Eurographics Symposium on Parallel Graphics and Visualization (2017)
J. C. Bennett, A. Telea (Editors)

# PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance

Stephanie Labasan[†][1,2], Matthew Larsen[2], Hank Childs[1], and Barry Rountree[2]

[1]University of Oregon Eugene, Oregon, United States
[2]Lawrence Livermore National Laboratory, Livermore, CA, United States

**Abstract**

*Power consumption is widely regarded as one of the biggest challenges to reaching the next generation of high-performance computing. One strategy for achieving an exaflop given limited power is hardware overprovisioning. In this model, the theoretical peak power usage of the system is greater than the maximum allowable power usage, and a central manager keeps the aggregate power usage at the maximum by enforcing power caps on each node in the system. For this model to be effective, the central manager must be able to make informed trade-offs between power usage and performance. With this work, we introduce PaViz, a software framework designed to optimize the distribution of power for visualization algorithms, which have different characteristics than simulation codes. In this study, we focus specifically on rendering. Our strategy uses a performance model, where nodes predicted to have a small amount of work are allocated less power, and nodes predicted to have a large amount of work are allocated more power. This approach increases the likelihood of all nodes finishing at the same time, which is optimal for power efficiency. At best, our adaptive strategy achieves up to 33% speedup over the traditional strategy, while using the same total power.*

Categories and Subject Descriptors (according to ACM CCS): D.1.3 [Software]: Concurrent Programming—Parallel Programming I.3.2 [Computer Graphics]: Graphics Systems—Distributed/Network Graphics

## 1. Introduction

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 2. The main contribution of this paper 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 paper is organized as follows. We present an overview of power with respect to HPC in Section 2. The related work is detailed in Section 3. The contributions of the PaViz framework and power scheduling strategies are discussed in Section 4. Section 5 identifies the study parameters. We evaluate the benefits of PaViz in Section 6. In Section 7, we summarize our findings and present some ideas for future exploration in this space.

## 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 [PLR*13], short for hardware overprovision-

[†] slabasan@cs.uoregon.edu

DOI: 10.2312/pgv.20171088

ing, 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 [ZLPF14].

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 sub-optimal. 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 [PLS*15]. Alternatively, there are dynamic optimizations to individual jobs that may be realized by rebalancing power and changing node configuration at runtime [geo16, ESC*16, MBL*15]. 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.

## 3. Related Work

In the following subsections, we survey related work.

### 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 [Lev90] 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 [CBW*12] and ParaView [AGL05]) and packages from hardware vendors (*e.g.,* Intel's OSPRay [WJA*17] and NVIDIA's IndeX [Ind]). For our study, we chose volumetric ray casting because of its widespread use, and also because of its existing performance model [LHK*16].

### 3.2. Power

Some of the earliest solutions in addressing energy use in HPC were CPU MISER [GFcFC07], Jitter [FKLB08], and Adagio [RLdS*09]. 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) [Cor16], AMD TDP PowerCap [Dev13], and IBM EnergyScale [IBM15]) enable more recent efforts to focus on optimizing performance under a power bound. Conductor [MBL*15] used initial iterations to determine an ideal schedule of per-node power caps, thread concurrency, and per-core operating frequency. GEOPM [geo16,ESC*16] 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.

## 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 [CBW*12] or ParaView [AGL05]. 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 [AcFW*15,RPL*16]. 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. [LLC15] 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. [G*13] also looked at the power-performance trade-offs of various parameters at scale.

## 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 [LHK*16]. 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.

| 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 1:** *Power scheduling strategy parameters.*

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. 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 [LHK*16]. 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.

### 4.1. 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).

#### 4.1.1. *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.

#### 4.1.2. *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.

#### 4.1.3. *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.

#### 4.1.4. *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.

#### 4.1.5. *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.

## 5. Study Overview

The following section provides an overview of the methodology.

### 5.1. Software Infrastructure

For our software infrastructure, we used Strawman [LBC*15], an open-source in situ framework containing three physics proxy applications. Of these three proxy applications, we used Cloverleaf3D [MBG*13, clo17], a hydrodynamics mini-app on a three-dimensional structured grid. Strawman also includes a rendering infrastructure, which combines node-level rendering using VTK-m [M*16], configured with Intel's Thread Building Blocks [Rei07], and distributed memory image compositing using IceT [MKPH11]. For our study, we integrated PaViz into Strawman and added infrastructure to calculate per node rendering estimates based on the performance model.

### 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 [msr16]. Using Intel's Running Average Power Limit (RAPL) technology [Cor16], 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.

### 5.3. Study Parameters

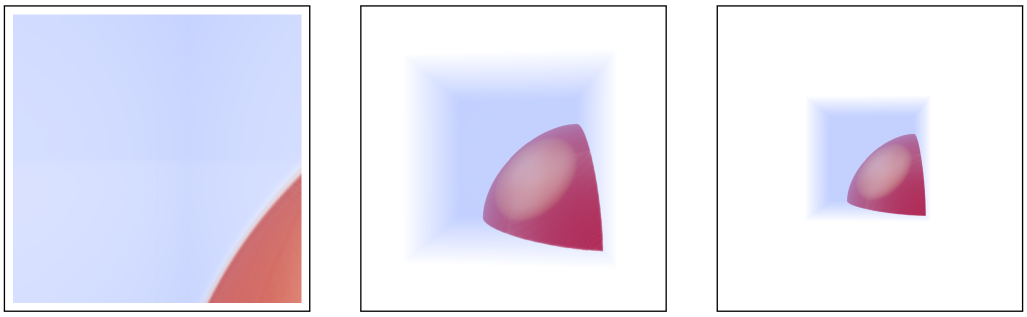
We varied the following parameters as part of this study:

- 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.

#### 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



Rendering Workloads A and C (camera positioned inside the data set) — Rendering Workload B (camera positioned near the data set) — Rendering Workload D (camera positioned far from the data set)

**Figure 1:** *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.*

| 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 2:** *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.*

task. The configurations used in this study are enumerated in Table 2. Images of the data 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 1.

### 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 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.

### 5.3.3. Power Scheduling Strategy

We explore the performance improvements of the five power scheduling strategies defined in Section 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 rebalancing power based on a performance model, we wanted to explore the benefits of using different power scheduling strategies.

### 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.

## 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.

### 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 [AJO*14], where hundreds of images are rendered per time step. This use case increases overall render time and amortizes out the RAPL delay.

#### 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.

## 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 4.1. 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 5.3. In each phase, we vary the job power budget, and analyzed its impact.

### 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 2) and are shown in Figure 2.

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.

### 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 3). 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.

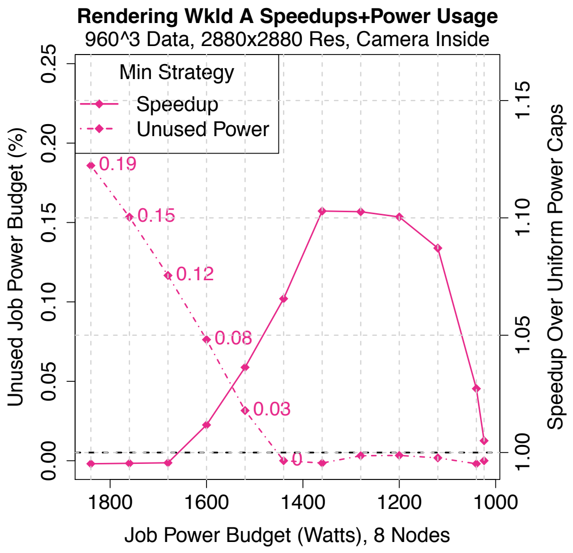


**Figure 2:** *Speedups and allocated power for Rendering Workload A using the Min power scheduling strategy. The solid line shows the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup is 1, indicating no change in performance. The 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.*

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.

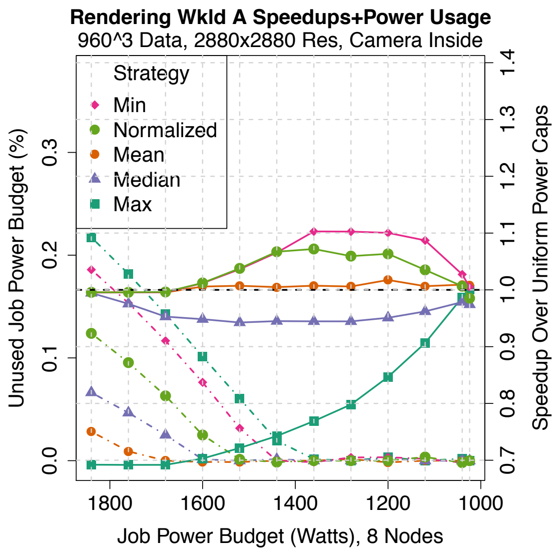


**Figure 3:** *Comparing speedups and unused job power budget for Rendering Workload A across all five power scheduling strategies.*

## 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 4 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 3. 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.

## 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 5 shows the speedups and unused job power for all rendering configurations enumerated in Table 2 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.

## 7. Conclusion and Future Work

With this work, 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. We accomplished this by incorporating prediction, 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 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.

## Acknowledgements

LLNL-CONF-727082. 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

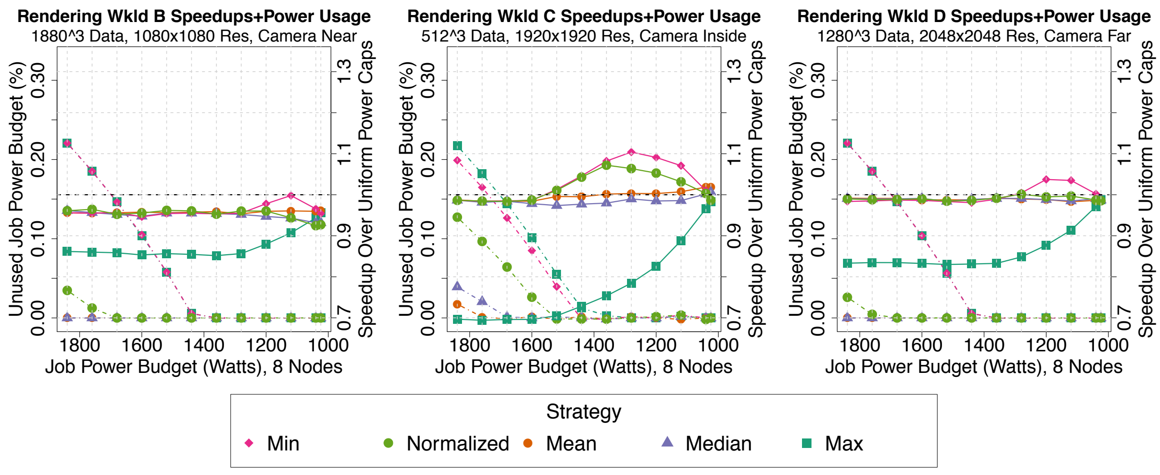


**Figure 4:** *Comparing speedups and unused job power for Rendering Workloads B, C, and D at 8 node concurrency using all five power scheduling strategies. The solid lines show the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup 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).*

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 by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-689101).

Hank Childs is grateful for support from the DOE Early Career Award, Contract No. DE-SC0010652, Program Manager Lucy Nowell.

## References

[AcFW*15] Adhinarayanan V., chun Feng W., Woodring J., Rogers D., Ahrens J.: On the Greenness of In-Situ and Post-Processing Visualization Pipelines. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International* (May 2015), pp. 880–887. doi:10.1109/IPDPSW.2015.132. 3

[AGL05] Ahrens J., Geveci B., Law C.: ParaView: An End-User Tool for Large Data Visualization. 2, 3

[AJO*14] Ahrens J., Jourdain S., O'Leary P., Patchett J., Rogers D. H., Petersen M.: 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* (Piscataway, NJ, USA, 2014), SC '14, IEEE Press, pp. 424–434. URL: https://doi.org/10.1109/SC.2014.40, doi:10.1109/SC.2014.40. 5

[CBW*12] Childs H., Brugger E., Whitlock B., Meredith J., Ahern S., Pugmire D., Biagas K., Miller M., Harrison C., Weber G. H., Krishnan H., Fogal T., Sanderson A., Garth C., Bethel E. W., Camp D., Rübel O., Durant M., Favre J. M., Navrátil P.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. Oct 2012, pp. 357–372. 2, 3

[clo17] Cloverleaf. http://uk-mac.github.io/CloverLeaf/, 2017. 4

[Cor16] Corporation I.: *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3: System Programming Guide*, June 2016. 2, 4

[Dev13] Devices A. M.: *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*, January 2013. 2

[ESC*16] Eastep J., Sylvester S., Cantalupo C., Ardanaz F., Geltz B., Al-Rawi A., Keceli F., Livingston K.: Global extensible open power manager: A vehicle for hpc community collaboration toward co-designed energy management solutions. In *Proceedings of the 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems* (Piscataway, NJ, USA, 2016), PMBS '16, IEEE Press. URL: https://doi.org/10.1109/PMBS.2016.6, doi:10.1109/PMBS.2016.6. 2

[FKLB08] Freeh V. W., Kappiah N., Lowenthal D. K., Bletsch T. K.: Just-in-time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs. *J. Parallel Distrib. Comput. 68*, 9 (Sept. 2008), 1175–1185. URL: http://dx.doi.org/10.1016/j.jpdc.2008.04.007, doi:10.1016/j.jpdc.2008.04.007. 2

[G*13] Gamell M., et al.: 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* (New York, NY, USA, 2013), SC '13, ACM, pp. 77:1–77:12. URL: http://doi.acm.org/10.1145/2503210.2503303, doi:10.1145/2503210.2503303. 3

[geo16] geopm. https://github.com/geopm/geopm, 2016. 2

[GFcFC07] Ge R., Feng X., chun Feng W., Cameron K.: CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters. In *Parallel Processing, 2007. ICPP 2007. International Con-*

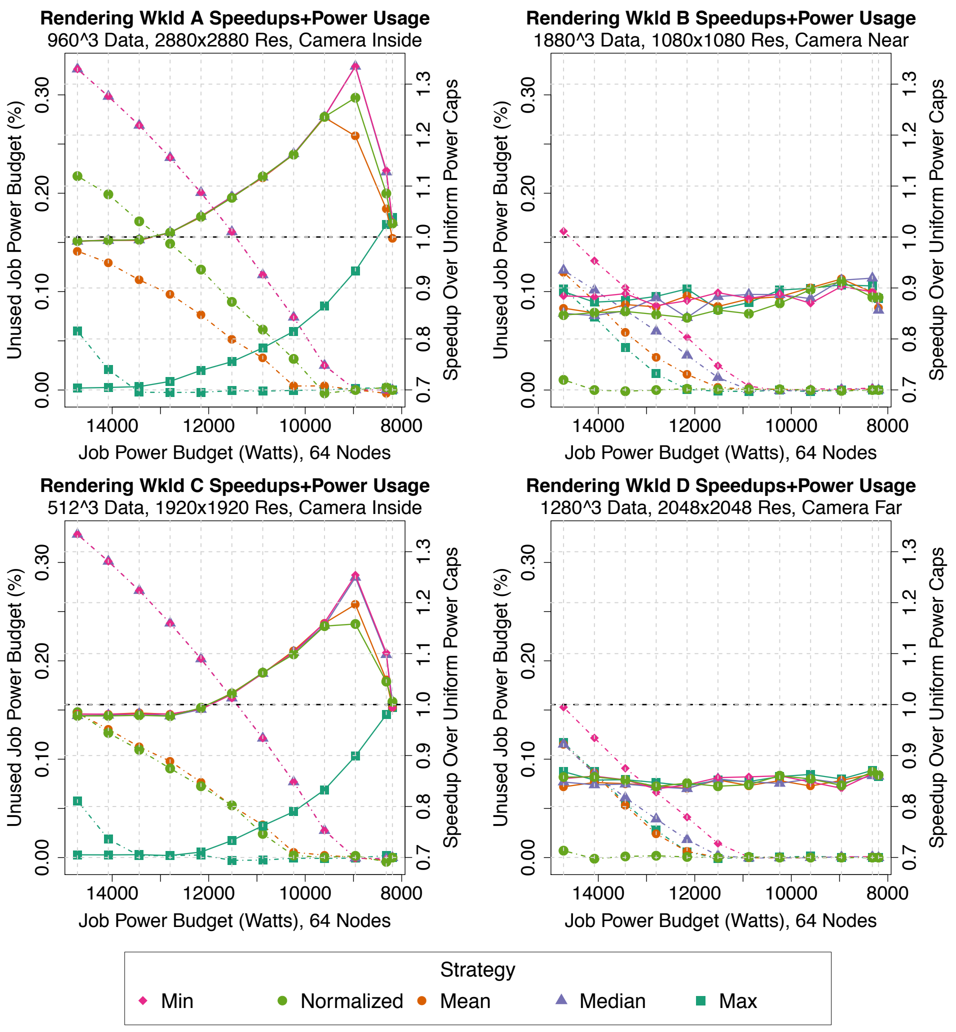


**Figure 5:** *Comparing speedups and unused job power for all rendering workloads at 64 node concurrency using the five power scheduling strategies. The solid lines show the resulting speedups as compared to uniform power caps (right y-axis). The black dotted line identifies where the speedup is 1, indicating no change in performance. The dotted lines show the percentage of unused job power that resulted in a particular speedup (left y-axis).*

*ference on* (Sept 2007), pp. 18–18. doi:10.1109/ICPP.2007.29. 2

[IBM15] IBM: *IBM EnergyScale for POWER8 Processor-Based Systems*, November 2015. 2

[Ind] Nvidia index. https://developer.nvidia.com/index. Accessed: 2017-03-02. 2

[LBC*15] Larsen M., Brugger E., Childs H., Eliot J., Griffin K., Harrison C.: 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* (2015), ACM, pp. 30–35. 4

[Lev90] Levoy M.: Efficient ray tracing of volume data. *ACM Transactions on Graphics (TOG) 9*, 3 (1990), 245–261. 2

[LHK*16] Larsen M., Harrison C., Kress J., Pugmire D., Meredith J., Childs H.: Performance Modeling of In Situ Rendering. In *The International Conference for High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, UT, Nov. 2016). 2, 3

[LLC15] Labasan S., Larsen M., Childs H.: Exploring Tradeoffs Between Power and Performance for a Scientific Visualization Algorithm. In *Large Data Analysis and Visualization (LDAV), 2015 IEEE 5th Symposium on* (Oct 2015), pp. 73–80. doi:10.1109/LDAV.2015.7348074. 3

[M*16] Moreland K., et al.: Vtk-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications 36*, 3 (May 2016), 48–58. doi:10.1109/MCG.2016.48. 4

[MBG*13] Mallinson A., Beckingsale D. A., Gaudin W., Herdman J., Levesque J., Jarvis S. A.: Cloverleaf: Preparing hydrodynamics codes for exascale. *The Cray User Group* (2013), 6–9. 4

[MBL*15] Marathe A., Bailey P. E., Lowenthal D. K., Rountree B., Schulz M., Supinski B. R.: *High Performance Com-*

*puting: 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. URL: http://dx.doi.org/10.1007/978-3-319-20119-1_28, doi:10.1007/978-3-319-20119-1_28. 2

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An Image Compositing Solution at Scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011), ACM, p. 25. 4

[msr16] msr-safe. https://github.com/llnl/msr-safe, 2016. 4

[PLR*13] PATKI T., LOWENTHAL D. K., ROUNTREE B., SCHULZ M., DE SUPINSKI B. R.: Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing* (New York, NY, USA, 2013), ICS '13, ACM, pp. 173–182. URL: http://doi.acm.org/10.1145/2464996.2465009, doi:10.1145/2464996.2465009. 1

[PLS*15] PATKI T., LOWENTHAL D. K., SASIDHARAN A., MAITERTH M., ROUNTREE B. L., SCHULZ M., DE SUPINSKI B. R.: Practical Resource Management in Power-Constrained, High Performance Computing. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing* (New York, NY, USA, 2015), HPDC '15, ACM, pp. 121–132. URL: http://doi.acm.org/10.1145/2749246.2749262, doi:10.1145/2749246.2749262. 2

[Rei07] REINDERS J.: *Intel Threading Building Blocks*, first ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007. 4

[RLdS*09] ROUNTREE B., LOWENTHAL D. K., DE SUPINSKI B. R., SCHULZ M., FREEH V. W., BLETSCH T.: Adagio: Making DVS Practical for Complex HPC Applications. In *Proceedings of the 23rd International Conference on Supercomputing* (New York, NY, USA, 2009), ICS '09, ACM, pp. 460–469. URL: http://doi.acm.org/10.1145/1542275.1542340, doi:10.1145/1542275.1542340. 2

[RPL*16] RODERO I., PARASHAR M., LANDGE A. G., KUMAR S., PASCUCCI V., BREMER P. T.: 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)* (May 2016), pp. 156–164. doi:10.1109/CCGrid.2016.95. 3

[WJA*17] WALD I., JOHNSON G., AMSTUTZ J., BROWNLEE C., KNOLL A., JEFFERS J., GÜNTHER J., NAVRATIL P.: Ospray-a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (2017), 931–940. 2

[ZLPF14] ZHANG Z., LANG M., PAKIN S., FU S.: Trapped Capacity: Scheduling under a Power Cap to Maximize Machine-Room Throughput. In *Energy Efficient Supercomputing Workshop (E2SC), 2014* (Nov 2014), pp. 41–50. doi:10.1109/E2SC.2014.10. 2