were extremely fast (less than 0.08 sec), the range of estimates was much closer to one another that they were with eight nodes (ranging from 0.06 sec to 0.07 sec), and the scheduling policies assigned uniform power caps across all nodes.

4.7 Conclusion

In this chapter, we explored the viability of using prediction to improve the performance of visualization workloads. Specifically, we considered parallel rendering in the context of overprovisioned supercomputers. Like other HPC research on overprovisioning, we set per-node power caps in an effort to allocate power to the nodes that needed it most. However, since visualization workloads are highly variable, they required a new approach for deciding how to assign power caps. This new approach leverages prediction of execution times. We incorporated an existing performance model, and considered five strategies that make use of per-node workload estimates. The resulting study demonstrated that our approach is beneficial, with results as much as 33% faster than a uniform distribution strategy

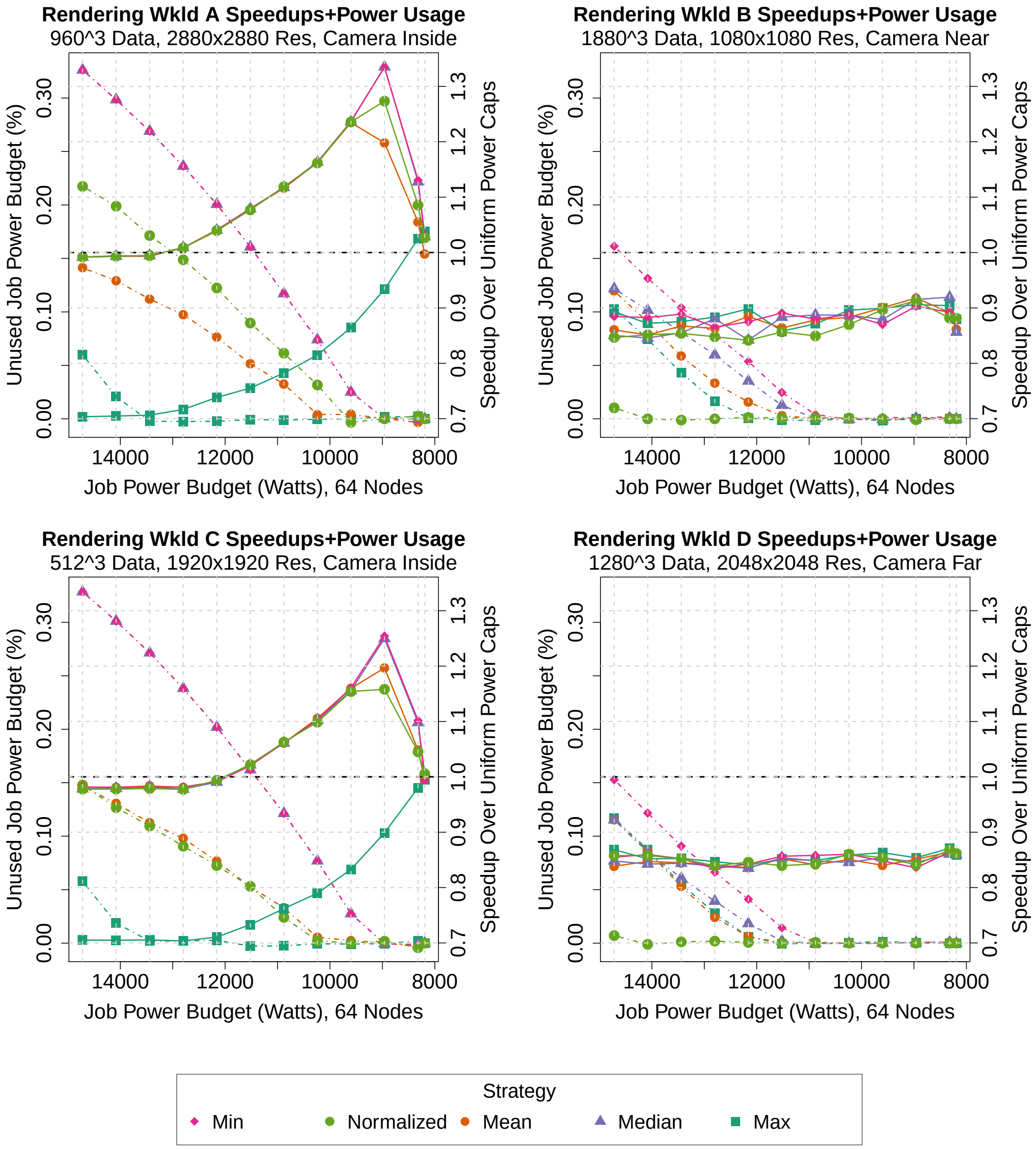


Figure 17. Comparing speedups and unused job power for all rendering workloads at 64 node concurrency using the five power scheduling strategies. The solid lines show the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup is 1, indicating no change in performance. The dotted lines show the percentage of unused job power that resulted in a particular speedup (left y-axis).

while using the same power. In terms of future work, we would like to explore how PaViz can be adapted to additional predictive models for visualization algorithms.

CHAPTER V

EVALUATING TECHNIQUES FOR SCHEDULING POWER

Most of the text in this chapter comes from [37], which is a collaboration between Matthew Larsen (LLNL), Hank Childs (UO), Barry Rountree (LLNL), and myself. This journal paper is currently in progress.

Hank Childs and Matthew Larsen provided feedback on the overall organization of the manuscript. Barry Rountree contributed to the related work on power research in HPC. I developed the predictive runtime system. The comparator runtime system is developed by Intel and uses adaptation of progress to assign power allocations across nodes. I designed and executed the study, and was the primary contributor to the writing of the overall paper.

This chapter compares the methodologies of using prediction and adaptation to make decisions on allocating power as a resource to nodes within a job. Some runtime systems use adaptation, and extrapolate the execution behavior of the first few iterations of a loop to the subsequent iterations. Our runtime system introduced in Chapter IV uses prediction, which may be better suited for visualization routines which are data dependent and can be highly irregular. Findings from this study inform which methodology is best suited for managing power resources for visualization applications.

5.1 Introduction

One of the key challenges in achieving an exascale system is power usage. At the beginning of this decade, scaling current technologies to higher concurrency would lead to excessive power consumption costs. High power costs are expensive and unsustainable, so the entire HPC environment, is being re-evaluated with power efficiency in mind.

Due to the imbalance between computational and I/O performance of future systems, the visualization community is transitioning from post hoc processing to in situ processing. With the in situ paradigm, visualization and analysis occur while the simulation is running, making it unnecessary to store the simulations state to disk and read it afterwards. Running the simulation with the visualization means that the overall time-to-solution will increase by some factor. The magnitude of this factor is dependent on the desired quantity of visualization. The percentage of time spent doing visualization is highly variable, sometimes taking as much as 10% or 20% of the overall turn-around time. For this study, the visualization component contributes up to 14% of the overall time. We assume an in situ workflow for this work and explore two strategies for optimizing performance of the visualization pipeline, thus reducing the overall execution time.

Both scientific simulations and visualization routines need to adapt to a power-limited environment, meaning compute nodes will have their power usage capped. Based on current practices, a lower power cap would uniformly be applied across all compute nodes in the system. However, a uniform power cap is a sub-optimal strategy since the runtime behaviors of distributed applications can be highly variable across nodes. The nodes assigned the largest amount of work become a bottleneck and determine the overall performance of the application. On the other hand, nodes that are assigned the smallest amount of work finish quickly and wait until the other nodes have finished execution. A more intelligent strategy is to assign power to where it will result in the most benefit. Ideally, we can assign the power such that all nodes finish executing at the same time despite varying workloads.

The main contribution of this paper is an evaluation that compares adaptive and predictive power management schemes for visualization workloads. As part of this research, we used two existing power-aware runtime systems, GEOPM and PaViz, on a ray tracing workload. GEOPM leveraged an online method of adapting to current execution behaviors, while PaViz incorporated runtime predictions based on an accurate performance model. In terms of findings, we found that, in limited power budget environments, using a power-aware runtime system with performance model predictions led to better speedups than an adaptive model.

The rest of this chapter is organized as follows. The related work is detailed in Section 5.2. An overview of the GEOPM and PaViz runtime systems and their respective adaptive and predictive power scheduling strategies are discussed in Section 5.3. Section 5.4 identifies the study parameters. We evaluate the effectiveness of adaptive and predictive scheduling strategies in Section 5.5. Lastly, Section 5.6 summarizes our findings and presents some ideas for future research.

5.2 Background and Related Work

5.2.1 Power. Energy use has been a long-term challenge in HPC. Early solutions used dynamic frequency and voltage scaling (DVFS) to make tradeoff decisions between performance and energy savings at varying granularities [25, 29, 62]. The common goal of these approaches was to minimize energy usage by incurring a small performance degradation.

Vendor technologies for applying power caps have enabled more recent research focusing on performance under a power bound. Some of these technologies include Intel's Running Average Power Limit [19], AMD's Application Power Management [20], IBM EnergyScale [30], NVIDIA's NVML [53]. Conductor [47] used initial iterations to determine an ideal schedule of per-node power caps, thread concurrency, and per-core operating frequency. GEOPM [2, 22] is a production-grade runtime framework for optimizing performance under resource constraints. GEOPM supports manual application markup as well as automated phase detection to dynamically reallocate power. Its architecture supports multiple plugins, but currently it does not support any particular policy targeted at in situ visualization.

5.2.2 Ray Tracing. Ray tracing is a common method for rendering images. With ray tracing, rays are traced from the viewpoint through pixels, and intersect with the geometry to be rendered. The process of tracing rays is embarrassingly parallel, however, scaling the algorithm to render millions of pixels is challenging. Previous efforts have implemented a parallel ray tracer [15, 56]. For our study, we chose ray tracing because of its widespread use and because of its existing performance model [41].

5.2.3 Visualization and Power. Due to the I/O bandwidth limitations at exascale, visualization is moving away from a traditional post-processing method to in situ. In the post-processing method, the simulation writes out data to disk at regular time steps. Once the simulation has completed, the data is read back from disk for post-processing analysis and visualization using tools, such as VisIt [17] or ParaView [10]. As simulations increase in complexity, the amount of data they can write out increases exponentially, making it unsustainable to write out data with high temporal frequency. The critical challenge is saving enough data from the simulation without impacting fidelity or losing notable areas of interest.

In the in situ model, visualization and analysis occur alongside the simulation to mitigate the impacts of reduced I/O bandwidth. The data from the simulation is visualized and the resulting images are written to disk, vastly reducing the total amount of data written out. Since power is a critical challenge to reaching the next generation of computing, research efforts have been dedicated to understanding the power profiles of this new workflow, particularly with respect to how data is moved through the storage hierarchy [7, 61]. These works compared the power profiles of each pipeline, concluding that in situ drastically reduces energy usage by reducing the total runtime to complete the simulation and analysis.

Labasan et al. [35] provided an initial exploration of the various factors that may impact power and performance tradeoffs for an isosurfacing algorithm implemented in two frameworks. This work studied the performance impacts of various parameters as the CPU operating frequency was gradually reduced. Similarly, work by Gamell et al. [27] also looked at the power-performance tradeoffs of various parameters at scale.

5.3 Power-Aware Runtime Systems

The following subsections detail the two different power scheduling strategies being evaluated in this paper. The strategies are implemented in two runtime systems, known as GEOPM and PaViz, which we explain here. Both share a common goal to improve performance in a power-limited environment, but do so in their own ways.

In an overprovisioned system, not all nodes can run at full power simultaneously. In order to run all nodes at the same time, a uniform power cap is applied at a reduced limit. In a load imbalanced workload, like visualization, this strategy is a suboptimal choice, since the performance is ultimately determined by the last processor to finish its work. A more performant strategy is to assign power to where it is needed (e.g., the critical path), speeding up the processors that are behind and slowing down the processors that are further ahead.

The other key challenge for these runtime systems to be successful is to redistribute the power, such that the systemwide power limit is not exceeded. Exceeding the given power limit will cause electrical issues at the system level, which may cause failures or breakage.

5.3.1 GEOPM: Adaptive Runtime System. GEOPM [2, 22] is an open-source framework for power and energy research on future HPC systems. GEOPM is a collaborative project, started and supported by Intel. It is designed to target Intel platforms, but can be extended to support other hardware platforms that provide power management capabilities. It leverages a tree design to be portable at high concurrencies. GEOPM is also designed to support the different power and energy management requirements across computing facilities through an agent plugin architecture.

The GEOPM runtime system analyzes execution behaviors within the application, then optimizes performance by coordinating decisions to hardware or software control knobs across compute nodes. Some control knobs are per-core clock frequencies and processor-level power limits. By tuning control knobs during an application's execution, GEOPM may improve performance despite workload imbalances and manufacturing variations across nodes.

GEOPM's runtime system uses a balanced tree to provide hierarchical feedback about the application's performance. There is a controller running on a single thread per node to handle different roles and tasks. One of the controllers is the root node, and will make the final decision based on feedback from the leaf nodes. This hierarchical design allows GEOPM to be performant at high levels of concurrency. For this study, we use the Power Balancer plugin, which is discussed in Section 5.4.

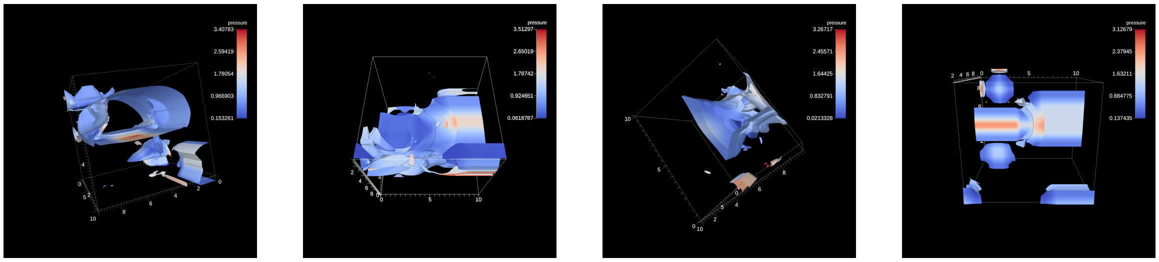
5.3.2 PaViz: Predictive Runtime System.

PaViz [38] uses prediction to dynamically allocate power across nodes in a job. For our approach, we incorporate an existing rendering performance model [41] into our runtime system. If the performance model predicts a long rendering time due to a high volume of work, then we allocate more power to that node. Alternatively, we allocate less power if the performance model predicts a short rendering time due to less work being assigned.

Larsen et. al [41] created and validated performance models for scientific visualization workloads. The performance model makes a prediction based on camera position and data set size. These performance models have easily calculable inputs. The performance models are semi-empirical, meaning that they are both based on algorithmic characteristics and observed execution behaviors on specific hardware architectures. In order to leverage these models, the models must be fitted to a particular hardware architecture.

When it comes to deciding how much power should be allocated to each node, we implemented multiple power scheduling strategies and previously evaluated them [38]. The best scheduling strategy is most aggressive in assigning power caps to the longest and shortest running nodes.

Specifically for rendering, the amount of work per rank is highly variable from one timestep to another depending many factors, such as the visualization operation and resulting output. Being able to predict the amount of work before execution is ideal in this case since you cannot assume behavior will be constant between timesteps.



(a) Workload A (b) Workload B (c) Workload C (d) Workload D

Figure 18. Ray traced images of CloverLeaf at the 200th simulation cycle. The figure shows a contour of the pressure at various values after it has expanded from the initial position where the energy was deposited.

5.4 Experimental Overview

The following subsections detail the experimental setup and methodology.

5.4.1 Software Infrastructure. We used VTK-m and Ascent to provide visualization and analysis capabilities. VTK-m [49] is a library of scientific visualization algorithms optimized for shared-memory parallelism. The algorithms use data parallel primitives to provide portable performance across many different hardware architectures. VTK-m is an extension of the Visualization ToolKit [64], a library of visualization algorithms. VTK is the basis for VisIt [17] and ParaView [10].

Ascent is a flyweight in situ visualization and analysis runtime system for scientific simulations. It aims for portable performance on future many-core CPU and GPU architectures [39]. It is designed to support other in situ visualization tools, such as VisIt's LibSim [68] and ParaView's Catalyst [13]. Ascent depends on VTK-m for inter-node parallelism and OpenMP for intra-node parallelism. Ascent's framework includes three proxy simulations — Kripke [34], Lulesh [33], and CloverLeaf [4, 46]. For the scientific simulation in this study, we used CloverLeaf, a hydrodynamics proxy application on a three-dimensional structured grid.

5.4.2 Hardware Architecture.

We conduct our experiments on the Quartz supercomputer at Lawrence Livermore National Laboratory, which is a 2,634 node Broadwell system (Intel E5-2695). Each node contains two hyperthreaded processors and 18 physical cores per processor. The base clock frequency is 2.10 GHz and the processor is rated at 120W TDP.

The msr-safe [3] kernel module enables power monitoring and control from user space. We enforce a processor-level power limit using Intel's Running Average Power Limit (RAPL) interfaces [19]. Under a more severe power limit, the processor operates at a lower CPU frequency to guarantee that the average power usage does not exceed the specified limit. For this particular architecture, we can enforce a processor-level power cap ranging from 120W down to 40W.

5.4.3 Study Parameters.

This study was designed to evaluate two power-aware runtime systems (PaViz and GEOPM) under a variety of rendering workloads. We varied the following parameters in order to study a representative set of configurations:

- Job Power Budget (9 options)
- Rendering Workload (4 options)
- MPI Tasks (5 options)
- Power Scheduling Strategy (2 options)

We ran the cross product of the aforementioned parameters totaling 104 tests configurations. Each of the parameters are discussed in the following subsections.

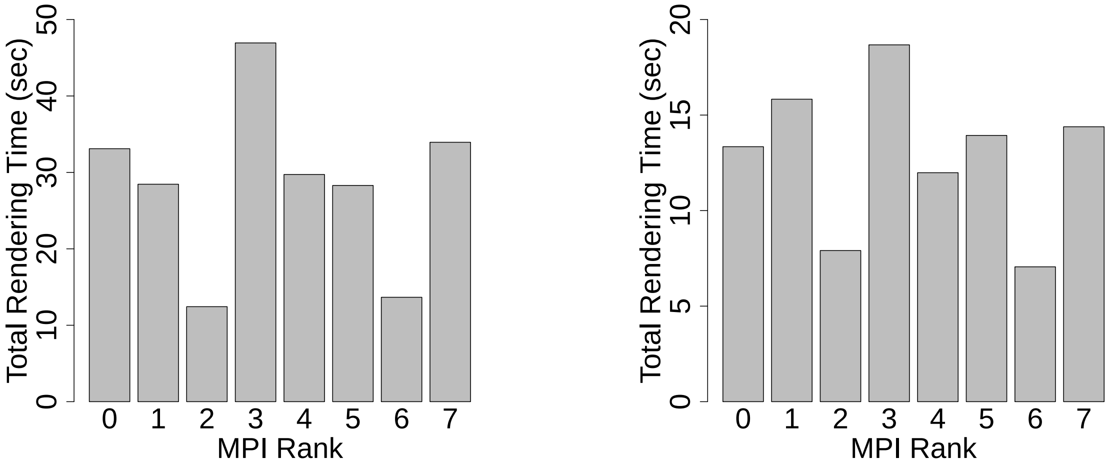
5.4.3.1 Job Power Budget.

Both GEOPM and PaViz assume a specified job power budget. Individual compute nodes may be allocated varying amounts of power, but at no given moment can the aggregate sum of the allocated power across all nodes exceed the specified job power budget. To simplify this study, both runtime systems are only concerned with the power usage of the processors within the node. Table 15 shows the power limits for each runtime system.

GEOPM (Per-Node)	**PaViz** (Per-Processor)	**% TDP**
260W	120W	100%
200W	100W	83.3%
180W	90W	75%
160W	80W	66.7%
140W	70W	58.3%
136W	68W	56.7%
120W	60W	50%
100W	50W	41.7%
80W	40W	33.3%

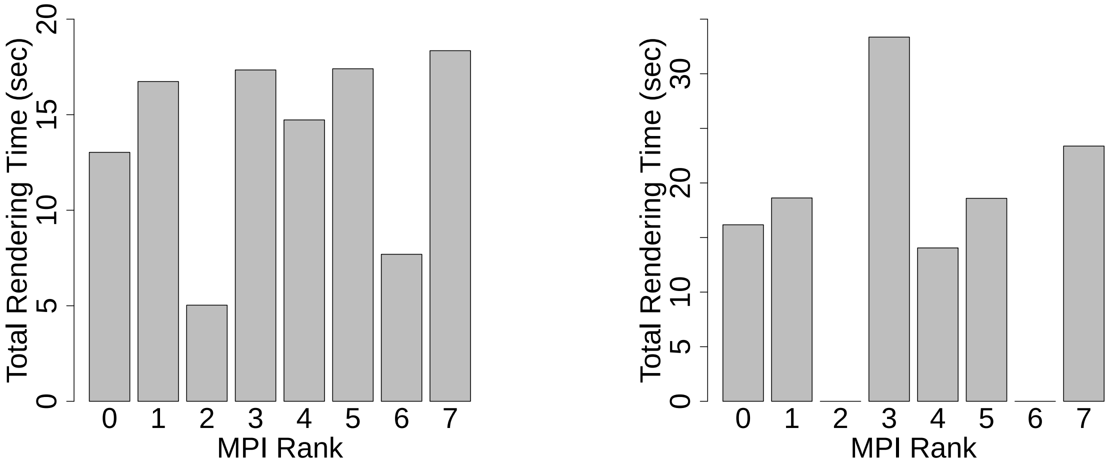
Table 15. Enumerating the user-specified power budgets used in each runtime system. The GEOPM power budget is node-level (split evenly across all processors within the node). The PaViz power budget is processor-level (scale by the number of processors within the node to derive node power budget). Scaling the power budgets by the number of nodes will determine the total job power budget.

5.4.3.2 Rendering Workload. We selected four workloads that varied in the size of the data set, the isovalues used for the contour, the number of images generated per visualization cycle, and the image resolution. The rendering workloads and their parameters are listed in Table 16 and an image of the resulting contour is shown in Figure 18. These configurations span commonly used values for each parameter, and each workload exhibits a different amount of work imbalance. Figure 19 shows the imbalance in the total rendering times across all visualization cycles per rank.



(a) Rendering Workload A

(b) Rendering Workload B



(c) Rendering Workload C

(d) Rendering Workload D

Figure 19. For each rank, we aggregate the time spent rendering across all visualization cycles. If the workload was perfectly balanced, each rank would have the same execution time. However, rendering is a highly imbalanced workload, so there are significant differences in execution time across ranks. We aim to address the imbalance by shifting power away from the ranks with low execution times, and shifting power to the ranks with high execution times.

The amount of time spent doing rendering can vary greatly depending on different user-specified parameters, such as the camera position and the image resolution. Typically, rendering is a very quick operation and is a small fraction of the total time doing visualization operations. Reducing the amount of data saved to disk is a common strategy for mitigating the I/O challenges of future supercomputers. One method of reducing the amount of data is rendering hundreds to thousands of images per timestep of the resulting analysis (i.e., cinema) and save them to disk, which is significantly smaller than the original data set.

Our rendering infrastructure used cinema-based in situ [11, 54], where an interactive database is generated by taking many pictures from various camera positions around the data set. In this paradigm, the cost of rendering can become a significantly large percentage of the overall visualization and analysis pipeline. Selecting the number of images to be generated during each visualization cycle is another user-specified parameter that can greatly impact overall execution time. Thus, understanding how to improve the performance of this operation is critical.

Wkld	**Data Set**	**Isoval**	**Res**	**Phi**	**Theta**	**IFact**
A	240^3	0.4	2880^2	17	10	1.32
B	190^3	0.6	1080^2	18	9	1.26
C	128^3	0.9	1920^2	17	10	1.18
D	320^3	1, 3.4, 5.2	2048^2	17	10	1.56

Table 16. Selected rendering workloads for this study. The data set size is per rank. It is multiplied by the cube root of the number of nodes to get the total data set size. The number of images rendered per cycle is determined by $Phi \times Theta$. The simulation ran for a total of 300 cycles. Visualization occurred every 50 cycles. IFact is a quantitative representation of the work imbalance, derived by taking the maximum aggregate sum of the predicted render time over the median of all estimates.

5.4.3.3 MPI Tasks. We varied the number of MPI tasks to study the scaling behaviors of adaptive and predictive strategies at higher concurrencies. We used a single MPI task per node (OpenMP threaded), and selected the number of tasks such that the data set size was constant per task. The number of nodes used were 8, 125, 216, 343, and 512.

5.4.3.4 GEOPM Power Balancer Policy. We used the Power Balancer policy included in GEOPM for dynamically varying the power limit of individual nodes to optimize performance. The Power Balancer monitors the performance of the application and regularly redistributes power between the nodes. Similar to PaViz, more power is given to the nodes on the critical path, enabling them to run at a higher power limit, while power is taken away from nodes not on the critical path. With this mode, power allocations to each node are non-uniform, inversely proportional to the load imbalance. The Power Balancer measures the loop execution time on each node and compares execution times across nodes to identify critical path nodes and how much correction is needed.

The power redistribution occurs in a hierarchical tree. The average power cap is passed to all compute nodes. Each node reports their performance under this power cap up the tree. The root node aggregates the performance per-node, and passes back the worst performance. Each node reduces its power limit until its performance matches that of the worst node, passing back its extra unneeded power. This pool of unused power is redistributed across the nodes to improve overall performance.

5.4.3.5 PaViz Power Scheduling Strategy. We used the *Min* power scheduling strategy, which resulted in the best performance of all the strategies explored in [38]. The *Min* strategy uses the difference from the fastest predicted render time to determine the power allocation. Using the per-node predicted rendering execution times, the node power cap is computed as follows:

$$pow_{node\_min} + \frac{|ren_{min} - ren_i|}{\sum_{i=0}^{n-1} |ren_{min} - ren_i|} * pow_{avail},$$

where pow_{node_min} is the hardware-specified minimum node power needed to execute the job, ren_{min} is the global minimum predicted render time among all n MPI tasks, ren_i is the predicted render execution time for rank i, and pow_{avail} is the available power to allocate to the job. Nodes that are furthest away from the minimum (i.e., highest render time, most work to be done) are allocated a large amount of power, resulting in the highest speedups in a balanced and imbalanced workload configuration, since the rendering task is only as fast as the slowest processor.

5.5 Results

PaViz Power Decisions

Cycle Rank	0	1	2	3	4	5
0	70W	70W	69W	63W	60W	58W
1	58W	58W	59W	58W	59W	61W
2	52W	50W	48W	40W	40W	40W
3	76W	77W	75W	70W	66W	65W
4	58W	58W	62W	66W	67W	65W
5	61W	62W	62W	58W	60W	65W
6	40W	40W	40W	58W	61W	60W
7	65W	65W	66W	68W	67W	66W

Table 17. Comparing GEOPM and PaViz specified power limits for each node across all visualization cycles for Rendering Workload A. The job power limit assumes a processor-level power cap of 60W.

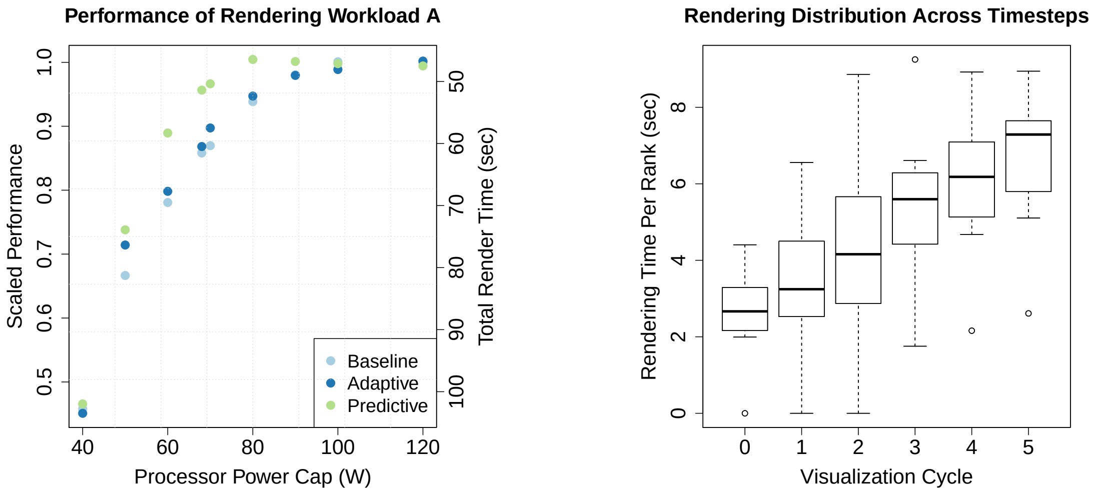
We organize the results into several phases. The first phase studies a base case. Subsequent phases varied additional study parameters. In each phase, we vary the processor power cap and analyze its impacts.

To evaluate the power scheduling strategies, we normalize the performance of our adaptive and predictive strategies to the performance at a uniform power distribution of 120W per processor, which is the maximum power draw for this processor architecture. The uniform power allocation strategy is currently used in practice. With the GEOPM and PaViz runtime systems, each node may be assigned a different power cap, but the sum total of the power caps is less than or equal to the job power budget.

We focus on the scaled performance at the power caps in the region of interest. Power caps towards the low-end of the range are limited power scenarios and power caps towards the high-end are the unconstrained power consumption of the application.

5.5.1 Base Case. In this phase, we focus on Rendering Workload A running on 8 nodes. We sweep over all processor power caps, ranging from 120W down to 40W, and compare the performance of the adaptive and predictive strategies. The left figure in Figure 20 shows the scaled performance for the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz. The performance of the scheduling strategies is compared to the Baseline performance at a uniform power cap of 120W per processor. The right figure shows the imbalanced rendering execution times per visualization cycle across all ranks. As the simulation iterates across time steps, the rendering time increases and the work per rank varies.

At higher power caps, the Baseline, adaptive strategy, and predictive strategy have the same performance because there is unlimited power, and not much room for improvement by shifting power. Similarly at a severe power cap of 40W, all three configurations have the same performance because there is a



(a) Scaled Performance

(b) Workload Distribution

Figure 20. The left figure shows the scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the Baseline for Rendering Workload A. We normalize the performance to that of the Baseline, which applies a uniform power cap of 120W, which is 3X higher than the lowest power cap of 40W. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The right figure shows the distribution in rendering execution times (i.e., work) per rank at each visualization cycle. The input data at each cycle impacts the amount of time spent rendering by each rank, as well as the execution time at each cycle.

minimum power cap and CPU frequency for safe and reliable operation of the processor.

At power caps ranging between 80W and 50W, we start to see the differences in using adaptation versus prediction on the highly irregular visualization workload. This is the range where the application is consuming all the available power and benefits from shifting power intelligently.

At a processor power cap of 60W, we compare the power decisions made by the adaptive and predictive strategy across all visualization cycles in Table 17. For the first few cycles, GEOPM keeps all ranks at this power cap, since it uses the first few iterations to identify the most and least effective nodes. In the first cycle, the predictive strategy identifies rank 6 with having no work to do, and reduces the power cap to the minimum for reliable operation. At later visualization cycles, the adaptive strategy has identified the least efficient ranks, and shifts power such that these nodes receive more power.

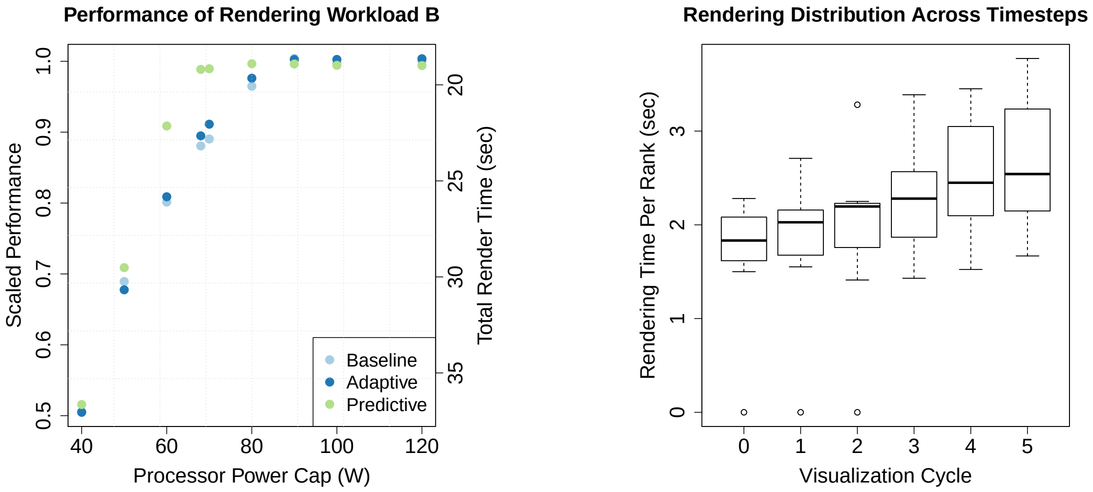
The distribution of render times in Figure 19 shows rank 3 being assigned the most work, so it is expected that both strategies will assign it the highest power cap. In doing so, the adaptive and predictive strategies reduce overall execution time and perform better than the baseline.

5.5.2 Vary Workload Configuration. In this set of tests, we vary across the remaining three rendering workload configurations outlined in Table 16, using the same number of nodes as in Section 5.5.1. The goal of this study is to demonstrate how rendering parameters may impact the potential for performance improvements. If the visualization operation results in a high variance in the number of active pixels to be rendered by each rank, there is more room to exploit the imbalance by shifting power. On the other hand, if there is an evenly distributed number of active pixels to be rendered by each rank, this may inhibit the benefits of shifting power to improve performance.

Figure 21, Figure 22, and Figure 23 compare the results of the adaptive strategy and predictive strategy to the Baseline as well as the distribution of predicted execution times over all visualization cycles. These figures can be compared with Figure 20, which did the same analysis for Rendering Workload A. Compared to the previous Rendering Configuration A, the distribution of execution times are more evenly balanced.

For these workload configurations, using prediction sees more benefit than using adaptation in a similar range of power caps as before. The prediction model identifies which ranks will have no work to do before the visualization occurs. Reducing the power of those ranks to the minimum enables more power to be given to the ranks with lots of work to. This allows them to run faster, complete their work in less time, and reduce overall performance. An adaptive strategy will also identify which ranks have less work to do, but spends the first set of iterations performing the necessary analysis.

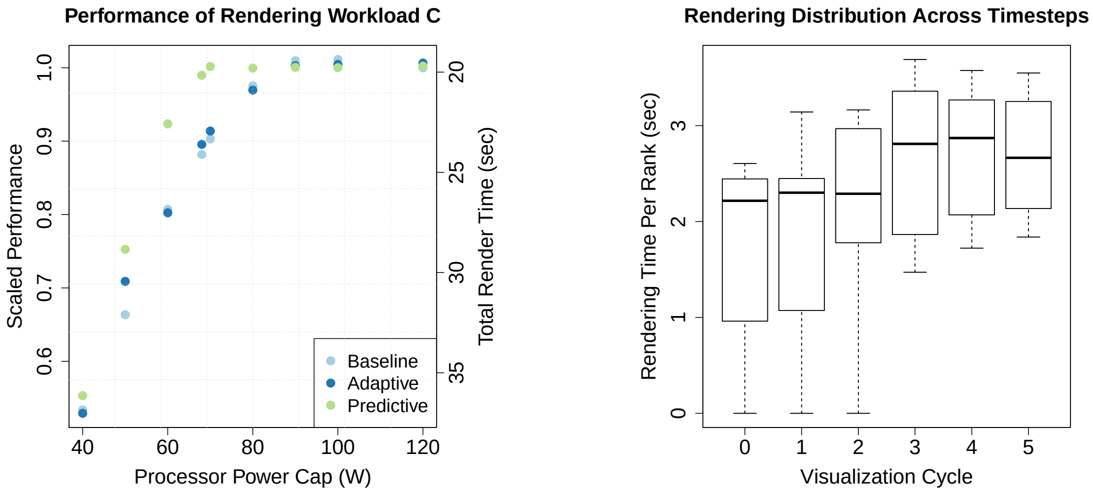
5.5.3 Vary Concurrency. In this phase, we vary the node concurrency to compare the impacts of using adaptation and prediction at scale. The intuition is that at higher concurrency, there is a bigger work imbalance per node as well as a larger job level power budget that can be reallocated between nodes. This phase focuses on Rendering Workload C, which is defined in Table 16. We sweep over processor power caps ranging between 50W and 70W, since previous phases identified this range as the region of interest. We weak scale the data size to maintain the same work per node. Figure 24 shows the scaled performance for



(a) Scaled Performance

(b) Workload Distribution

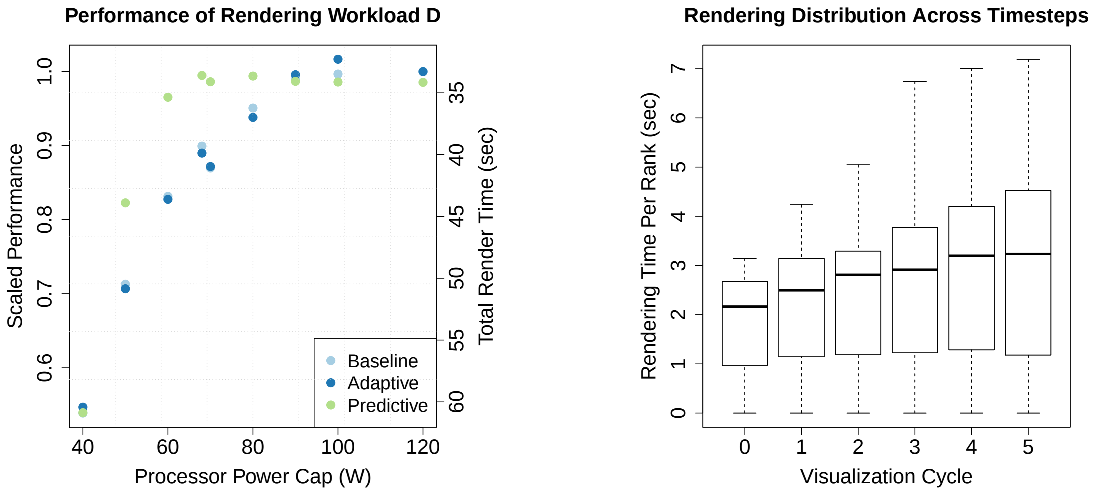
Figure 21. The left figure shows the scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the Baseline for Rendering Workload B. We normalize the performance to that of the Baseline, which applies a uniform power cap of 120W, which is 3X higher than the lowest power cap of 40W. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The right figure shows the distribution in rendering execution times (i.e., work) per rank at each visualization cycle. The input data at each cycle impacts the amount of time spent rendering by each rank, as well as the execution time at each cycle.



(a) Scaled Performance

(b) Workload Distribution

Figure 22. The left figure shows the scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the Baseline for Rendering Workload C. We normalize the performance to that of the Baseline, which applies a uniform power cap of 120W, which is 3X higher than the lowest power cap of 40W. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The right figure shows the distribution in rendering execution times (i.e., work) per rank at each visualization cycle. The input data at each cycle impacts the amount of time spent rendering by each rank, as well as the execution time at each cycle.



(a) Scaled Performance

(b) Workload Distribution

Figure 23. The left figure shows the scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the Baseline for Rendering Workload D. We normalize the performance to that of the Baseline, which applies a uniform power cap of 120W, which is 3X higher than the lowest power cap of 40W. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The right figure shows the distribution in rendering execution times (i.e., work) per rank at each visualization cycle. The input data at each cycle impacts the amount of time spent rendering by each rank, as well as the execution time at each cycle.

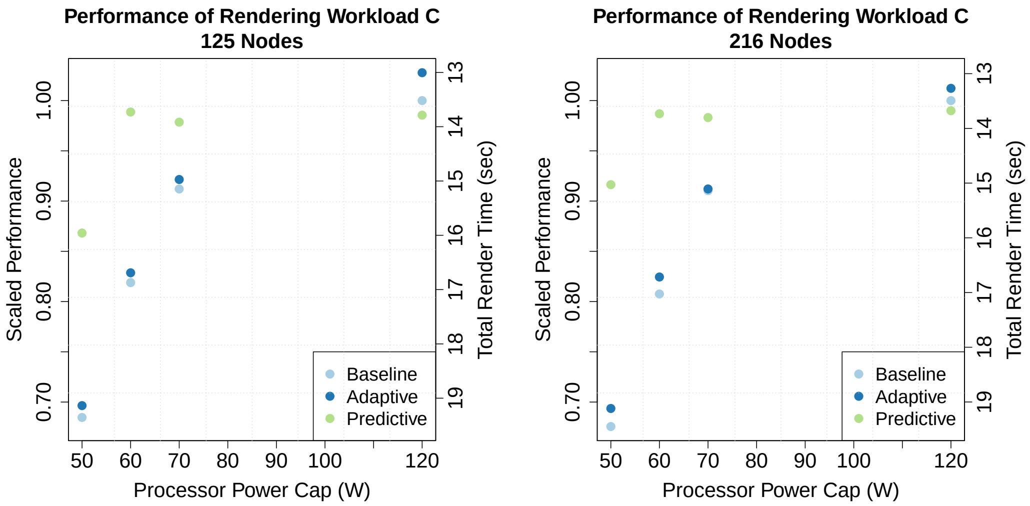
the adaptive strategy in GEOPM and the predictive strategy in PaViz at different levels of concurrency.

For this rendering configuration, using a predictive strategy results in 27% improvement over an adaptive strategy. As the concurrency increases, an increasing percentage of the nodes have very little, or even no, geometry to render. Figure 25 shows the difference in scaled performance between PaViz and GEOPM. The smallest differences in speedups occur at the highest processor power cap of 120W, since power is unlimited (i.e., the application is not consuming this amount of power). The difference is inversely related to the processor power cap. That is to say, the difference grows as the power cap is reduced. This is a result of efficiently reallocating the limited power to the nodes that need it most.

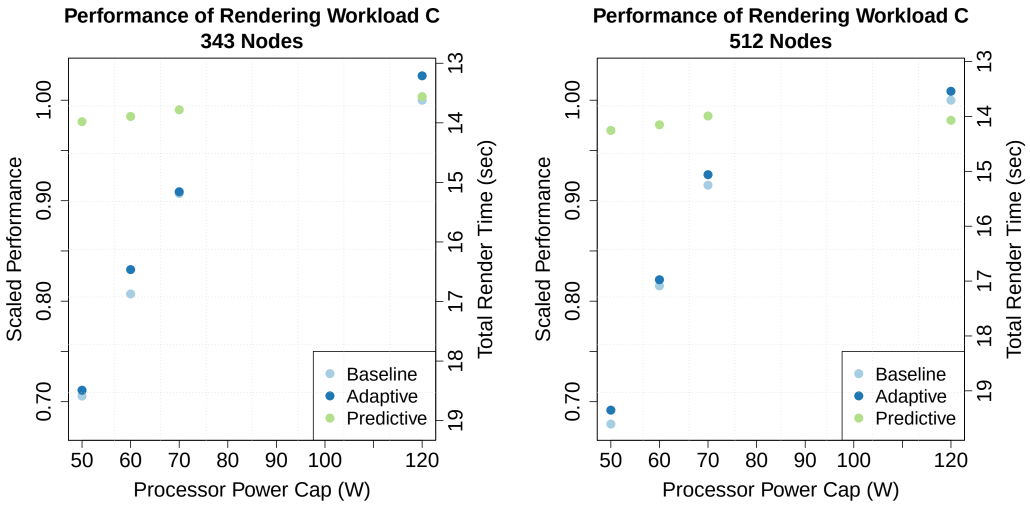
The highest concurrency of 512 nodes shows the largest difference of 28% at the lowest node power cap of 100W (i.e., 50W per processor). The predictive strategy identifies the nodes with no work to do, setting the lowest power cap, enabling some nodes with lots of work to do to run at 200W. The adaptation strategy is not able to fully exploit the level of imbalance across the nodes.

5.6 Conclusion and Future Work

In this chapter, we compared the effectiveness of two different approaches in dynamically scheduling power to improve the performance of scientific visualization workloads in a power-limited environment. Specifically, we considered an adaptive and predictive approach and focused on parallel ray tracing, which is a highly variable workload. Like other HPC research on overprovisioning, both strategies determine per-node power caps in an effort to allocate power to the nodes that needed it most. The adaptive method is online and adapts power based on current workload execution behaviors. This method can be well-suited for predictable



(a) 125 Nodes (b) 216 Nodes



(c) 343 Nodes (d) 512 Nodes

Figure 24. The scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the Baseline for Rendering Workload C at higher levels of concurrency. We normalize the performance to that of the Baseline, which applies a uniform power cap of 120W. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context.

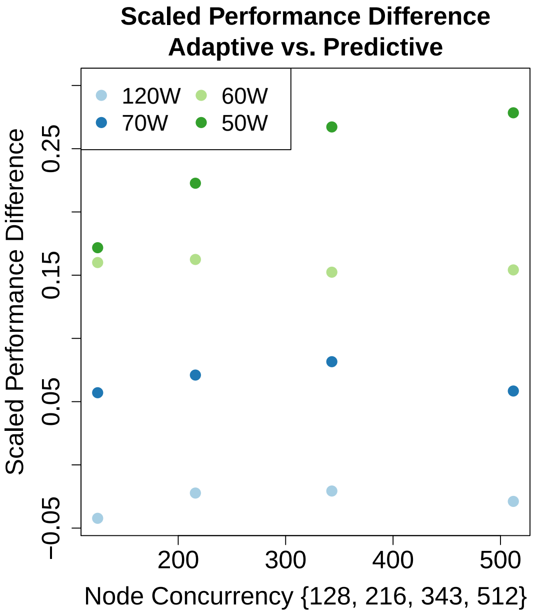


Figure 25. Difference in speedup between the adaptive and predictive strategies at different levels of node concurrency.

workloads whose iterations are regular from one to the next. The predictive method introduced in Chapter IV uses an existing performance model to assign power based on a prediction of execution time. While this method is well-suited for irregular workloads (e.g., visualization), there is a high overhead cost (i.e., human time) to create the performance models. The resulting study demonstrated that the predictive approach is beneficial for irregular visualization workloads, with results as much as 27% faster than an adaptive strategy.

In terms of future work, we would like to understand which scenarios are beneficial for prediction and which scenarios are beneficial for adaptation. One of the downsides of using prediction is the effort in creating and validating accurate performance models. On the other hand, one of the downsides of using adaptation is needing time at the beginning of the routine to identify the most and least efficient nodes. We want to explore if prediction is possible with minimal overhead costs, and identify cost effective scenarios for using adaptation.

CHAPTER VI

CONCLUSION AND FUTURE DIRECTIONS

6.1 Synthesis

Scientific visualization is a key component in the scientific discovery process. It enables the exploration and analysis of scientific data, and the ability to communicate findings through a comprehensible image. Visualization at exascale will be challenging due to constraints in power usage. Under this power-limited environment, visualization algorithms merit special consideration, since they are more data intensive in nature than typical HPC applications like simulation codes. At present, there is a very limited set of work addressing the challenges of visualization and analysis with respect to power constraints on future architectures. The central question of this thesis is: *How can we optimize the performance of scientific visualization workloads in a power-limited environment?*

The research presented in this dissertation explored this field uniquely positioned at the intersection of power-constrained HPC and scientific visualization. This dissertation first provided an in-depth exploration of the variation in visualization routines. We found that most visualization algorithms are data intensive and have favorable power and performance tradeoffs when the clock frequency is slowed whether directly or indirectly through lower power limits enforced on Intel architectures. Then, we exploited the variation present in visualization algorithms and showed performance improvements. We used existing performance models for parallel rendering to develop a power-aware resource manager called PaViz. PaViz dynamically allocates power based on a prediction of execution times. We compare this predictive approach with an adaptive approach in a runtime system known as GEOPM, which automatically adapts power to the current execution behaviors. This piece of the dissertation found that prediction performs better than adaptation on irregular visualization workloads.

Looking forward, the HPC community is moving towards heterogeneous computing for improved power efficiency and performance. This new supercomputing architecture will significantly challenge the way we have traditionally designed our software. The research in this dissertation specifically targeted Intel x86 architectures, since Intel was one of the early architectures to expose power capping on their processors. Additional vendors have since exposed similar power capping mechanisms on their platforms. The granularity of control as well as the range of available power limits may vary across vendor architectures. For example, Intel's power capping technology controls the processor and DRAM domain through a register interface. The underlying clock frequencies are modulated to adjust to the specified power limit. IBM's power capping technology applies the limit at the server-level specified through a platform management interface. The server power limit will affect the power limit allowed to each component contained within the server. NVIDIA's power capping technology affects the accelerator itself by modulating clock frequencies similar to Intel.

Each architecture may expose a different range of acceptable power limits. The range will depend on the minimum and maximum power limits that allow for reliable operation. If the power limit is too low, then the components will not operate, and if the power limit is too high, then the components will overheat due to excessive power. Depending on the range of power limits (and the range of clock frequencies), this may enable favorable opportunities for power and energy savings. However, for heterogeneous platforms containing CPU and GPU architectures, it may not be a flexible solution to set power limits per device. Instead, it may be better to expose a ratio that informs how node power should be shared between the two central components.

VTK-m has enabled the development of visualization algorithms portable across many different types of platforms. This dissertation provided a thorough exploration of the opportunities for power and energy savings for a representative set of VTK-m algorithms as a more severe power limit is applied to an Intel processor. Heterogeneous GPU and CPU architectures are becoming prominent in supercomputer architectures. Visualization workloads targeting GPU architectures may expose different tradeoffs, since users may decide to run the workload on the GPU instead of the CPU. GPUs have a higher power usage, since they are designed for highly computational algorithms. The ability to slow down the GPU may not result in favorable energy and power savings as we have seen with CPU architectures.

The existing performance models for parallel rendering use rendering specific input variables, which will not be impacted as we move from one architecture to another. What will change as we move to a different architecture are the model coefficients, since these are determined by gathering runtime performance data across a range of configurations representing in situ rendering use cases. At present, the coefficients are obtained offline, but a more advanced infrastructure may determine and adjust the coefficients online.

Developing performance models for each visualization operation is untenable. One solution is to develop performance models for classes of algorithms. Alternatively, another solution is to develop feedback infrastructure where algorithms expose application-level information to an adaptive framework (like

GEOPM), enabling the framework to make more informed decisions on where the power should be allocated to do the most good.

6.2 Future Directions

There are four areas of future research that can build on the work presented in this dissertation. We detail them in the following subsections.

6.2.1 Exploring Tradeoffs on Different Hardware Architectures.

The majority of this research targeted Intel processors, which have provided fine-grained power capping technologies since the Sandy Bridge processor family. Future exascale architectures are moving to heterogeneous computing, meaning we will need to understand how the power and performance tradeoffs change when moving to different hardware platforms, such as accelerators and other CPUs. Every vendor has a different implementation of power capping, ranging from the granularity at which the power cap is applied (e.g., socket, motherboard, node) to how reactive the system is to the power cap. This is traditionally an operation limited to a set of privileged users. Exploring how the power and performance tradeoffs change when moving to a different architecture will impact how we develop future power scheduling strategies. Should future research explore the tradeoffs at large scale, a driver must be developed for each architecture to enable setting of power caps from user space.

6.2.2 Evaluating When to Use Prediction or Adaptation.

Chapter IV and Chapter V showed that additional performance for a distributed visualization application can be realized by leveraging a performance model or using adaptation to determine how to schedule power across nodes within a job. This area of research is two-fold. First, determining a theoretical upper bound for improved performance is critical in determining the effectiveness of such a runtime system. Once the theoretical bound has been identified, we can evaluate the efficacy of the adaptive or predictive approaches on performance.

Determining when to use prediction and when to use adaptation is a key area of future research. Prediction has high overhead costs (i.e., human time) to create the models, and may not be feasible if a performance model cannot be generated for the workload. But, prediction is also well-suited for irregular and unpredictable workloads, like visualization. On the other hand, adaptation is an online approach, minimizing the overhead costs of needing to do setup. For regular workloads, the online approach can spend the first few iterations learning the unique execution behaviors, and then apply the optimum strategy to the remaining iterations to improve performance. For irregular workloads, using adaptation may be a suboptimal strategy as the decisions may never converge on a solution.

6.2.3 Creating Additional Performance Models. This research used an existing performance model specifically for rendering, and showed it resulted in better performance improvements than the adaptive strategy. However, establishing models for other visualization algorithms is a challenging task, and may not be possible for all available visualization workloads. There are hundreds of different visualization algorithms; some will be straight-forward to create an accurate performance model, while others will be more difficult to estimate performance based on their input parameters. Depending on the number of visualization operations chained together in the pipeline, it may be less overhead for similar performance gains to use adaptation rather than create a performance model for each operation.

6.2.4 Integration into Production Resource Manager. If we want to make a significant impact on the mission-critical applications doing visualization, then the power-aware runtime system should be integrated into a production manager, like SLURM. In this model, SLURM would be extended to assign power budgets to each job [9], then it would be the duty of the job to manage the budget between the nodes it has been allocated. The power-aware resource manager could use the entirety of the power budget it has been given to complete its job. Alternatively, since findings in Chapter III identified that a large set of visualization algorithms consume very little power, the resource manager may choose to use a subset of the original budget, letting SLURM re-allocate the extra power to another job.

One of the challenges to improving performance with power reallocation is having a visualization pipeline that has a long enough execution time to see the impacts of a change in the power limit. Another challenge when considering power shifting is ensuring that the larger system does not exceed its system-wide power budget. Exceeding the system-wide power budget can cause damage to the underlying hardware and limit its lifespan.

REFERENCES CITED

[1] Nvidia index. `https://developer.nvidia.com/index`. Accessed: 2017-03-02.

[2] geopm. `https://github.com/geopm/geopm`, 2016.

[3] msr-safe. `https://github.com/llnl/msr-safe`, 2016.

[4] CloverLeaf. `http://uk-mac.github.io/CloverLeaf/`, 2017.

[5] Intel Processor Event Reference. `https://download.01.org/perfmon/index/`, 2017.

[6] Greg Abram and Lloyd A. Treinish. An Extended Data-Flow Architecture for Data Analysis and Visualization. Research report RC 20001 (88338), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, February 1995.

[7] V. Adhinarayanan, W. Feng, J. Woodring, D. Rogers, and J. Ahrens. On the Greenness of In-Situ and Post-Processing Visualization Pipelines. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, pages 880–887, May 2015.

[8] Sean Ahern, Arie Shoshani, Kwan-Liu Ma, Alok Choudhary, Terence Critchlow, Scott Klasky, Valerio Pascucci, Jim Ahrens, E. Wes Bethel, Hank Childs, Jian Huang, Kenneth I. Joy, Quincey Koziol, Jay Lofstead, Jeremy Meredith, Ken Moreland, George Ostrouchov, Mike Papka, Venkat Vishwanath, Matthew Wolf, Nick Wright, and K. John Wu. Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization, July 2011.

[9] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 9–17, Sep. 2014.

[10] James Ahrens, Berk Geveci, and Charles Law. ParaView: An End-User Tool for Large Data Visualization. January 2005.

[11] James Ahrens, Sébastien Jourdain, Patrick O'Leary, John Patchett, David H. Rogers, and Mark Petersen. An Image-based Approach to Extreme Scale in Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 424–434, Piscataway, NJ, USA, 2014. IEEE Press.

[12] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, Tony Mezzacappa, Parviz Moin, Mike Norman, Robert Rosner, Vivek Sarkar, Andrew Siegel, Fred Streitz, Andy White, and Margaret Wright. The Opportunities and Challenges of Exascale Computing. Technical report, Advanced Scientific Computing, Fall 2010.

[13] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. ParaView Catalyst: Enabling In Situ Data Analysis and Visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ISAV2015, pages 25–29, New York, NY, USA, 2015. ACM.

[14] Pete Beckman, Ron Brightwell, Bronis R. de Supinski, Maya Gokhale, Steven Hofmeyr, Sriram Krishnamoorthy, Mike Lang, Barney Maccabe, John Shalf, and Marc Snir. 2013 Exascale Operating and Runtime Systems. Technical report, Advanced Scientific Computing Research (ASCR), Nov 2012.

[15] James Bigler, James Guilkey, Christiaan Gribble, Charles Hansen, and Steven G. Parker. A Case Study: Visualizing Material Point Method Data. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'06, pages 299–306, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[16] Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[17] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 357–372. October 2012.

[18] Paolo Cignoni, Paola Marino, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Speeding up Isosurface Extraction using Interval Trees. *Visualization and Computer Graphics, IEEE Transactions on*, 3(2):158–170, 1997.

[19] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3B*. Intel Corporation, December 2017.

[20] Advanced Micro Devices. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*, January 2013.

[21] Jack Dongarra et al. The International Exascale Software Roadmap. *International Journal of High Performance Computer Applications*, 25(1):3–60, 2011.

[22] Jonathan Eastep, Steve Sylvester, Christopher Cantalupo, Brad Geltz, Federico Ardanaz, Asma Al-Rawi, Kelly Livingston, Fuat Keceli, Matthias Maiterth, and Siddhartha Jana. Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions. In Julian M. Kunkel, Rio Yokota, Pavan Balaji, and David Keyes, editors, *High Performance Computing*, pages 394–412, Cham, 2017. Springer International Publishing.

[23] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. E. Jansen. The ParaView Coprocessing Library: A Scalable, General Purpose In Situ Visualization Library. In *2011 IEEE Symposium on Large Data Analysis and Visualization*, pages 89–96, Oct 2011.

[24] Paul F. Fischer, James W. Lottes, and Stefan G. Kerkemeier. nek5000 Web page, 2008. http://nek5000.mcs.anl.gov.

[25] Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, and Tyler K. Bletsch. Just-in-time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs. *J. Parallel Distrib. Comput.*, 68(9):1175–1185, September 2008.

[26] Vincent W. Freeh and David K. Lowenthal. Using Multiple Energy Gears in MPI Programs on a Power-scalable Cluster. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 164–173, New York, NY, USA, 2005. ACM.

[27] Marc Gamell, Ivan Rodero, Manish Parashar, Janine C. Bennett, Hemanth Kolla, Jacqueline Chen, Peer-Timo Bremer, Aaditya G. Landge, Attila Gyulassy, Patrick McCormick, Scott Pakin, Valerio Pascucci, and Scott Klasky. Exploring Power Behaviors and Trade-offs of In-situ Data Analytics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 77:1–77:12, New York, NY, USA, 2013. ACM.

[28] Rong Ge, Xizhou Feng, and Kirk W. Cameron. Improvement of Power-Performance Efficiency for High-End Computing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12*, IPDPS '05, pages 233.2–, Washington, DC, USA, 2005. IEEE Computer Society.

[29] Rong Ge, Xizhou Feng, Wu chun Feng, and K.W. Cameron. CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters. In *Proceedings of the 2007 International Conference on Parallel Processing*, pages 18–18, Sept 2007.

[30] IBM. *IBM EnergyScale for POWER8 Processor-Based Systems*, November 2015.

[31] Björn Johnsson, Per Ganestam, Michael Doggett, and Tomas Akenine-Möller. Power Efficiency for Software Algorithms Running on Graphics Processors. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 67–75. Eurographics Association, 2012.

[32] Alan Kaminsky. *Big CPU, Big Data: Solving the World's Toughest Computational Problems with Parallel Computing.* CreateSpace Independent Publishing Platform, USA, 1st edition, 2016.

[33] I Karlin, J McGraw, J Keasler, and B Still. Tuning the LULESH Mini-App for Current and Future Hardware. In *Nuclear Explosive Code Development Conference Proceedings (NECDC12)*, 2012.

[34] AJ Kunen, TS Bailey, and PN Brown. KRIPKE-A Massively Parallel Transport Mini-App. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2015.

[35] S. Labasan, M. Larsen, and H. Childs. Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm. In *Proceedings of the 2015 IEEE 5th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 73–80, Oct 2015.

[36] S. Labasan, M. Larsen, H. Childs, and B. Rountree. Power and Performance Tradeoffs for Visualization Algorithms. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019.

[37] Stephanie Labasan, Matthew Larsen, Hank Childs, and Barry Rountree. Evaluating Predictive and Adaptive Methods for Improving Visualization Performance. *J. Parallel Distrib. Comput.*

[38] Stephanie Labasan, Matthew Larsen, Hank Childs, and Barry Rountree. PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance. In Alexandru Telea and Janine Bennett, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2017.

[39] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, ISAV'17, pages 42–46, New York, NY, USA, 2017. ACM.

[40] Matthew Larsen, Eric Brugger, Hank Childs, Jim Eliot, Kevin Griffin, and Cyrus Harrison. Strawman: A Batch In Situ Visualization and Analysis Infrastructure for Multi-Physics Simulation Codes. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 30–35. ACM, 2015.

[41] Matthew Larsen, Cyrus Harrison, James Kress, David Pugmire, Jeremy Meredith, and Hank Childs. Performance Modeling of In Situ Rendering. In *The International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, November 2016.

[42] Michael A. Laurenzano, Mitesh Meswani, Laura Carrington, Allan Snavely, Mustafa M. Tikir, and Stephen Poole. Reducing Energy Usage with Memory and Computation-aware Dynamic Frequency Scaling. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Euro-Par'11, pages 79–90, Berlin, Heidelberg, 2011. Springer-Verlag.

[43] Marc Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics (TOG)*, 9(3):245–261, 1990.

[44] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[45] M. Maiterth, G. Koenig, K. Pedretti, S. Jana, N. Bates, A. Borghesi, D. Montoya, A. Bartolini, and M. Puzovic. Energy and Power Aware Job Scheduling and Resource Management: Global Survey - Initial Analysis. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 685–693, May 2018.

[46] AC Mallinson, David A Beckingsale, WP Gaudin, JA Herdman, JM Levesque, and Stephen A Jarvis. Cloverleaf: Preparing Hydrodynamics Codes for Exascale. *The Cray User Group*, pages 6–9, 2013.

[47] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. Supinski. *High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings*, chapter A Run-Time System for Power-Constrained HPC Applications, pages 394–408. Springer International Publishing, Cham, 2015.

[48] S Martin and M Kappel. Cray XC30 Power Monitoring and Management. *Proceedings of CUG*, 2014.

[49] K. Moreland, C. Sewell, W. Usher, L. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. Ma, H. Childs, M. Larsen, C. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May 2016.

[50] Kenneth Moreland, Wesley Kendall, Tom Peterka, and Jian Huang. An Image Compositing Solution at Scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 25. ACM, 2011.

[51] Kenneth Moreland and Ron Oldfield. Formal Metrics for Large-Scale Parallel Performance. In Julian M. Kunkel and Thomas Ludwig, editors, *High Performance Computing*, pages 488–496, Cham, 2015. Springer International Publishing.

[52] Philip J. Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[53] NVIDIA. *NVML Reference Manual*, May 2015.

[54] Patrick O'Leary, James Ahrens, Sébastien Jourdain, Scott Wittenburg, David H Rogers, and Mark Petersen. Cinema Image-Based In Situ Analysis and Visualization of MPAS-ocean Simulations. *Parallel Computing*, 55:43–48, 2016.

[55] B. W. O'Shea, G. Bryan, J. Bordner, M. L. Norman, T. Abel, R. Harkness, and A. Kritsuk. Introducing Enzo, an AMR Cosmology Application. *ArXiv Astrophysics e-prints*, March 2004.

[56] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive Ray Tracing for Isosurface Rendering. In *Proceedings of the Conference on Visualization '98*, VIS '98, pages 233–238, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[57] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski. Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, New York, NY, USA, 2013. ACM.

[58] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski. Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 121–132, New York, NY, USA, 2015. ACM.

[59] Allan Porterfield, Rob Fowler, Sridutt Bhalachandra, and Wei Wang. OpenMP and MPI Application Energy Measurement Variation. In *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, E2SC '13, pages 7:1–7:8, New York, NY, USA, 2013. ACM.

[60] James Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[61] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, and P. Bremer. Evaluation of In-Situ Analysis Strategies at Scale for Power Efficiency and Scalability. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 156–164, May 2016.

[62] Barry Rountree, David K. Lowenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[63] Vivek Sarkar, William Harrod, and Allan E Snavely. Software Challenges in Extreme Scale Systems. In *Journal of Physics: Conference Series*, page 012045, 2009.

[64] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen. The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 93–ff. IEEE Computer Society Press, 1996.

[65] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI – The Complete Reference: The MPI Core, 2nd edition.* MIT Press, Cambridge, MA, USA, 1998.

[66] C. Upson, T. A. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *Computer Graphics and Applications*, 9(4):30–42, July 1989.

[67] I Wald, GP Johnson, J Amstutz, C Brownlee, A Knoll, J Jeffers, J Günther, and P Navratil. OSPRay-A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):931–940, 2017.

[68] Brad Whitlock, Jean M Favre, and Jeremy S Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EGPGV '11, pages 101–109, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[69] Z. Zhang, M. Lang, S. Pakin, and S. Fu. Trapped Capacity: Scheduling under a Power Cap to Maximize Machine-Room Throughput. In *Energy Efficient Supercomputing Workshop (E2SC), 2014*, pages 41–50, Nov 2014.