Power and Performance Tradeoffs for Visualization Algorithms

Stephanie Labasan*†, Matthew Larsen*, Hank Childs†, and Barry Rountree*
*Lawrence Livermore National Lab, Livermore, CA, USA, Email: {labasan1,larsen30,rountree}@llnl.gov
†University of Oregon, Eugene, OR, USA, Email: hank@uoregon.edu

***Abstract*—One of the biggest challenges for leading-edge supercomputers is power usage. Looking forward, power is expected to become an increasingly limited resource, so it is critical to understand the runtime behaviors of applications in this constrained environment in order to use power wisely. Within this context, we explore the tradeoffs between power and performance specifically for visualization algorithms. Visualization algorithms are unique in their execution behaviors under a power limit, since they are more data intensive than traditional HPC applications, like scientific simulations. This data intensive characteristic lends itself to alternative strategies regarding power usage. In this study, we focus on a representative set of visualization algorithms, and explore their power and performance characteristics as a power bound is applied. The result is a study that identifies how future research efforts can exploit the execution characteristics of visualization applications in order to optimize performance under a power bound.**

***Index Terms*—Power/energy; scientific visualization; high performance computing**

I. Introduction

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

The premise of this research is that simulations and visualization routines (and other components of the HPC ecosystem) will operate in a power-limited environment (see Section III-A). The Tokyo Institute of Technology in Japan is one example of a facility that has deployed power-limited production systems [1]. Two of their systems — TSUBAME2 and TSUBAME3 — must share the facility-level power budget (*i.e.,* inter-system power capping). Additionally, due to extreme heat during the summer months, the resource manager may dynamically turn off nodes to stay under a specified power cap.

At exascale, simulations and visualization routines are expected to run simultaneously (*i.e., in situ*) due to a decrease in the percentage of writable FLOPS. Further, power-limited environments will greatly impact the overall time-to-solution. Efforts to optimize performance under a power bound has typically focused on traditional HPC workloads rather than visualization, which can be a significant portion of the overall execution time. Additionally, visualization applications are more data intensive than traditional HPC workloads.

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

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

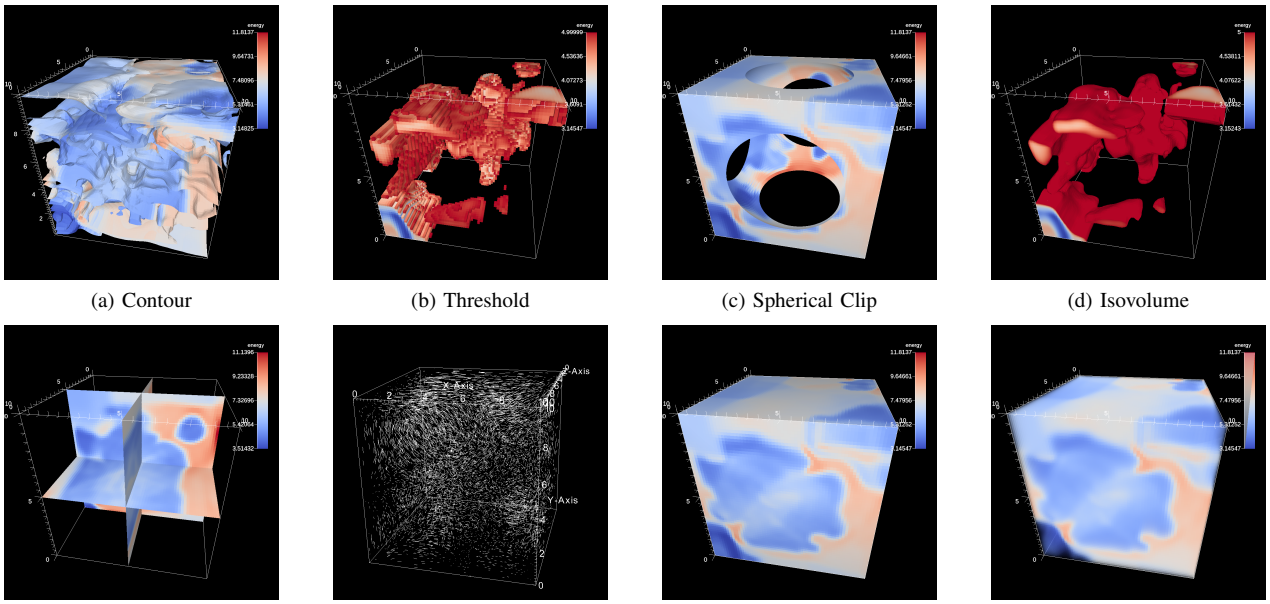
We selected eight common visualization algorithms, which we believe are representative of the execution behaviors of the hundreds of existing visualization algorithms. We also selected four data set sizes and varied the processor-level power cap to understand how the changes affect power and performance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

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

The rest of this paper is organized as follows. Section II discusses previous work. Section III provides an overview of power in HPC and the algorithms explored. The details of the experimental setup and methodology are presented in Section IV. We define the metrics and variables used in Section V. Results are discussed in Section VI. We summarize our findings in Section VII and identify ideas for future work in Section VIII.

II. Related Work

Relatively few works have explored the power and performance tradeoffs for visualization algorithms on supercomputers. One prominent work by Gamell et al. [2] investigated



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

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

Fig. 1. Renderings of the eight visualization algorithms explored in this study. We believe this set of algorithms is representative of the execution behaviors of the hundreds of existing visualization algorithms. The images show the energy field at the 200th time step of the CloverLeaf hydrodynamics proxy application.

the relationship between power and performance for in situ data visualization and analytics at large scale. One of the benefits of moving from a traditional post hoc visualization workflow to an in situ workflow is to mitigate the costs and overheads due to data movement of large volumes of data and I/O bandwidth. As such, other works investigate the power costs of storing data for subsequent visualization operations, in particular focusing on how data is moved through the storage hierarchy [3], [4]. There has also been work on incorporating performance prediction of visualization algorithms (*e.g.,* rendering) into dynamically reallocating power in a multi-node job [5].

The most relevant prior work to this study comes from Labasan et al. [6], where the authors studied a single visualization algorithm (isosurfacing) and considered explicit setting of the CPU frequency (which is less favorable for managing power usage on exascale systems than more recent power capping technologies such as Intel's Running Average Power Limit (RAPL) [7], AMD's TDP PowerCap [8], and IBM's EnergyScale [9]). In our current study, we consider eight algorithms — chosen to be representative of most visualization algorithms — and use the more current technique of power capping. Therefore, while the initial study [6] showed that a visualization algorithm has unique power and performance tradeoffs, the current study is considerably more comprehensive and also more relevant to exascale computing (*i.e.,* power capping versus setting CPU frequencies). Further, this study contains a series of findings that allow us to extrapolate behavior to other visualization algorithms.

III. Overview of Power and Algorithms

A. HPC Power Overview

Today's supercomputers are designed assuming every node in the system will run at their thermal design power (TDP) simultaneously. However, very few power-hungry applications are capable of consuming this theoretical maximum (due to bottlenecks), and most applications only consume 60% of peak [10]. Thus, designing supercomputers as if most applications consume peak power wastes power capacity and limits computational capacity (*i.e.,* nodes).

One solution to increase power utilization (and decrease trapped capacity [11]) is to design a hardware overprovisioned system (overprovisioned, for short), where more nodes are procured than can be fully powered simultaneously [12]–[14]. An adequately overprovisioned system will not exceed the system-wide power bound by implementing strategies to limit the power usage of the nodes. A naïve strategy is to apply a uniform power cap to all nodes. The effect of applying a power cap is that the CPU operating frequency is reduced. The effects of reducing the CPU operating frequency will vary across applications. Those dominated by compute instructions will slow down proportionally, while those dominated by memory accesses may be unaffected.

Uniform power capping across all nodes in the system has two limitations when considering a distributed application. First, such a strategy does not adapt to applications containing non-uniform workload distribution across nodes (causing static or dynamic computational imbalances). Further, uniform power caps translate to variations in performance across otherwise identical processors due to processor manufacturing variations [15].

Given a distributed application, nodes with lots of work to do (or less efficient nodes) determine the overall performance of the application, while nodes with little work to do (or more efficient nodes) finish early and sit idle until the other nodes have completed execution. A better strategy to optimize performance is to assign power to the nodes where it is needed most.

One of the key challenges with overprovisioning is understanding how different applications will behave under a power cap. With this study, we focus specifically on scientific visualization algorithms, which merit special attention since they behave differently (*i.e.,* more memory-bound) than traditional HPC applications, like simulations. Our findings begin to inform performance model input parameters in predicting the work distribution for scientific visualization applications.

B. Overview of Visualization Algorithms

We explored eight algorithms for this study. We believe this set of algorithms is representative of the behaviors and characteristics commonly found across all visualization algorithms. We provide a brief description of each of the eight algorithms in the following subsections (see Fig. 1 for a rendered image of each algorithm).

1) Contour: For a three-dimensional scalar volume, the output of a contour is a surface representing points of a constant value (*i.e.,* isovalue). For this study, the data set consisted of hexahedrons and the algorithm used was Marching Cubes [16]. The contour algorithm iterates over each cell in the data set, identifying cells that contain the constant value. The algorithm uses pre-computed lookup tables in combination with interpolation to generate triangles that represent the surface, and the resulting geometry is combined into the output data set. We used 10 different isovalues for a single visualization cycle.

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

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

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

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

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

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

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

IV. Experimental Overview

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

A. Software Framework

Our software infrastructure included VTK-m and Ascent. VTK-m [17] is an open-source library of scientific visualization algorithms designed for shared-memory parallelism. Its algorithms are implemented using a layer of abstraction enabling portable performance across different architectures. It is an extension of the Visualization ToolKit (VTK) [18], a well-established open-source library of visualization algorithms that form the basis of VisIt [19] and ParaView [20]. For this study, we configured VTK-m with Intel's Thread Building Blocks (TBB) [21] for thread-level parallelism.

The Ascent [22], [23] in situ framework is part of the multi-institutional project known as ALPINE. Ascent is a flyweight,

open-source in situ visualization framework designed to support VisIt's LibSim [24] and ParaView's Catalyst [25]. Of the three included multi-physics proxy applications, we used CloverLeaf [26], [27], a hydrodynamics simulation, tightly coupled with the visualization. By tightly coupled, we mean the simulation and visualization alternate while using the same resources.

B. Hardware Architecture

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

On LLNL systems, the `msr-safe` [28] driver provides an interface for sampling and controlling processor power usage, among other performance counters, via 64-bit model-specific registers. On this Broadwell processor, the power can be capped from 120W (TDP) down to 40W using Intel's Running Average Power Limit technology (RAPL) [7]. Then, the processor adjusts the operating frequency to guarantee the desired power cap.

C. Study Factors

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

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

D. Methodology

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

1) Phase 1: Processor-Level Power Cap: Phase 1 varied the processor-level power caps and studied the behavior of the contour algorithm implemented in VTK-m. With this phase, we extended a previous finding [6], which determined baseline performance for isosurfacing by explicitly setting CPU frequencies. This phase consisted of nine tests.
Test Configuration: (Contour algorithm, 128^3 data set size) $\times$ 9 processor power caps

2) Phase 2: Visualization Algorithm: In this phase, we continued varying processor-level power caps, and added variation in visualization algorithm. It consisted of 72 tests, nine of which were studied in Phase 1.
Test Configuration: (128^3 data set size) $\times$ 9 processor power caps $\times$ 8 visualization algorithms

3) Phase 3: Data Set Size: In this phase, we add variation in data set size. It consisted of 288 tests, of which nine were studied in Phase 1 and 63 were studied in Phase 2.
Test Configuration: 9 processor power caps $\times$ 8 visualization algorithms $\times$ 4 data set sizes

V. Definition of Metrics

In this section, we define the variables and metrics that will be used in the following results section.

A. Abstract Case

Let's assume a visualization algorithm takes T_D seconds to run at the default power (*i.e.,* thermal design power, or TDP) of P_D Watts. As the power cap is reduced, the same visualization algorithm now takes T_R seconds to run with a power cap of P_R Watts. We will use the following derived terms to explain our results:

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

Something to note, P_{ratio} and F_{ratio} have the default value in the numerator and the reduced value in the denominator, while T_{ratio} has them reversed. Inverting the ratio simplifies our comparisons, making all ratios be greater than 1.

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

B. Performance Measurements

To collect power usage information, we sample the energy usage of each processor in the node every 100 ms throughout the application (*i.e.,* simulation and visualization) execution. We derive the power usage for each processor by dividing

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

TABLE I

THE SLOWDOWN FOR THE CONTOUR ALGORITHM AS THE PROCESSOR POWER CAP IS REDUCED. THE CONFIGURATION USED FOR THIS ALGORITHM IS A DATA SET SIZE OF 128^3. P IS THE ENFORCED PROCESSOR POWER CAP. T IS THE TOTAL EXECUTION TIME IN SECONDS FOR THE CONTOUR ALGORITHM OVER ALL VISUALIZATION CYCLES. F IS THE EFFECTIVE CPU FREQUENCY GIVEN THE POWER CAP P. A 10% SLOWDOWN (DENOTED IN RED) DOES NOT OCCUR FOR THIS ALGORITHM UNTIL THE LOWEST POWER CAP.

the energy usage (contained in a 64-bit register) by the elapsed time between samples. In addition to energy and power counters, we also sample fixed counters, frequency-related counters, and two programmable counters — last level cache misses and references. From these counters, we can derive the following metrics. We show the derivation of these metrics using the Intel-specific performance counter event names [29], where applicable.

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

C. Efficiency Metric

We leverage a rate in terms of the size of the input (*i.e.,* data set size) rather than speedup for comparing the efficiency of one visualization algorithm to another. If the speedup of a parallel algorithm is defined as $\frac{T_{n,1}}{T_{n,p}}$, then one must know the serial execution time of the algorithm. This is challenging with increasingly complex simulations running at higher concurrency levels. Instead, we assess speedup using a rate originally proposed by Moreland and Oldfield [30], [31]. They express the rate in terms of the data set size, n, as follows: $\frac{n}{T_{n,p}}$.

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

VI. RESULTS

In this section, we describe the results from the phases detailed in Section IV-D.

A. Phase 1: Processor-Level Power Cap

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

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

Once the power cap is reduced by a factor of 3X (from 120W down to 40W), we see a change in the execution time and CPU frequency by a factor of 1.17X and 1.23X, respectively. At 40W, the algorithm takes longer to run (since the frequency is also reduced to maintain the desired power usage), but the algorithm did not slowdown proportionally to the reduction in power by a factor of 3. This confirms our finding in [6], where we determined that the contour algorithm was sufficiently data intensive to avoid slowing down proportional to the CPU clock frequency.

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

B. Phase 2: Visualization Algorithm

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

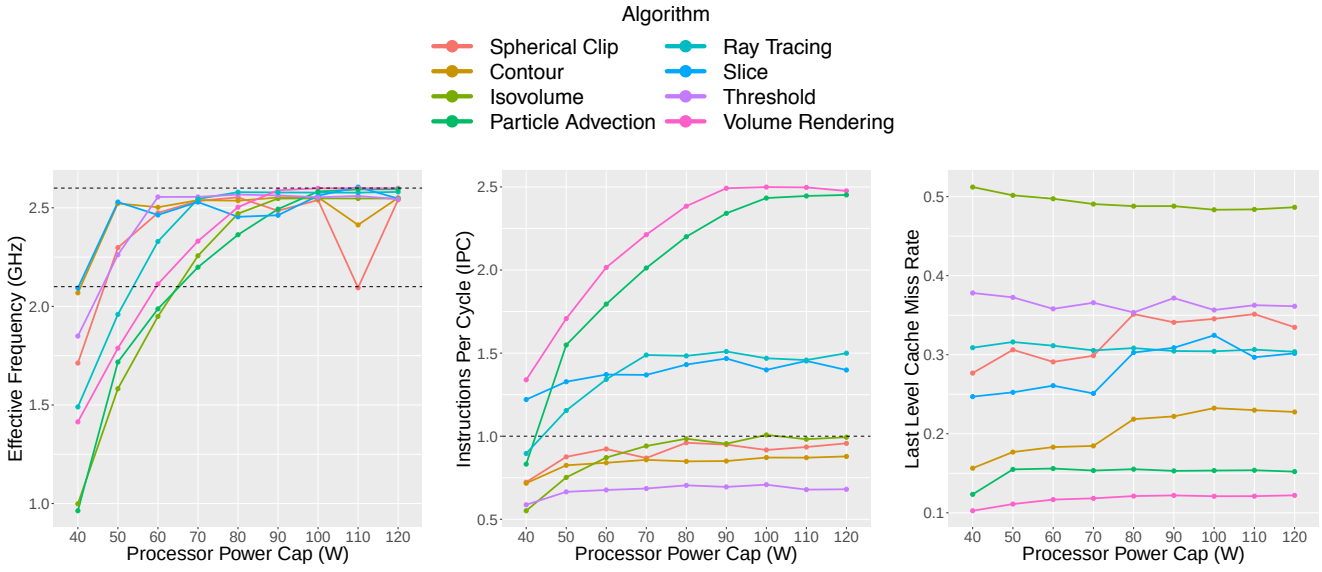
1) Power Opportunity Algorithms: The algorithms that fall into the power opportunity category are contour (discussed in the previous section), spherical clip, isovolume, threshold, slice, and ray tracing. Table II shows the slowdown in execution time and CPU frequency for all algorithms. The power opportunity algorithms do not see a significant slowdown (of 10%, denoted in red) until P_{ratio} is at least 2X or higher. These algorithms are data-bound — the bottleneck is the memory subsystem, not the processor — so reducing the power cap does not significantly impact the overall performance. This is confirmed since T_{ratio} is less than P_{ratio}.

When looking at the CPU operating frequency in Fig. 2a, we see that all algorithms, regardless of whether it is in the

	P	120W	110W	100W	90W	80W	70W	60W	50W	40W
	P_{ratio}	1.0X	1.1X	1.2X	1.3X	1.5X	1.7X	2.0X	2.4X	3.0X
Contour	T_{ratio}	1.00X	1.00X	1.00X	1.00X	1.00X	0.91X	0.91X	0.93X	1.17X
	F_{ratio}	1.00X	1.06X	1.00X	1.00X	1.01X	1.00X	1.02X	1.01X	1.23X
Spherical Clip	T_{ratio}	1.00X	1.01X	1.03X	1.02X	1.00X	1.05X	1.02X	1.18X	1.48X
	F_{ratio}	1.00X	1.21X	1.00X	1.02X	1.00X	1.00X	1.03X	1.11X	1.48X
Isovolume	T_{ratio}	1.00X	1.01X	0.99X	1.04X	1.02X	1.06X	1.14X	1.30X	1.81X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.03X	1.13X	1.31X	1.61X	2.55X
Threshold	T_{ratio}	1.00X	0.98X	0.98X	1.00X	0.99X	0.99X	1.02X	1.08X	1.31X
	F_{ratio}	1.00X	0.99X	1.00X	0.99X	0.99X	1.00X	1.00X	1.12X	1.38X
Slice	T_{ratio}	1.00X	0.98X	1.00X	0.99X	0.98X	1.02X	1.04X	1.03X	1.26X
	F_{ratio}	1.00X	0.98X	0.99X	1.03X	1.04X	1.01X	1.03X	1.01X	1.22X
Ray Tracing	T_{ratio}	1.00X	1.00X	0.99X	0.99X	1.00X	1.01X	1.10X	1.31X	1.75X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.00X	1.01X	1.11X	1.32X	1.73X
Particle Advection	T_{ratio}	1.00X	1.00X	1.01X	1.05X	1.11X	1.21X	1.34X	1.57X	3.12X
	F_{ratio}	1.00X	1.00X	1.00X	1.04X	1.10X	1.18X	1.31X	1.51X	2.69X
Volume Rendering	T_{ratio}	1.00X	1.00X	0.99X	1.00X	1.04X	1.12X	1.23X	1.46X	1.86X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.04X	1.12X	1.23X	1.45X	1.84X

TABLE II

SLOWDOWN FACTOR FOR ALL ALGORITHMS WITH A DATA SET SIZE OF 128^3. SLOWDOWN IS CALCULATED BY DIVIDING EXECUTION TIME AT 40W BY EXECUTION TIME AT 120W. NUMBERS HIGHLIGHTED IN RED INDICATE THE FIRST TIME A 10% SLOWDOWN IN EXECUTION TIME OR FREQUENCY OCCURS DUE TO THE PROCESSOR POWER CAP P.



(a) Effective frequency (b) IPC (c) Last level cache miss rate

Fig. 2. Effective frequency (GHz), instructions per cycle (IPC), and last level cache miss rate for all algorithms as the processor power cap is reduced. For each algorithm, we use a data set size of 128^3.

power opportunity or power sensitive class, run at the same frequency of 2.6 GHz at a 120W power cap, which is the maximum turbo frequency for this architecture when all cores are active. The differences across the algorithms are seen in the rate at which the frequency declines because of the enforced power cap and the power usage of the algorithms.

The default power usage varies across visualization algorithms, ranging from as low as 55W up to 90W per processor. For algorithms that do not consume TDP, the processor decides it can run in turbo mode (*i.e.,* above 2.1 GHz base clock frequency) to maximize performance. Once the power cap is at or below the power usage of the algorithm, the operating frequency begins to drop because the processor can no longer maintain a high frequency without exceeding the power cap. For algorithms with a high power usage, the frequency will start dropping at power caps close to TDP. For algorithms with a low power usage (*e.g.,* contour, described previously), the processor runs in turbo mode for most power caps to maximize performance. It is not until the lowest power cap of 40W that we see a reduction in the clock frequency for contour.

Fig. 2b shows the average instructions per cycle (IPC) for all algorithms. The dotted line drawn at an IPC of 1 shows the divide between compute-bound algorithms (IPC $>$ 1) and memory-bound algorithms (IPC $<$ 1). Spherical clip, contour, isovolume, and threshold make up one class of algorithms. Their IPC is characteristic of a data-bound algorithm, and their power usage is also very low, so the decrease in IPC is not seen until the lowest power cap of 40W. Threshold is dominated by loads and stores of the data, so it has a low IPC value. Contour and isovolume have higher IPC values (out of this

group of algorithms) because it calculates interpolations.

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

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

Fig. 2c shows the last level cache miss rate for all algorithms, and is the inverse of Fig. 2b. Isovolume has the highest last level cache miss rate, indicating that a high percentage of its instruction mix is memory-related. Because of the high miss rate, the isovolume algorithm spends a lot of time waiting for memory requests to be satisfied. Memory access instructions have a longer latency than compute instructions. Therefore, it cannot issue as many instructions per cycle, and has a low IPC.

Another interesting metric to investigate is shown in Fig. 3, which is the number of elements (in millions) processed per second. Because the power usage of these algorithms is low, the denominator (*e.g.,* seconds) stays constant for most power caps, yielding a near constant rate for each algorithm. At severe power caps, the number of elements processed per second declines because the algorithm incurs slowdown. Algorithms with very fast execution times will have a high rate, while algorithms with a longer execution time will have a low rate.

2) Power Sensitive Algorithms: The power sensitive algorithms are volume rendering and particle advection. They consume the most power at roughly 85W per processor. When the power cap drops below 85W, the frequency starts dropping as it can no longer maintain the desired power cap at the 2.6 GHz frequency. Thus, there are slowdowns of 10% at 70W and 80W, respectively, which is at a higher power cap than the power opportunity algorithms. These algorithms not only have the highest IPC values overall as shown in Fig. 2b (peak IPC of 2.68, highly compute-bound), but also have the biggest change in IPC as the power cap is reduced. Such algorithms are dominated by the CPU, so a reduction in power greatly

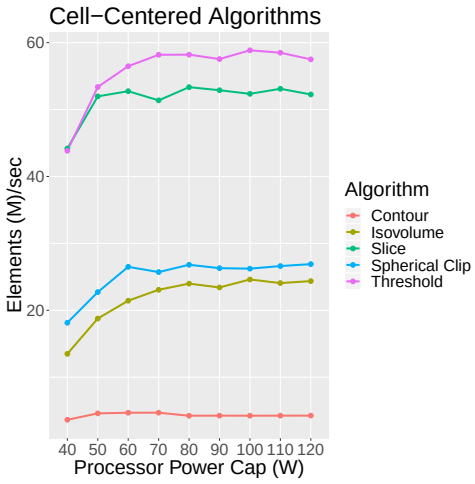


Fig. 3. Elements processed per second for cell-centered algorithms using 128^3 data set size.

impacts the number of cycles it takes to issue the same set of instructions (*i.e.,* slows down the algorithm).

Fig. 2b coupled with Fig. 2c shows volume rendering and particle advection with a high IPC because they have the lowest last level cache miss rate (*i.e.,* better memory performance). Additionally, more instructions can be retired per cycle because the processor is not stalled waiting on memory requests to be satisfied (*i.e.,* high IPC). Everything fit into cache, and IPC was changing drastically with changing power caps, so we can infer that IPC behavior was dominated by compute instructions.

3) Key Takeaways: For most of the algorithms explored in this paper, the power cap has little effect on performance. This is because the power usage of visualization algorithms is low compared to typical HPC applications. For similar algorithms, we can run them with the lowest power cap without impacting performance. In a larger scheme where we are running the simulation and visualization on the same resources, we can more intelligently allocate power between the two, rather than using a naïve scheme of evenly distributing the power. Said another way, we can allocate most of the power to the power-hungry simulation, leaving minimal power to the visualization, since it does not need it. Additionally, we find two of the algorithms explored (volume rendering and particle advection) have high power usage, consistent with typical HPC applications. These algorithms have a poor tradeoff between power and performance. There may be other algorithms that behave similarly.

C. Phase 3: Data Set Size

Phase 3 extended Phase 2 by varying over data set size. Table III shows the results for all algorithms using a data set size of 256^3. This table can be compared to Table II in Section VI-B.

As the data set size is increased from 128^3 in Table II to 256^3 in Table III, T_{ratio} changes across algorithms. For the

	P	120W	110W	100W	90W	80W	70W	60W	50W	40W
	P_{ratio}	1.0X	1.1X	1.2X	1.3X	1.5X	1.7X	2.0X	2.4X	3.0X
Contour	T_{ratio}	1.00X	1.00X	1.00X	1.00X	1.00X	1.00X	1.05X	1.19X	1.71X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.01X	0.99X	1.07X	1.18X	1.52X
Spherical Clip	T_{ratio}	1.00X	1.01X	1.01X	1.05X	1.01X	1.10X	1.16X	1.41X	2.13X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.01X	1.05X	1.17X	1.42X	1.95X
Isovolume	T_{ratio}	1.00X	0.98X	0.97X	1.01X	1.01X	1.01X	1.17X	1.33X	1.76X
	F_{ratio}	1.00X	1.00X	0.97X	1.00X	1.00X	1.05X	1.11X	1.32X	1.79X
Threshold	T_{ratio}	1.00X	1.02X	0.99X	0.99X	0.98X	1.09X	1.16X	1.30X	1.53X
	F_{ratio}	1.00X	1.01X	1.02X	1.02X	1.02X	1.05X	1.17X	1.38X	1.66X
Slice	T_{ratio}	1.00X	1.00X	0.99X	0.99X	1.00X	1.00X	0.99X	1.33X	1.69X
	F_{ratio}	1.00X	0.98X	1.01X	0.93X	1.01X	0.98X	1.01X	1.24X	1.44X
Ray Tracing	T_{ratio}	1.00X	1.00X	1.00X	1.01X	1.00X	1.02X	1.10X	1.28X	2.00X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.00X	1.01X	1.10X	1.29X	2.05X
Particle Advection	T_{ratio}	1.00X	1.00X	1.03X	1.07X	1.14X	1.39X	1.64X	2.13X	2.67X
	F_{ratio}	1.00X	1.00X	1.02X	1.06X	1.13X	1.35X	1.57X	2.05X	2.56X
Volume Rendering	T_{ratio}	1.00X	1.00X	1.00X	1.00X	1.06X	1.13X	1.24X	1.45X	1.81X
	F_{ratio}	1.00X	1.00X	1.00X	1.00X	1.06X	1.13X	1.23X	1.45X	1.82X

TABLE III

SLOWDOWN FACTOR FOR ALL ALGORITHMS WITH A DATA SET SIZE OF 256^3. SLOWDOWN IS CALCULATED BY DIVIDING EXECUTION TIME AT 40W BY EXECUTION TIME AT 120W. NUMBERS HIGHLIGHTED IN RED INDICATE THE FIRST TIME A 10% SLOWDOWN IN EXECUTION TIME OR FREQUENCY OCCURS DUE TO THE PROCESSOR POWER CAP P.

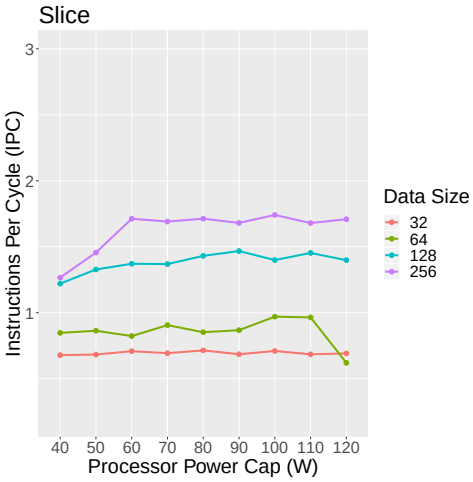


Fig. 4. This category of algorithms sees an increase in IPC as the data set size increases. Algorithms that fall into this category are slice, contour, isovolume, threshold, and spherical clip.

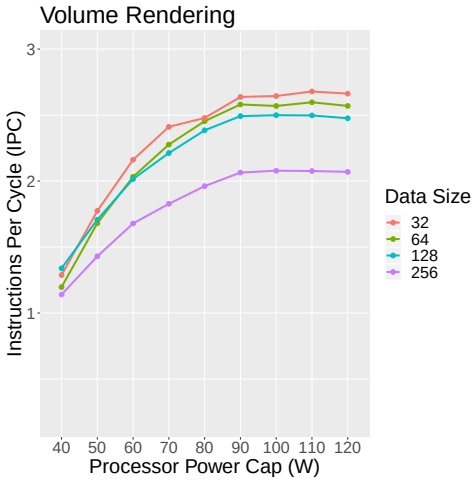


Fig. 5. This category of algorithms sees an increase in IPC as the data set size decreases. Volume rendering is the only algorithm exhibiting this behavior.

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

Depending on the algorithm, the IPC may increase or decrease as the data set size is increased. Fig. 4, Fig. 5, and Fig. 6 show the IPC for three different algorithms over all power caps and data set sizes. The IPC of the three different algorithms shown in the figures represent three categories.

The first category consists of slice, contour, isovolume, threshold, and spherical clip. As the data set size increases, the IPC also increases for these algorithms as shown in Fig. 4. Particularly for slice and spherical clip, the number

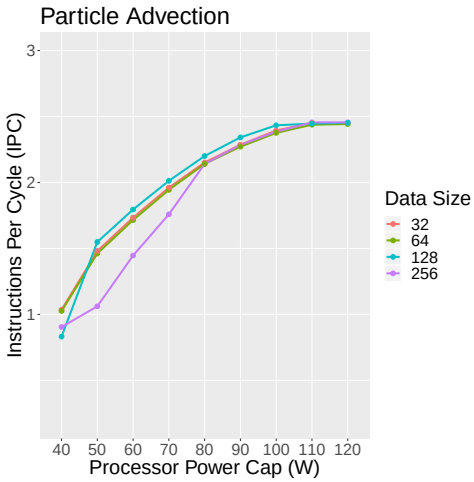


Fig. 6. This category of algorithms see no change in IPC as the data set size changes. Algorithms exhibiting this behavior are particle advection and ray tracing.

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

The second category contains volume rendering, which shows an inverse relationship between data set size and IPC as shown in Fig. 5. Here, the IPC increases as the data set size decreases. As an example, as the data set size increases from 128^3 to 256^3 (8X bigger), the IPC only drops by 20% going from 2.5 down to 2. On average, the IPC of volume rendering is higher than any of the other algorithms explored in this paper. Volume rendering is an image-order algorithm and has a high number of floating point instructions resulting in high power and high IPC.

The third category consists of algorithms whose IPC does not change with increases in data set sizes as illustrated in Fig. 6. The algorithms identified here are particle advection and ray tracing. For particle advection, we held the following constant regardless of the data set size: the same number of seed particles, step length, and number of steps. Because we chose to keep these parameters consistent, particles may get displaced outside the bounding box depending on the data set size. When particles are displaced outside the bounding box, they terminate, and there is no more work to do for that particle.

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

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

VII. Summary of Findings

One of the key goals of this paper was to identify the impacts of various factors on power usage and performance of visualization algorithms in order to better inform scientists and tool developers. We summarize the findings from the previous sections here.

On varying processor power caps (Section VI-A):

- The VTK-m implementation of contour is sufficiently data intensive to avoid a significant slowdown from reducing the power cap. This extends a previous finding [6] which set CPU frequencies and used a custom implementation, and is additionally noteworthy since our study uses a general toolkit designed to support a wide variety of algorithms and data types.
- The execution time remains unaffected until an extreme power cap of 40W, creating opportunities for redistributing power throughout the system to more critical phases or applications.

On comparing different visualization algorithms (Section VI-B):

- Most of the visualization algorithms studied in this paper consume low amounts of power, so they can be run under a low power cap without impacting performance. These algorithms have lower IPC values, characteristic of data-bound workloads.
- Two of the explored algorithms consume higher power, similar to what we commonly see of traditional compute-bound benchmarks, such as Linpack. These algorithms will see significant slowdown from being run at a lower power cap, up to 3.2X. As such, the slowdown begins around 80W, roughly 67% of TDP. These algorithms have high IPC values, which are characteristic of compute-bound workloads.

On varying the input data set size (Section VI-C):

- For most algorithms, increasing the data set size is a poor tradeoff for performance. With a higher data set size, these algorithms start to slowdown at higher power caps. So instead of seeing a 10% slowdown at 50W with a data set size of 128^3, the slowdown begins at 70W for a data set size of 256^3.
- For the algorithms that were significantly compute-bound (and consuming high amounts of power), the change in data set size does not impact the power usage.

We can apply these recipes to two use cases in the context of a power-constrained environment. First, when doing post hoc visualization and data analysis on a shared cluster, requesting the lowest amount of power will leave more for other power-hungry applications. Second, when doing in situ visualization, appropriately provisioning power for visualization can either leave more power for the simulation or improve turn-around time for the visualization pipeline. We can integrate the findings into a job-level runtime system, like PaViz [5] or GEOPM [32], [33], to dynamically reallocate the power to the various components within the job. By providing more tailored information about the particular visualization routine, the runtime system may result in better overall performance.

VIII. Conclusion and Future Work

Our study explored the impacts of power constraints on scientific visualization algorithms. We considered a set of eight representative algorithms, nine different processor-level power caps, and four data set sizes, totaling 288 total test configurations. We believe the results of the study provide insights on the behavior of visualization algorithms on future exascale supercomputers. In particular, this study showed that visualization algorithms use little power, so applying an extremely low power cap will not impact the performance.

(Refer back to Section VII for specific findings.) We believe these findings can be used to dynamically reallocate power between competing applications (*i.e.,* simulation and visualization) when operating under a power budget. The runtime system would identify visualization workflows that are compute- or data-bound and allocate power accordingly, such that the scarce power is used wisely.

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

ACKNOWLEDGMENT

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-753659).

REFERENCES

[1] M. Maiterth *et al.*, "Energy and Power Aware Job Scheduling and Resource Management: Global Survey - Initial Analysis," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2018, pp. 685–693.

[2] M. Gamell *et al.*, "Exploring Power Behaviors and Trade-offs of In-Situ Data Analytics," in *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2013, pp. 1–12.

[3] V. Adhinarayanan *et al.*, "On the Greenness of In-Situ and Post-Processing Visualization Pipelines," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, May 2015, pp. 880–887.

[4] I. Rodero *et al.*, "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.

[5] S. Labasan *et al.*, "PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance," in *Eurographics Symposium on Parallel Graphics and Visualization*, A. Telea and J. Bennett, Eds. Eurographics Association, 2017.

[6] S. Labasan *et al.*, "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.

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

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

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

[10] T. Patki *et al.*, "Practical Resource Management in Power-Constrained, High Performance Computing," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: ACM, 2015, pp. 121–132.

[11] Z. Zhang *et al.*, "Trapped Capacity: Scheduling under a Power Cap to Maximize Machine-Room Throughput," in *2014 Energy Efficient Supercomputing Workshop*, Nov 2014, pp. 41–50.

[12] T. Patki *et al.*, "Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ser. ICS '13. New York, NY, USA: ACM, 2013, pp. 173–182.

[13] O. Sarood *et al.*, "Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 807–818.

[14] O. Sarood *et al.*, "Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2013, pp. 1–8.

[15] A. Marathe *et al.*, "An Empirical Survey of Performance and Energy Efficiency Variation on Intel Processors," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, ser. E2SC'17. New York, NY, USA: ACM, 2017, pp. 9:1–9:8.

[16] W. E. Lorensen *et al.*, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169.

[17] K. Moreland *et al.*, "VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures," *IEEE Computer Graphics and Applications*, vol. 36, no. 3, pp. 48–58, May 2016.

[18] W. J. Schroeder *et al.*, "The Design and Implementation of an Object-oriented Toolkit for 3D Graphics and Visualization," in *Proceedings of the 7th Conference on Visualization '96*, ser. VIS '96. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996, pp. 93–ff.

[19] H. Childs *et al.*, "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.

[20] J. Ahrens *et al.*, "ParaView: An End-User Tool for Large Data Visualization." 01 2005.

[21] J. Reinders, *Intel Threading Building Blocks*, 1st ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2007.

[22] M. Larsen *et al.*, "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*, ser. ISAV'17. New York, NY, USA: ACM, 2017, pp. 42–46.

[23] M. Larsen *et al.*, "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*. ACM, 2015, pp. 30–35.

[24] B. Whitlock *et al.*, "Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System," in *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, ser. EGPGV '11. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2011, pp. 101–109.

[25] N. Fabian *et al.*, "The ParaView Coprocessing Library: A scalable, general purpose in situ visualization library," in *2011 IEEE Symposium on Large Data Analysis and Visualization*, Oct 2011, pp. 89–96.

[26] "CloverLeaf," http://uk-mac.github.io/CloverLeaf/, University of Warwick, UK, 2017.

[27] A. Mallinson *et al.*, "Cloverleaf: Preparing Hydrodynamics Codes for Exascale," *Cray User Group*, pp. 6–9, 2013.

[28] "msr-safe," https://github.com/llnl/msr-safe, LLNL, 2016.

[29] "Intel processor event reference," https://download.01.org/perfmon/index/, Intel Corporation, 2017.

[30] K. Moreland *et al.*, "Formal Metrics for Large-Scale Parallel Performance," in *High Performance Computing*, J. M. Kunkel and T. Ludwig, Eds. Cham: Springer International Publishing, 2015, pp. 488–496.

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

[32] "geopm," https://github.com/geopm/geopm, Intel Corporation, 2016.

[33] J. Eastep *et al.*, "Global Extensible Open Power Manager: A Vehicle for HPC Community Collaboration on Co-Designed Energy Management Solutions," in *High Performance Computing - 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18-22, 2017, Proceedings*, 2017, pp. 394–412.