Evaluating Adaptive and Predictive Power Management Strategies for Optimizing Visualization Performance on Supercomputers

Stephanie Brink[a], Matthew Larsen[a], Hank Childs[b], Barry Rountree[a]

[a]*Lawrence Livermore National Laboratory, 7000 East Avenue, Livermore, California 94550, USA*
[b]*University of Oregon, 1477 E 13th Avenue, Eugene, Oregon 97403, USA*

Abstract

Power is becoming an increasingly scarce resource on the next generation of supercomputers, and should be used wisely to improve overall performance. One strategy for improving power usage is hardware overprovisioning, *i.e.,* systems with more nodes than can be run at full power simultaneously without exceeding the system-wide power limit. With this study, we compare two strategies for allocating power throughout an overprovisioned system — adaptation and prediction — in the context of visualization workloads. While adaptation has been suitable for workloads with more regular execution behaviors, it may not be as suitable on visualization workloads, since they can have variable execution behaviors. Our study considers a total of 104 experiments, which vary the rendering workload, power budget, allocation strategy, and node concurrency, including tests processing data sets up to 1 billion cells and using up to 18,432 cores across 512 nodes. Overall, we find that prediction is a superior strategy for this use case, improving performance up to 27% compared to an adaptive strategy.

Keywords: high performance computing, parallel systems, power management, scientific visualization, power-constrained computing, runtime systems

1. Introduction

Power has become one of the most important considerations for leading-edge supercomputers, as high power usage can be very expensive. The last decade has seen significant research in how to achieve increases in FLOPS without requiring commensurate increases in power. One significant innovation for this challenge has been utilizing many-core accelerators, in particular GPUs, since they offer more FLOPS per Watt. While the efficiency increases from these hardware architectures have been significant, additional research has introduced complementary approaches that improve FLOPS per Watt ratios even further.

With this research, we consider the overprovisioning approach for improving power utilization on HPC systems [1, 2, 3, 4]. With a traditional (non-overprovisioning) approach, all nodes in the supercomputer are able to consume their theoretical maximum power draw simultaneously. However, very few applications are capable of using this amount of power, meaning this traditional approach not only wastes power capacity, but also increases trapped capacity [5]. An overprovisioned system addresses this problem by adding more compute nodes, while enforcing a strict system power budget.

Overprovisioning works by capping the power individual nodes can use, in order to ensure that the total power usage never exceeds the limit. A baseline approach for allocating power is to apply a uniform power cap across all compute nodes. However, uniform power capping 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 will become a bottleneck and limit 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 finished executing. A better strategy is to actively assign power to where it will result in the most benefit. This is the direction we pursue with this study. In the ideal case, the power assignments will enable all nodes to finish their execution at the same time, despite varying workloads.

Many research efforts in assigning power on supercomputers have focused on the adaptive approach. Adaptive power management schemes actively monitor the performance of the application at regular and frequent intervals and redistribute the power across the nodes based on current progress. In practice, this approach commonly focuses on the cycles of an HPC workload running across the supercomputer. At the end of each cycle, the central manager divides the power based on outcomes from the cycle that just occurred. This approach works well for computational workloads that are similar from cycle to cycle, meaning an assignment of power based on the previous cycle is often well-suited for the next cycle.

With our previous work on the PaViz [6] system, we introduced an alternate strategy for power assignment: prediction. Our system considered scientific visualization workloads, where the work from cycle to cycle is more variable. Our predictive approach used a performance model to assess how much work would need to be done based on the application configuration and input parameters, and then adapted power based on the results of the model. Our study focused on different assignment strategies, and showed that many of them were superior to

Email address: brink2@llnl.gov (Stephanie Brink)

uniformly distributing the power. That said, our previous study on PaViz did not compare with the adaptive approach.

The relative benefits of adaptation and prediction are not well understood. The predictive approach requires additional constructs to perform its predictions (*i.e.,* performance models), where the adaptive approach does not. From this perspective, the adaptive approach is superior. However, our hypothesis is that the predictive approach will offer superior performance for variable workloads (workloads whose execution time vary from task to task and node to node), specifically the variable workloads from scientific visualization. The purpose of our current study is to evaluate whether this hypothesis is true, and the extent to which prediction can outperform adaptation. To evaluate our hypothesis, we leverage two existing power-aware runtime systems, GEOPM [7, 8] and PaViz [6], which make use of the adaptive and predictive approaches. Finally, we believe the results of this evaluation is important for the HPC community because of the rise of in situ processing — visualization is now commonly running at scale on supercomputers [9, 10, 11].

2. Related Work

Our study compares adaptive and predictive power management in the context of a visualization workload. In this section, we survey motivation and research on power management in HPC (Section 2.1) and important aspects of visualization workloads for HPC systems (Section 2.2).

2.1. Power

This subsection discusses related research in HPC power management (Section 2.1.1), as well as power management for visualization workloads (Section 2.1.2).

2.1.1. Power in HPC

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 [12, 13, 14]. The common goal of these approaches was to minimize energy usage by incurring a small performance degradation.

More recent research efforts center on enforcing power caps via vendor-specific technologies. Some of these technologies include Intel's Running Average Power Limit [15], AMD's Application Power Management [16], IBM EnergyScale [17], and NVIDIA's NVML [18]. A runtime system incorporating these technologies, Conductor [19], used initial iterations to determine an ideal schedule of per-node power caps, thread concurrency, and per-core operating frequency. Another runtime system incorporating these technologies is GEOPM [7, 8], which is a scalable production-grade 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 visualization workloads. We use GEOPM in our experiments to evaluate an adaptive power management strategy, and discuss the specifics of GEOPM in more depth in Section 3.1.2. Finally, while we focus on GEOPM in our study, there have been other significant efforts for the adaptive power management strategy [12, 13, 19].

There have been several research efforts that develop and make use of offline models to save energy and power with minimal performance degradation [14, 20, 21, 22, 23, 24]. Many of these offline models were based on behavior exhibited by traditional HPC benchmarks, which tend to have regular computation patterns. PaViz [6] used a performance model to do predictive power assignments for visualization workloads, which are highly irregular. PaViz used prediction to improve performance of the visualization workloads under a limited power budget. This runtime system is also used in our experiments to evaluate a predictive power management strategy, and is discussed in further depth in Section 3.2.2.

There has also been previous work on overprovisioning in a data center to optimize throughput. Studies have shown that consolidating workloads on fewer servers in a data center can be more power-efficient [25, 26, 27, 28]. However, consolidation is not a strategy that is used in the HPC domain. That is to say, nodes in an HPC system are not time shared between users (*i.e.,* multiple users do not run on the same nodes simultaneously) due to the large resource requirements of typical HPC workloads, such as scientific simulations. Furthermore, saving power and energy by turning on and off nodes is another strategy not common in HPC. HPC nodes are typically turned on and off only at system reboot, and are otherwise kept online to maximize throughput of the system.

2.1.2. Optimizing Power Usage for Visualization Workloads

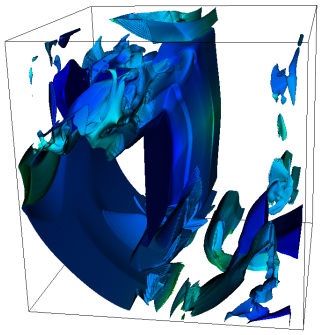
Research efforts focusing on power usage with respect to visualization workloads are minimal. Some research efforts explored the power profiles for an in situ workflow, particularly with respect to how data is moved through the hierarchy [29, 30]. Separate research focused on how visualization workload configurations impact overall power usage. Labasan et al. [31] provided an initial exploration of the impacts of different configuration options on power and performance within a node. Specifically, this work looked at how different parameters impact the performance for an isosurfacing workload as the processor clock frequency was gradually reduced. A related work by Gamell et al. [32] looked at the power and performance tradeoffs of various parameters for visualization workloads at larger node counts.

2.2. Visualization

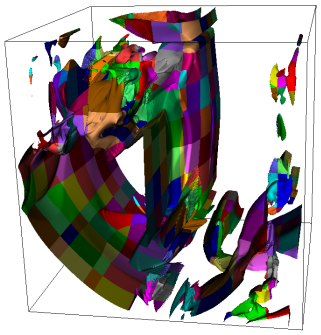
This subsection is divided into two parts: why visualization is an important workload for HPC systems (Section 2.2.1) and why visualization workloads are highly variable across cycles and processes (Section 2.2.2).

2.2.1. Visualization in HPC

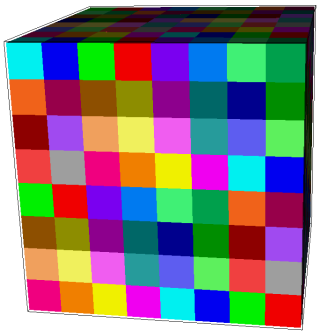
Upward trends in supercomputing are mandating that visualization routines be executed at scale, making it important to



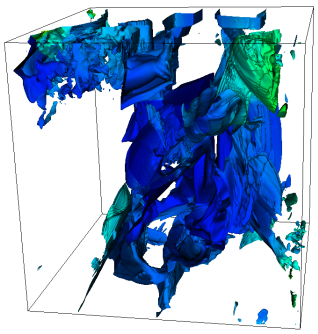
(a) An isosurface of the velocity field at simulation timestep t_n



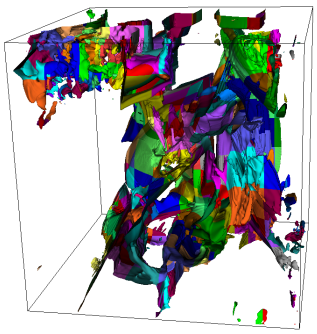
(b) Isosurface at t_n colored by MPI domain



(c) MPI domain decomposition



(d) An isosurface of the velocity field at simulation timestep t_{n+1}



(e) Isosurface at t_{n+1} colored by MPI domain

Figure 1: An illustration of the level of imbalance introduced by visualization between simulation cycles and across ranks. The rightmost figure 1(c) shows the domain decomposition of the mesh across 512 MPI processes. The mesh has been broken down into multiple pieces of equal size and mapped to individual MPI processes, denoted by the different colors. The left figures 1(a) and 1(d) show the resulting isosurfaces created by applying a contour to the velocity field of the CloverLeaf hydrodynamics simulation at two different timesteps. The middle figures 1(b) and 1(e) show the domain decomposition of the isosurface at these two simulation timesteps. At the timestep shown in figure 1(b), the resulting isosurface only contains data on 317 of the 512 MPI processes (that is, 38% of the MPI processes contain no data, and have no work to do). The isosurface at timestep t_{n+1} has an even larger percentage of MPI processes with no data.

optimize their power and performance. Traditionally, visualization has been done as a post-hoc process. With post-hoc processing, the simulation runs to completion, periodically writing its data to disk. A visualization program, such as VisIt [33] or ParaView [34], ingests the data from disk and performs various analyses. However, post-hoc processing is becoming increasingly less viable: as FLOPS increase, simulation codes produce more and more data, and yet I/O bandwidth is not increasing commensurately [11, 35]. As a result, post-hoc processing has become too slow in many cases, as performance becomes dominated by slow I/O. To avoid these costs, the visualization community has moved towards a new paradigm: in situ processing [9, 10]. In situ processing entails running visualization while the simulation generates data, and often occurs using the same hardware resources as the simulation code. In all, visualization regularly takes as much as 10% of the total cycles for a computational simulation.

Visualization occurs as a set of transformations (*i.e.,* pipeline). Initially, the input to the pipeline is the simulation's entire dataset, so the data is evenly distributed across the ranks. During the analysis, the visualization operator is applied to extract a particular region of the data that is of interest. All subsequent visualization operations work on the output data, which is a smaller subset of the simulation's full dataset. Since the visualization operators have extracted a smaller subset of the data, there is likely a bigger imbalance of ranks with data and ranks with no data. In this study, we focus on rendering, since it is a critical component of the visualization pipeline, however, other visualization operations would run into the same imbalance.

While visualization algorithms are varied and diverse, they all conclude with a "rendering" phase where computer graphics algorithms are applied to transform data (typically surfaces or volumes) into images. Thus, rendering is a critical component of visualization. Further, in situ visualization is increasingly considering a model with many, many renderings. Since in situ visualization is typically run with no human-in-the-loop, the idea is to render from many camera locations, in hopes of getting some that will interest a domain scientist. This paradigm is the main idea behind the popular Cinema project [36, 37]. In this scenario, rendering goes from being a critical aspect in per-

formance to the dominant aspect in performance. For example, consider the use case of contouring (also known as isosurfacing), where surfaces are extracted from the mesh at all locations that match some isovalue. While calculating a contour typically takes about the same amount of time as rendering it, rendering the contour hundreds of times from hundreds of camera positions will consume most of the overall visualization time.

There are different techniques for rendering. In this study, we use ray tracing, which is good for supercomputing use cases, since it performs well when there is much more data than pixels and for repeated rendering (*i.e.,* Cinema). With ray tracing, rays are traced from the camera location through pixels and intersect with the geometry to render. The process of tracing rays is embarrassingly parallel, although scaling the algorithm to render millions of pixels is challenging. Despite these challenges, previous works have made good progress on achieving parallel ray tracing [38, 39].

2.2.2. Visualization Workloads and Their Irregularity

Rendering is different from traditional simulation workloads. Simulations typically solve differential equations on a spatial discretization (*i.e.,* a mesh), and number of operations a simulation executes is tied to the number of elements and the complexity of the modeled physics. Rendering performance, on the other hand, depends on the amount of geometry to render, the relationship between that geometry and the camera position, and the size of the output image. The first two factors lead to highly variable performance and are described later in this subsection. That said, the imbalance for rendering workloads presents systems such as ours with opportunities to shift resources (such as power) to MPI ranks with significantly more work than others, while taking resources away from ranks with less work.

Figure 1 illustrates the imbalance that can occur across MPI tasks in terms of data size (*e.g.,* number of triangles to render) and across time (*e.g.,* simulation cycles). In this example, the original mesh data is distributed across 512 MPI processes. After contouring the velocity field at time t_n, only 317 (62%) of the processes contain the isosurface, leaving 195 (38%) of the processes with no data. As the simulation progresses and the wavefront propagates, the velocity field changes, which will cause the amount of geometry in the isosurface to change. More importantly, the location of this geometry will change as well, causing the distribution between the MPI processes to change.

Rendering not only inherits the input imbalance (*e.g.,* the contour example in the previous paragraph), but adds additional imbalance depending on the placement of the camera. Only geometry that is in front of the camera is actually rendered. Building on the previous example, one possible camera placement leaves only half of the remaining 62% of the data within the view of the camera. Consequently, rendering can only utilize 31% of the computational resources without reorganizing the data.

Finally, while data redistribution would be one possible solution, in situ visualization operates under more constraints (*i.e.,* time and memory), and so this solution is not pursued in practice. For this reason, we do not consider redistribution as a comparator for our approach.

2.3. Our Study in Relation to Previous Related Works

Overall, our study continues the research direction of overprovisioning and adapting power to where it will do the most good. The common technique for decision making is the adaptive approach. Our previous work on PaViz considered a different decision making technique, which is the predictive approach. However, this previous work did not evaluation the adaptive approach, leaving open questions about how the adaptive and predictive approaches compare. This study aims to address this gap. It is, to our knowledge, the first ever to compare the adaptive and predictive approaches in the context of overprovisioning and power adaptation.

3. Power-Aware Scheduling Strategies

The power limitations of future supercomputers will require allocating the power wisely in order to optimize performance. One solution for optimizing the system performance and power utilization is overprovisioning. In an overprovisioned system, more compute nodes are added to the system, but their individual power usage has been capped so as not to exceed the system power budget. A baseline approach is to apply a uniform power cap across all nodes. However, this strategy is not ideal for a load imbalanced workload, like visualization, since the performance is determined by the node with the most amount of work. A better strategy is to schedule power to where it will do the most good (*e.g.,* the critical path), speeding up the nodes that are further behind and slowing down the nodes that are further ahead.

One key requirement for power scheduling to be successful is to redistribute the power without exceeding the system-wide power limit. Exceeding the given power limit can cause electrical issues at the system level, which may result in failures or breakage. The following subsections detail the adaptive and predictive power scheduling strategies evaluated in this paper. Additionally, we provide an overview of two runtime systems—GEOPM and PaViz—that apply these power scheduling strategies in practice.

3.1. Adaptive Power Management

This subsection provides an overview of adaptive power management (Section 3.1.1), as well as a description of GEOPM (Section 3.1.2).

3.1.1. Overview of Adaptive Power Management

Adaptive power management strategies monitor the performance of the application at frequent intervals and redistribute the power across the nodes. At each interval, the adaptive strategy gives more power to nodes that are further ahead in completing their assigned work, while removing power from nodes that are falling behind. With this approach, power allocations to each node are non-uniform, and are scaled by the amount of load imbalance in the application.

One way of monitoring real-time performance (or progress) of the application is to measure the loop execution time on each node. The loop execution times on each node are compared to identify critical path nodes and determine how to distribute the power to correct the imbalance. An adaptive strategy will repeat this process of monitoring and redistributing the power until the application has finished executing. Because adaptive strategies are dynamically reacting to the application performance, it may take several iterations of the application for the global power decisions to converge.

Finally, interval frequency is an important consideration. The frequency at which these decisions occur is configurable, impacted by the amount of overhead and the granularity of monitoring to catch changes in behavior. For applications that are fairly regular in their execution behaviors, monitoring less frequently is well-suited, since it is likely that behaviors will stay the same throughout. On the other hand, for applications that are highly irregular, there may be swings in the power management decisions, leading to additional performance left on the table. There are also limits to how short the interval can be. In some cases, the hardware can take as much as one second to enforce the desired power limit. In response, the interval frequency must be greater than the frequency at which the underlying hardware knob can enforce the power limit, so the workload has time to react the new power limit.

3.1.2. Adaptive Runtime System: GEOPM

GEOPM [7, 8] is an open-source framework for power and energy management on future HPC systems. It is a collaborative project, started and supported by Intel. GEOPM is designed to support Intel platforms, but can be extended to support other hardware platforms providing power management capabilities, such as IBM and NVIDIA. GEOPM is also designed to support the different power and energy management needs across many supercomputing facilities through an agent plugin architecture. For example, included in GEOPM are two different agents known as the Power Balancer agent (used in this study) and the Energy Efficient agent. While the Power Balancer agent shifts power across nodes through power capping, the Energy Efficient agent optimizes for energy or performance by adjusting CPU frequencies.

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 examples of 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.

For global coordination of power decisions, GEOPM's runtime system uses a hierarchical tree to gather information about the application's performance and redistributes power across nodes. This hierarchical design allows GEOPM to be performant at high levels of concurrency. There is a controller running on a single thread per node to handle different roles and tasks of the GEOPM runtime. One of the controllers is the root node, and is tasked with aggregating the performance per-node, and passing 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.

3.2. Predictive Power Management

This subsection is divided into two parts: an overview of predictive power management (Section 3.2.1) and a description of PaViz (Section 3.2.2).

3.2.1. Overview of Predictive Power Management

Predictive power management strategies use performance models to estimate the execution time of each node based on the application configuration. Depending on the application, creating an accurate performance model can be a large challenge, and in some cases, may not be possible.

Using a mathematical formula, the performance estimates for each node are translated into a power assignment to account for the imbalance at each visualization cycle. Nodes with lower performance estimates (when compared to the estimates of other nodes) will have lower power assignments, since they will likely finish quickly and sit idle. This enables nodes with higher performance estimates to be assigned more power, since they have more work to finish and are on the critical path.

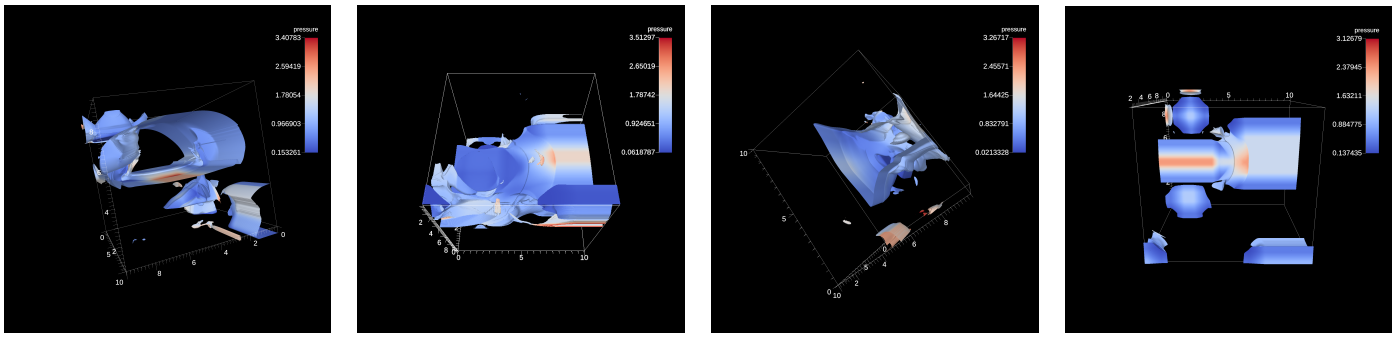
PaViz evaluated multiple formulas for translating the performance estimates into a power assignment [6]. The best performing formula used the difference from the fastest predicted execution time to determine the power allocation. Using the per-node predicted execution times, the power assignment is computed as follows:

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

where pow_{node_min} is the hardware-specified minimum node power needed for reliability, ren_{min} is the global minimum predicted execution time among all n MPI tasks, ren_i is the predicted 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 execution 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 overall performance is only as fast as the slowest processor.

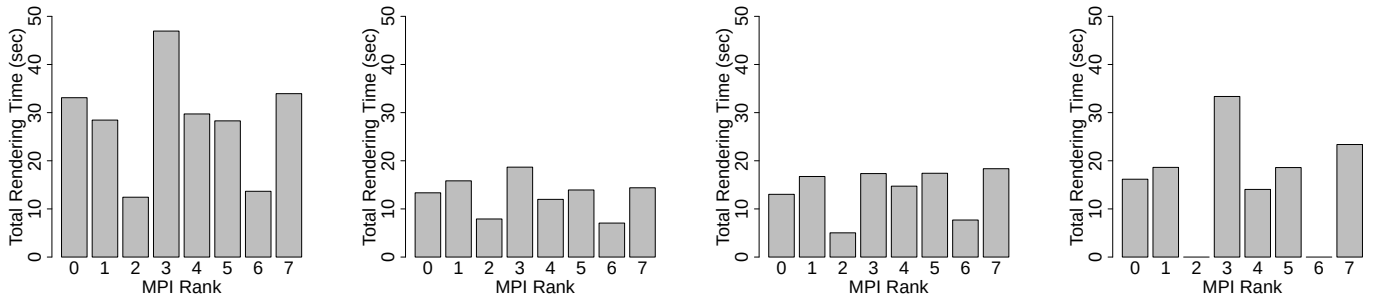
3.2.2. Predictive Runtime System: PaViz

PaViz [6] uses prediction to dynamically allocate power across nodes in a job. The predictions are based on a rendering performance model (described later in this section) that is integrated into the PaViz runtime system. If the performance model predicts a long rendering time due to a high volume of work, then PaViz allocates more power to that node. Alternatively, if the performance model predicts a short rendering time due to less work being assigned, then PaViz allocates less power to that node.



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

Figure 2: Ray traced images of the CloverLeaf hydrodynamics mini-application at the 200th simulation cycle for each rendering workload. Each figure shows a contour of the pressure at various values after it has expanded from the position where the energy was initially deposited.



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

Figure 3: A small example highlighting the variation present in the rendering workload. 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.

With respect to the rendering performance model, PaViz incorporates some existing models by Larsen et al. [40]. These models apply to different scientific visualization rendering workloads and are semi-empirical in nature, meaning that they are based both on algorithmic characteristics and observed execution behaviors on specific hardware architectures. For our study, we followed their guidance on producing a model appropriate for our hardware. Finally, the models take as inputs the camera position and the data set size to make predictions, and an extensive validation study showed the models to be highly accurate even when considering these two factors alone.

4. Experimental Overview

The following subsections detail the experimental setup and methodology.

4.1. Study Parameters

This study was designed to evaluate two power scheduling techniques (adaptive and predictive) under a variety of rendering workloads. We varied the following parameters in order to explore a representative set of configurations:

- Job Power Budgets (9 options)
- Rendering Workloads (4 options)
- MPI Tasks (5 options)
- Power Scheduling Strategies (2 options)

We ran the cross product of the aforementioned parameters totaling 360 tests configurations. For brevity, we report the results for 104 tests in this paper to highlight the key findings. The results are reported as an average over many trials to account for performance variability. Each of the parameters are discussed in the following subsections.

4.1.1. Job Power Budgets

Both GEOPM and PaViz assume the same specified job power budget. To simplify this study, both runtime systems ignore the power usage of the DRAM and other components within the node (*e.g.,* accelerators, network interface controllers), and are only concerned with the package power (this includes core and uncore). A more advanced approach is to use the node-level power budget to steer power across the components within the node (not just the package). However, this approach is

Power Cap (Per-Package)	**% TDP**
120W	100%
100W	83.3%
90W	75%
80W	66.7%
70W	58.3%
68W	56.7%
60W	50%
50W	41.7%
40W	33.3%

Table 1: Enumerating the package-level power budgets that can be requested by each runtime system. The range of available power budgets is specified by the hardware vendor. In this case, the package (core and uncore) has a maximum power draw of 120W (thermal design point or TDP) and a minimum power draw of 40W (for reliability purposes). Scaling the power budgets by the number of nodes will determine the total job power budget. The right column shows the percentage of each power cap out of TDP. The highest power cap of 120W is 100% of TDP. Lower percentages indicate a more severe power cap.

highly complex for two reasons. First, the DRAM power measurements are not well validated and are unstable [41], making it difficult to accurately determine total node power budgets. Second, other components within the node may have power measurements, but at different time scales and with different error bars, making it even more difficult to reach a precise node power usage. A more holistic node-level power budget is actively being defined in the community.

In order for an overprovisioned system to be successful, the system power budget must never be exceeded, otherwise leading to catastrophic failures considering these systems may use several MegaWatts of power. Within a supercomputer, job schedulers are being augmented to shift individual job power budgets throughout the system, while runtime systems can adjust power of its compute nodes within a job. The power cap impacts the speed (*i.e.,* operating frequency) of the processor. Enforcing a lower power cap will slow down the process running on that processor, while a higher power cap will speed up the process' execution. The goal of applying different power caps to different processes is such that all processes finish simultaneously, reducing idle time spent wasting power. For those processors with a wide range of available power caps (such as the one used in this study), this increases the granularity at which performance improvements can be made. A narrower range of power caps will minimize the potential benefits of this approach.

Table 1 shows the range of available power caps as specified by the processor vendor. When we identify the power caps in Section 5, we are referring to the package-level power caps. At the highest power cap of 120W, there is essentially unlimited power available for the application, since the application likely does not consume 120W. That is to say, the application's performance is not impacted by the 120W power cap, and should have the fastest runtime. We evaluate the performance at this power cap, so we can evaluate the performance of our runtime systems to the optimum performance of the application.

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

Table 2: Selected rendering workloads for this study. The data set size is per rank. The total data set size is derived by multiplying this value by the cube root of the number of ranks. 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 for a total of 6 visualization cycles. $IFact$ is a quantitative representation of the work imbalance, derived by taking the maximum predicted render time over the median of all predicted times.

4.1.2. Rendering Workload

We selected four rendering 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 2 and an image of the resulting contour is shown in Figure 2. These configurations span commonly used values for each parameter, and each workload exhibits a different amount of work imbalance. Figure 3 shows the imbalance in the total rendering times across all visualization cycles per rank.

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. However, with increasing I/O limitations of upcoming supercomputers, we are seeing new methods that reduce the amount of data saved to disk. One strategy for mitigating the I/O gap is rendering hundreds to thousands of images per timestep of the resulting analysis (*e.g.,* Cinema [36]) and save them to disk, which is significantly smaller than the original data set.

Our rendering infrastructure used Cinema-based in situ [37, 36], 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.

4.1.3. MPI Tasks

To study the scaling behaviors of adaptive and predictive strategies at higher concurrencies, we varied the number of MPI tasks. Experimenting with our framework at scale impacts the level of work imbalance in the simulation. The adaptive and predictive strategies improve performance by exploiting this imbalance. The intuition is that at higher concurrency, there is a bigger work imbalance per node because there are more ranks across which the input data can be decomposed, and likely more and more ranks will be assigned no data. For those ranks with

no work to do, we can shift power away from those ranks to those ranks on the critical path.

We used a single MPI rank per node (OpenMP threaded) to maximize the node memory allocation. The number of ranks used lends itself to a constant data set size per rank. The number of ranks used were 8, 125, 216, 343, and 512, corresponding to 288, 4,500, 7,776, 12,348, and 18,432 cores, respectively. These numbers are all powers of 3, which allowed the Cloverleaf simulation code to evenly decompose the computational mesh across MPI ranks. While Cloverleaf can run with powers of 2, its decomposition of the computational mesh varies based on concurrency, which has the potential to introduce performance quirks across concurrency. Powers of 3 decompose uniformly across concurrency, avoiding these quirks.

Finally, many of our experiments used only 8 MPI tasks, which is smaller than a typical in situ use case. One benefit of using 8 MPI tasks was a cost savings (compute cycles incurred), but our main motivation was to illustrate underlying phenomena in a simpler environment before moving up to higher concurrencies. Further, we feel these 8 node runs could still be valuable, in particular in the context of ensembles.

4.1.4. Power Scheduling Strategies

To evaluate the adaptive power management strategy, we used the Power Balancer agent included in GEOPM for our tests. For this study, we used GEOPM v1.0.0. A more detailed description of GEOPM is in Section 3.1.2.

To evaluate the predictive strategy, we used PaViz, which includes multiple algorithms for assigning power based on performance predictions [6]. Specifically, we used the *Min* algorithm, since it resulted in the best performance of all the algorithms explored. See Section 3.2.2 for a more detailed description of PaViz.

4.2. Software Infrastructure

We used VTK-m v1.2.0 and Ascent v0.4.0 to provide visualization and analysis capabilities. VTK-m [42] 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 [43], a library of visualization algorithms that serves as the basis for VisIt [33] and ParaView [34].

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 [44]. It is designed to support other in situ visualization tools, such as VisIt's LibSim [45] and ParaView's Catalyst [46]. Ascent depends on VTK-m for inter-node parallelism and OpenMP for intra-node parallelism. Ascent's framework includes three proxy simulations — Kripke [47], Lulesh [48], and CloverLeaf [49, 50]. For the scientific simulation in this study, we used CloverLeaf, a hydrodynamics proxy application on a three-dimensional structured grid.

4.3. Hardware Architecture

We conduct our experiments on the Quartz Intel E5-2695 (Broadwell) supercomputer at Lawrence Livermore National Laboratory, running the current major version of TOSS [51], TOSS 3. 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 [52] kernel module enables power monitoring and control from user space. We enforce a package-level power limit using Intel's Running Average Power Limit (RAPL) interfaces [15], which impacts both the core and the uncore. Under a more severe power limit, the package (*i.e.,* 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 package-level power cap ranging from 120W down to 40W.

GEOPM Package Power Decisions

N \ C	0	1	2	3	4	5
0	60W	60W	60W	55W	45W	43W
1	60W	60W	60W	54W	45W	49W
2	60W	60W	60W	47W	52W	46W
3	60W	60W	60W	69W	76W	81W
4	60W	60W	60W	61W	63W	65W
5	60W	60W	60W	55W	47W	51W
6	60W	60W	60W	54W	53W	53W
7	60W	60W	60W	61W	55W	57W

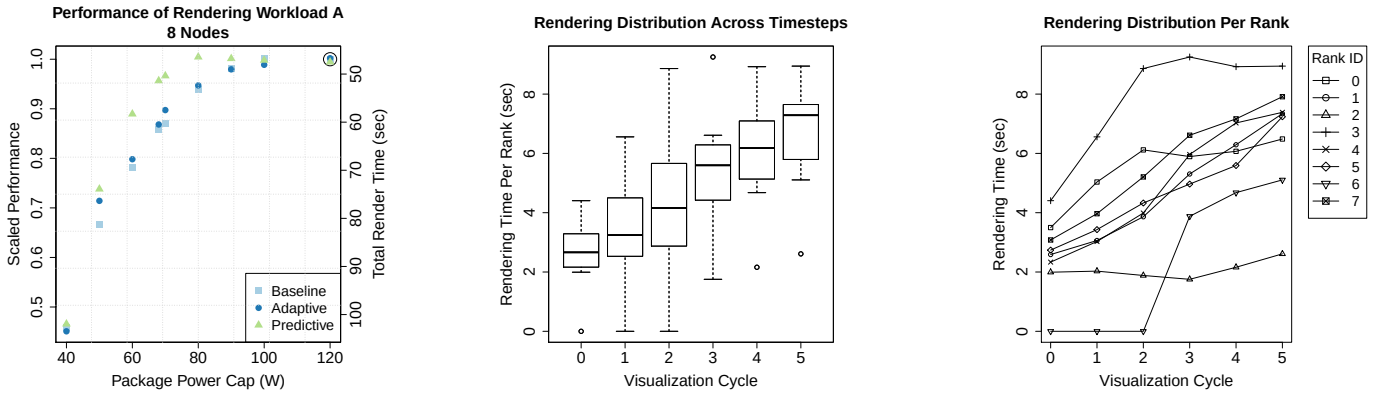
PaViz Package Power Decisions

N \ C	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 3: Comparing package power limits determined by GEOPM and PaViz for Rendering Workload A on 8 nodes. For each visualization cycle (denoted by *C*), the runtime uses the power scheduler to make decisions on what the power limits should be for each node (denoted by *N*). The job power limit assumes a package-level power cap of 60W. We see that as the visualization cycle increases, the adaptive scheduler takes a couple cycles to start adjusting the power limits relative to how the application work is distributed. The predictive scheduler is able to make its adjustments at each visualization cycle, and react appropriately.

5. Results

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



(a) Scaled Performance

(b) Workload Distribution Across Ranks

(c) Workload Distribution Per Rank

Figure 4: 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 running on 8 nodes. We normalize the performance to that of the baseline at 120W package power cap (circled in black). The baseline applies a uniform power cap across all packages within the node. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The middle figure shows the distribution of rendering execution times (*i.e.,* work) for all ranks at each visualization cycle, while the right figure shows the rendering times per rank and per 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.

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 package, which is the maximum power draw for this processor architecture. The uniform power distribution strategy is currently used in practice. Since each node on the target supercomputer is dual-socket, the node power allocation is derived by taking the sum of the power caps assigned to both packages. Each node may be assigned a different power cap by the GEOPM or PaViz runtime system, but the aggregate sum total of the node 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.

5.1. Base Case

In this phase, we studied Rendering Workload A, running on 8 nodes. We sweep over all package power caps, ranging from 120W down to 40W, and compare the performance of the adaptive and predictive strategies. The left figure in Figure 4 shows the scaled performance for three strategies, baseline, adaptive, and predictive. The baseline strategy applies a uniform power budget across all packages, the adaptive strategy uses GEOPM's Power Balancer, and the predictive strategy uses PaViz. The performance of all three scheduling strategies is compared to the baseline at 120W package power cap. Some of the performance values may be slightly higher than 1 due to system noise. The middle figure shows the distribution of 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 becomes more variable. The specifics of how the rendering time can change across simulation cycles within a rank can be seen in the rightmost figure.

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

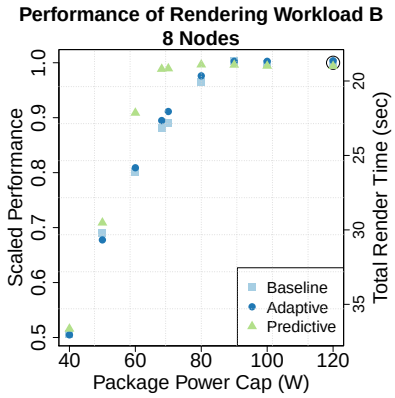
For power caps that range between 80W and 50W, we start to see the differences in using adaptation versus prediction on the highly irregular visualization workload. This is the range where the application is consuming all the available power and benefits from shifting power intelligently.

At a package power cap of 60W, we compare the power decisions made by the adaptive and predictive strategy across all visualization cycles in Table 3. 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. Since the visualization workload is highly variable from cycle to cycle, GEOPM's power decisions are unable to react to the variable rendering time for a given rank, leaving performance on the table. 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.

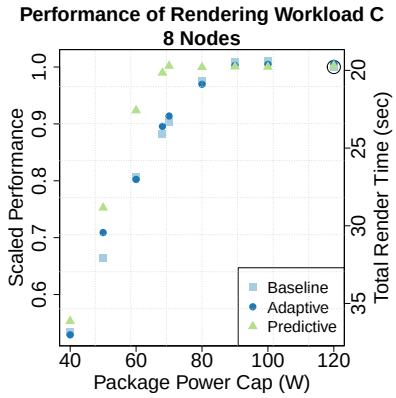
The distribution of render times in Figure 3 shows rank 3 being assigned the most work for this workload configuration, so it is expected that both strategies will assign it the highest power cap. In doing so, the adaptive and predictive strategies reduce overall execution time and perform better than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

5.2. Vary Rendering Workloads

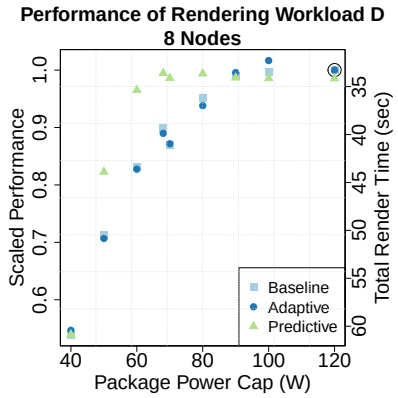
In this set of tests, we vary across the remaining three rendering workload configurations outlined in Table 2, using 8



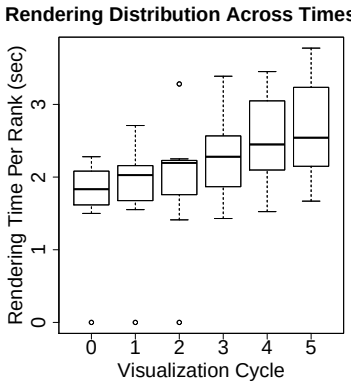
(a) Scaled Performance, Rendering Workload B



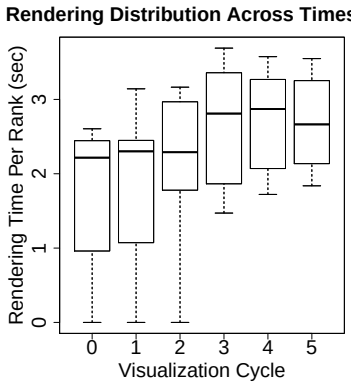
(b) Scaled Performance, Rendering Workload C



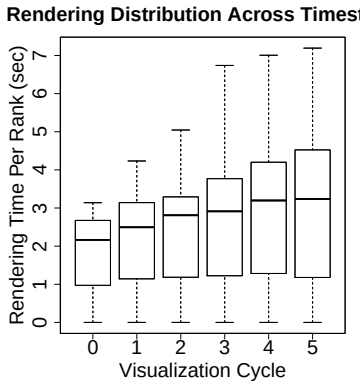
(c) Scaled Performance, Rendering Workload D



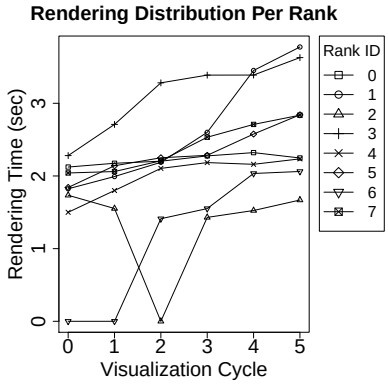
(d) Workload Distribution Across Ranks, Rendering Workload B



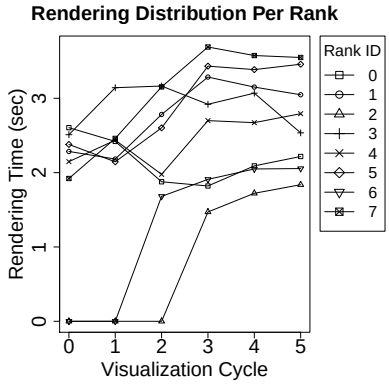
(e) Workload Distribution Across Ranks, Rendering Workload C



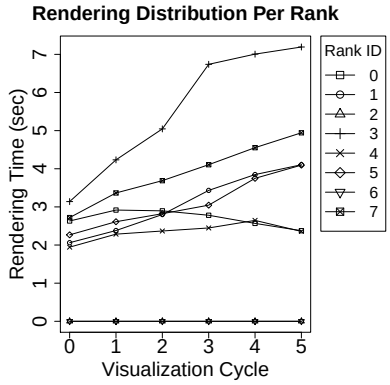
(f) Workload Distribution Across Ranks, Rendering Workload D



(g) Workload Distribution Per Rank, Rendering Workload B



(h) Workload Distribution Per Rank, Rendering Workload C



(i) Workload Distribution Per Rank, Rendering Workload D

Figure 5: The top row of figures shows the scaled performance of the adaptive strategy in GEOPM's Power Balancer and the predictive strategy in PaViz to the baseline for Rendering Workloads B, C, and D on 8 nodes. 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. We denote this normalized value with an open black circle. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. The middle row of figures shows the distribution of rendering execution times (*i.e.,* work) for all ranks at each visualization cycle, while the bottom row of figures shows the rendering times per rank and per 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 bottom row of figures highlights how the rendering time can vary for each rank across simulation cycles.

nodes, the same node concurrency as in Section 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 5 compares 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 subfigures can be compared with Figure 4, which did the same analysis for Rendering Workload A. Compared to the previous Rendering Workload A, the distribution of execution times is 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.

5.3. 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 bigger job power budget that can be reallocated across nodes. For this phase, we report the results for Rendering Workload C, which showed the best improvement in the previous phase. The geometry generated by the isovalue in this workload are sparsely distributed across nodes. As the simulation advances and becomes more complex, the isovalue geometry leads to a larger imbalance. This characteristic provided the best opportunity for performance improvements. We sweep over package 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 6 shows the scaled performance for the adaptive strategy in GEOPM and the predictive strategy in PaViz at different levels of concurrency.

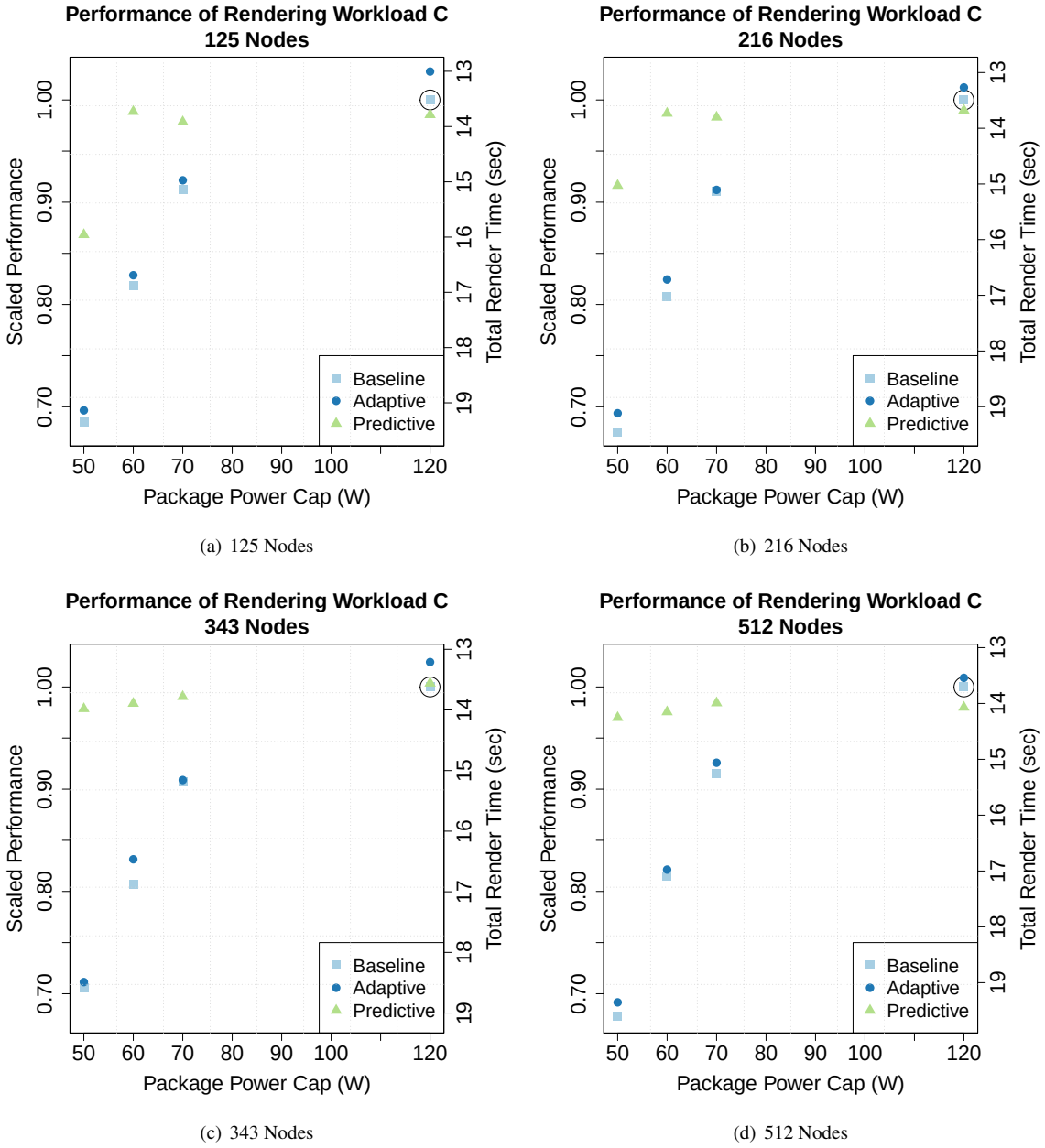
For this rendering workload, using a predictive strategy results in 27% improvement over an adaptive strategy. As the concurrency increases, an increasing percentage of the nodes have very little, or even no geometry to render. Figure 7 shows the difference in scaled performance between PaViz and GEOPM. The smallest differences in speedups occur at the highest package 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 package power cap. That is to say, the difference in scaled performance 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 27% at the lowest package power cap of 50W. The predictive strategy uses its performance model to quickly identify the nodes with no work to do. For those nodes with nothing to do, it sets the lowest power cap, enabling some nodes with lots of work to do to run with the highest power cap per package, and complete their work as fast as possible. Due to the high variability in this workload, the adaptive strategy is slower to adjust the power usage based on the level of imbalance across the nodes. As a result, the performance of the adaptive strategy is worse than the predictive strategy.

6. Conclusion and Future Work

The hypothesis of our study was that prediction would outperform adaptation for visualization workloads due to their irregular nature. Our experiments confirmed this hypothesis, showing improvements up to 27%. We found that the level of concurrency and the level of imbalance in the rendering workload had a large impact on the potential speedup of using a predictive or adaptive strategy. The larger the concurrency, the more imbalanced the workload may become (more ranks have no work to do), freeing up more power that can be shifted to bottleneck ranks in order to improve the overall performance. Prediction performed well on workloads with a high variation in execution time across visualization cycles, since we had an accurate performance model. While for other workloads where the variation across cycles was small, we saw a narrower margin between the two approaches, since the adaptive approach benefited from similar execution behaviors in the previous time step. In all, we feel the main takeaway from this study is that the predictive approach should be considered for power-aware run-time systems when faced with irregular workloads.

In terms of future work, we plan to pursue multiple directions. A major hurdle for the predictive approach is the availability of models. In our case, we were able to use an existing performance model which was very accurate. For other irregular workloads, such performance models would need to be generated, constituting an important area of future work. Further, having such performance models would enable evaluation of the efficacy of the predictive approach in more contexts (*i.e.,* beyond visualization). There are many such contexts, including physics simulations, their individual components (*e.g.,* sparse linear algebra), and even general HPC workloads beyond physics simulations. Another area of future work is to create frameworks that can do both predictive and adaptive strategies, so that adopters are not forced to choose between one or the other (*e.g.,* use GEOPM or PaViz) and are not forced to do extra work (*e.g.,* use both). We hope to release PaViz as open source software in the near future, to assist in this goal, and also to assist in reproducibility for future research. Finally, this work focused on power allocation and also considered in situ visualization as a motivating use case. Outside this setting, traditional approaches such as dynamic data redistribution would make for interesting comparisons.



(a) 125 Nodes

(b) 216 Nodes

(c) 343 Nodes

(d) 512 Nodes

Figure 6: 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. We denote the normalized value with an open black circle. The second y-axis shows the raw rendering times at each power cap, since the scaled performance value does not provide this context. Instead of sweeping over all power caps, we focus on caps between 50W and 70W, which were regions of interest in previous phases.

7. Acknowledgements

This document was prepared as an account of work sponsored by an agency of the United States government. 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. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-782698-DRAFT).

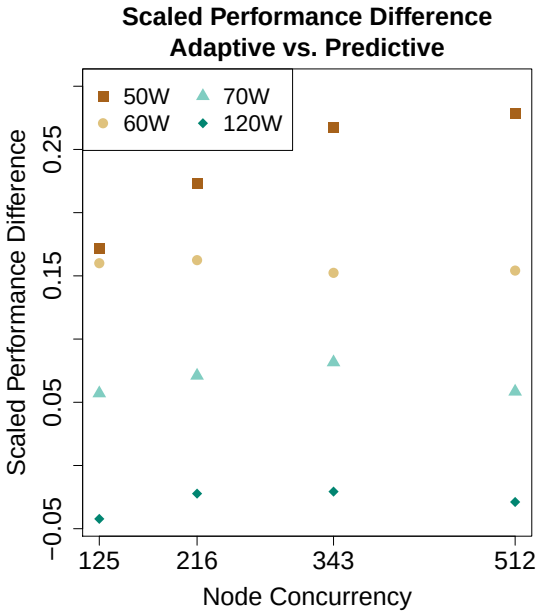


Figure 7: Difference in speedup for Rendering Workload C using the adaptive and predictive strategies at different node concurrencies. The performance at each concurrency is normalized to the performance at 120W power cap.

References

[1] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, B. 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, Association for Computing Machinery, New York, NY, USA, 2013, p. 173182.

[2] N. Gholkar, F. Mueller, B. Rountree, Power Tuning HPC Jobs on Power-Constrained Systems, in: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), 2016, pp. 179–190.

[3] D. A. Ellsworth, A. D. Malony, B. Rountree, M. Schulz, Dynamic Power Sharing for Higher Job Throughput, in: SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2015, pp. 1–11.

[4] D. A. Ellsworth, A. D. Malony, B. Rountree, M. Schulz, POW: System-Wide Dynamic Reallocation of Limited Power in HPC, in: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 15, Association for Computing Machinery, New York, NY, USA, 2015, p. 145148.

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

[6] S. Labasan, M. Larsen, H. Childs, B. Rountree, PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance, in: A. Telea, J. Bennett (Eds.), Eurographics Symposium on Parallel Graphics and Visualization, The Eurographics Association, 2017.

[7] geopm, `https://github.com/geopm/geopm` (2016).

[8] J. Eastep, S. Sylvester, C. Cantalupo, B. Geltz, F. Ardanaz, A. Al-Rawi, K. Livingston, F. Keceli, M. Maiterth, S. Jana, Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions, in: J. M. Kunkel, R. Yokota, P. Balaji, D. Keyes (Eds.), High Performance Computing, Springer International Publishing, Cham, 2017, pp. 394–412.

[9] A. C. Bauer, H. Abbasi, J. Ahrens, H. Childs, B. Geveci, S. Klasky, K. Moreland, P. O'Leary, V. Vishwanath, B. Whitlock, E. W. Bethel, In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms, Computer Graphics Forum (CGF) 35 (3) (2016) 577–597.

[10] H. Childs, J. Bennett, C. Garth, B. Hentschel, In Situ Visualization for Computational Science, IEEE Computer Graphics and Applications (CG&A) 39 (6) (2019) 76–85.

[11] T. Peterka, D. Bard, J. C. Bennett, E. W. Bethel, R. A. Oldfield, L. Pouchard, C. Sweeney, M. Wolf, Priority research directions for in situ data management: Enabling scientific discovery from diverse data sources, The International Journal of High Performance Computing Applications.

[12] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, T. Bletsch, Adagio: Making DVS Practical for Complex HPC Applications, in: Proceedings of the 23rd International Conference on Supercomputing, ICS '09, ACM, New York, NY, USA, 2009, pp. 460–469.

[13] V. W. Freeh, N. Kappiah, D. K. Lowenthal, T. K. Bletsch, Just-in-time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs, J. Parallel Distrib. Comput. 68 (9) (2008) 1175–1185.

[14] R. Ge, X. Feng, W. chun Feng, K. Cameron, CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters, in: Parallel Processing, 2007. ICPP 2007. International Conference on, 2007, pp. 18–18.

[15] I. Corporation, Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3: System Programming Guide (June 2017).

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

[17] IBM, IBM EnergyScale for POWER8 Processor-Based Systems (November 2015).

[18] NVIDIA, NVML Reference Manual (May 2015).

[19] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, B. R. Supinski, High Performance Computing: 30th International Conference, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings, Springer International Publishing, Cham, 2015, Ch. A Run-Time System for Power-Constrained HPC Applications, pp. 394–408.

[20] K. W. Cameron, R. Ge, X. Feng, High-Performance, Power-Aware Distributed Computing for Scientific Applications, Computer 38 (11) (2005) 4047.

[21] R. Ge, Xizhou Feng, K. W. Cameron, Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, in: SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 2005, pp. 34–34.

[22] C.-H. Hsu, U. Kremer, The Design, Implementation, and Evaluation of a Compiler Algorithm or CPU Energy Reduction, in: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, PLDI 03, Association for Computing Machinery, New York, NY, USA, 2003, p. 3848.

[23] Chung-hsing Hsu, Wu-chun Feng, A Power-Aware Run-Time System for High-Performance Computing, in: SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, 2005, pp. 1–1.

[24] M. Endrei, C. Jin, M. N. Dinh, D. Abramson, H. Poxon, L. DeRose, B. R. de Supinski, Energy efficiency modeling of parallel applications, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 18, IEEE Press, 2018.

[25] J. Arjona Aroca, A. Chatzipapas, A. Fernández Anta, V. Mancuso, A measurement-based analysis of the energy consumption of data center servers, in: Proceedings of the 5th International Conference on Future Energy Systems, e-Energy 14, Association for Computing Machinery, New York, NY, USA, 2014, p. 6374.

[26] A. Verma, G. Dasgupta, T. K. Nayak, P. De, R. Kothari, Server workload analysis for power minimization using consolidation, in: Proceedings of the 2009 Conference on USENIX Annual Technical Conference, USENIX09, USENIX Association, USA, 2009, p. 28.

[27] E. Pinheiro, R. Bianchini, E. Carrera, T. Heath, Load balancing and unbalancing for power and performance in cluster-based systems, 2001.

[28] Z. Cao, S. Dong, Energy-aware framework for virtual machine consolidation in cloud computing, in: 2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013, pp. 1890–1895.

[29] V. Adhinarayanan, W. chun Feng, J. Woodring, D. Rogers, J. Ahrens, On the Greenness of In-Situ and Post-Processing Visualization Pipelines, in: Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International, 2015, pp. 880–887.

[30] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, P. T. Bremer, Evaluation of In-Situ Analysis Strategies at Scale for Power Effi-

ciency and Scalability, in: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 156–164.

[31] S. Labasan, M. Larsen, H. Childs, Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm, in: Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on, 2015, pp. 73–80.

[32] M. Gamell, I. Rodero, M. Parashar, J. C. Bennett, H. Kolla, J. Chen, P.-T. Bremer, A. G. Landge, A. Gyulassy, P. McCormick, S. Pakin, V. Pascucci, S. 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, ACM, New York, NY, USA, 2013, pp. 77:1–77:12.

[33] H. Childs, E. Brugger, B. Whitlock, J. Meredith, S. Ahern, D. Pugmire, K. Biagas, M. Miller, C. Harrison, G. H. Weber, H. Krishnan, T. Fogal, A. Sanderson, C. Garth, E. W. Bethel, D. Camp, O. Rübel, M. Durant, J. M. Favre, P. Navrátil, VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data, in: High Performance Visualization–Enabling Extreme-Scale Scientific Insight, 2012, pp. 357–372.

[34] J. Ahrens, B. Geveci, C. Law, ParaView: An End-User Tool for Large Data Visualization.

[35] S. Ahern, et al., Scientific Discovery at the Exascale: Report for the DOE ASCR Workshop on Exascale Data Management, Analysis, and Visualization (July 2011).

[36] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, M. 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, IEEE Press, Piscataway, NJ, USA, 2014, pp. 424–434.

[37] P. O'Leary, J. Ahrens, S. Jourdain, S. Wittenburg, D. H. Rogers, M. Petersen, Cinema Image-Based In Situ Analysis and Visualization of MPAS-ocean Simulations, Parallel Computing 55 (2016) 43–48.

[38] J. Bigler, J. Guilkey, C. Gribble, C. Hansen, S. G. Parker, A Case Study: Visualizing Material Point Method Data, in: Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization, EUROVIS'06, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2006, pp. 299–306.

[39] S. Parker, P. Shirley, Y. Livnat, C. Hansen, P.-P. Sloan, Interactive Ray Tracing for Isosurface Rendering, in: Proceedings of the Conference on Visualization '98, VIS '98, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 233–238.

[40] M. Larsen, C. Harrison, J. Kress, D. Pugmire, J. Meredith, H. Childs, Performance Modeling of In Situ Rendering, in: The International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, 2016.

[41] S. Desrochers, C. Paradis, V. M. Weaver, A validation of dram rapl power measurements, in: Proceedings of the Second International Symposium on Memory Systems, MEMSYS 16, Association for Computing Machinery, New York, NY, USA, 2016, p. 455470.

[42] K. Moreland, et al., VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures, IEEE Computer Graphics and Applications 36 (3) (2016) 48–58.

[43] W. J. Schroeder, K. M. Martin, W. 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, IEEE Computer Society Press, 1996, pp. 93–ff.

[44] M. Larsen, J. Ahrens, U. Ayachit, E. Brugger, H. Childs, B. Geveci, C. 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, ACM, New York, NY, USA, 2017, pp. 42–46.

[45] B. Whitlock, J. M. Favre, J. 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, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2011, pp. 101–109.

[46] U. Ayachit, A. Bauer, B. Geveci, P. O'Leary, K. Moreland, N. Fabian, J. 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, ACM, New York, NY, USA, 2015, pp. 25–29.

[47] A. Kunen, T. Bailey, P. Brown, KRIPKE-A Massively Parallel Transport Mini-App, Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2015).

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

[49] CloverLeaf, `http://uk-mac.github.io/CloverLeaf/` (2017).

[50] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, S. A. Jarvis, Cloverleaf: Preparing Hydrodynamics Codes for Exascale, The Cray User Group (2013) 6–9.

[51] Toss: Speeding up commodity cluster computing, `https://computing.llnl.gov/projects/toss-speeding-commodity-cluster-computing`.

[52] msr-safe, `https://github.com/llnl/msr-safe` (2016).