SeeSAw: Optimizing Performance of In-Situ Analytics Applications under Power Constraints

Ivana Marincic
University of Chicago
Chicago, IL, USA
imarincic@cs.uchicago.edu

Venkatram Vishwanath
Argonne National Laboratory
Lemont, IL, USA
venkat@anl.gov

Henry Hoffmann
University of Chicago
Chicago, IL, USA
hankhoffmann@cs.uchicago.edu

***Abstract*—Future supercomputers will need to operate under a power budget. At the same time, in-situ analysis—where a set of analysis tasks are concurrently executed and periodically communicate with a scientific simulation—is expected to be a primary HPC workload to overcome the increasing gap between the performance of the storage system relative to the computational capabilities of these machines. Ongoing research focuses on efficient coupling of simulation and analysis considering memory or I/O constraints, but power poses a new constraint that has not yet been addressed for these workflows. There are two state-of-the-art HPC power management approaches: 1) a *power-aware* scheme that measures and reallocates power based on observed usage and 2) a *time-aware* scheme that measures the relative time between communicating software modules and reallocates power based on timing differences. We find that considering only one feedback metric has two major drawbacks: 1) both approaches miss opportunities to improve performance and 2) they often make incorrect decisions when facing the unique requirements of in-situ analysis. We therefore propose SeeSAw—an application-aware power management approach, which uses both time and power feedback to balance a power budget and maximize performance for in-situ analysis workloads. We evaluate SeeSAw using the molecular dynamics simulation LAMMPS with a set of built-in analyses running on the Theta supercomputer on up to 1024 nodes. We find that the strictly power-aware approach slows down LAMMPS as much as ∼25%. The strictly time-aware approach shows improvements of up to ∼13% and slowdowns as much as ∼60%. In contrast, SeeSAw achieves ∼4–30% performance improvements.**

***Index Terms*—HPC, power-constraints, in-situ analysis**

I. Introduction

Future high performance computing (HPC) systems are expected to operate within strict power budgets [1], [2]. To optimize performance under power constraints, we require more intelligent power management at the system-, node- and process-level, to the application level. Application-level power management in HPC has been given relatively little treatment. In this work we demonstrate that we can harness application-specific knowledge from HPC application developers to achieve performance improvements in face of even the tightest power constraints.

We address in-situ analysis as it is of paramount importance to HPC. An in-situ analysis workflow consists of two sets of tasks: simulation and analysis. The analysis provides scientific insights into the simulation through computational or visualization tasks while reducing the I/O needs. As simulation runs, the analysis is invoked periodically to synchronize with the simulation. Figure 1 is a snapshot of a typical periodic synchronization pattern between a simulation and analysis process in the molecular dynamics simulation LAMMPS. The analysis process spends nearly half the time waiting to receive data from simulation, followed by brief spikes in activity after which it waits again resulting in unused power. With a more efficient power distribution, we minimize the idle time of the faster task, and even speed up both the simulation and analysis enabling them to reach the synchronization point sooner, thus, improving overall application runtime and energy efficiency.

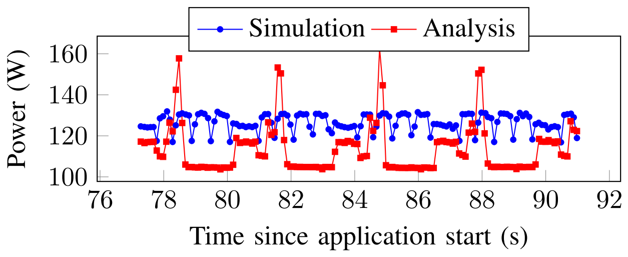


Fig. 1: Periodic synchronization pattern between a simulation and analysis process in LAMMPS.

The slack power and time in the example above can be detected by observing 1) differences in power—the waiting process will consume less power, or 2) differences in time—given a synchronization point, time measurements can differentiate faster from slower processes. However, the detection alone of these differences and linking them to application-specific events is non-trivial. There are two factors that make power management of in-situ frameworks challenging: 1) there is a complex organization of MPI processes – for instance, in LAMMPS there are both intra- and inter-dependencies of MPI sub-communicators, and not all dependencies equally contribute to power utilization inefficiencies, 2) different processes follow different workflows – e.g. LAMMPS can be configured such that different analyses communicate with the simulation at different intervals. These challenges can be addressed by exposing HPC application developers' knowledge about processes organization and communication patterns, which enables time and power measurements specific to the programmatic events we are interested in: compute and synchronization phases of simulation and analysis.

There are two state-of-the-art power management systems deployed on production HPC systems: the job scheduler SLURM [3] uses power feedback to shift power from nodes below to nodes at the power cap, and Intel's Power Balancer as part of GEOPM [4] uses time feedback to shift power from faster to slower nodes. By looking at only one feedback metric, both approaches miss opportunities for efficient power allocation and can make wrong decisions when faced with complex in-situ analysis workflow. Prior work by Zhang and Hoffmann [5] uses both time and power to shift power from fast to slow applications running concurrently and demonstrate improvements over SLURM. However, the timing data are based on estimates from offline profiles and, without code instrumentation of the application, do not tie to application-specific events of importance to in-situ analysis.

To optimize performance of power-constrained in-situ analysis workflows we propose See**SA**w, which finds the optimal power allocation between **s**imulation and **a**nalysis such that the two synchronize at the same time. Using minimal code instrumentation, SeeSAw observes time and power to obtain *energy* as a feedback metric, which provides insights the SLURM- and GEOPM-based approaches are lacking: tasks with no differences in power may still exhibit differences in time, while tasks with no differences in time may not utilize power efficiently.

Our paper makes the following contributions:

- We propose SeeSAw: the first dynamic and fully online power management solution for coordinating developer knowledge with system power management for in-situ analysis workflows.
- An empirical demonstration that production power management systems that rely on either power or timing alone miss crucial information.
- We propose energy as the right feedback metric for optimizing performance of in-situ analysis workflows under power constraints.
- An interface that allows scientists to communicate their application knowledge with minimal code instrumentation.

We refer to SLURM's approach as the *strictly power-aware approach*, and GEOPM as the *strictly time-aware approach*, and implement both on our evaluation platform, the Theta supercomputer system at Argonne National Laboratory, since both are not available on the system. We compare SeeSAw against the power- and time-aware approach relative to a static power allocation as the baseline on up to 1024 nodes, under varying simulation and analysis compositions. As a widely used and representative HPC workload, we use LAMMPS with its built-in analyses as a case study. We find that the strictly power-aware approach slows down LAMMPS as much as $\sim$25% compared to the baseline in all cases. The strictly time-aware approach shows improvements of up to $\sim$13% and slowdowns as much as $\sim$60%. SeeSAw achieves $\sim$4–30% improvement in time to complete the simulation.

II. Background and Related Work

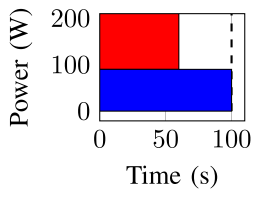
In-situ Analysis. Many in-situ analysis frameworks have been developed to date, enabling fast and scalable simulation-time analysis [6]–[16]. Few, however, consider resource constraints. Malakar et al [17], [18] consider IO and memory as constraints, and a fixed memory, compute and I/O resource profile, which can vary for a simulation over time and the dynamic needs weren't considered. SeeSAw adapts dynamically without requiring offline power and performance profiles.

Several works model energy and power requirements of in-situ frameworks, but do not take power constraints into account [19]–[23]. Adhinarayanan et al [24], [25] compare the energy cost of in-situ analysis against post-processing. Labasan et al [26] characterize the power and performance trade-off in visualization algorithms under power caps. These works do not consider the opportunity to optimize the slack power exposed through communication patterns of in-situ analysis that we are concerned with.

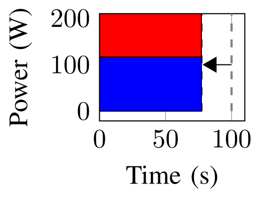
Power-constrained HPC. Recent works concern power-constrained HPC applications. The collection of power shifting algorithms proposed in [27] move power from I/O-intensive phases to compute phases, and require the duration of these phases to be known ahead of time. SeeSAw does not require any time or power information up front. *PowerShift* [5] is a collection of heuristics that rely on power and performance profiles collected offline of individual coupled applications to shift power from the faster to the slower application. The application couples are synthesized of stand-alone benchmarks. SeeSAw obtains feedback dynamically, and we demonstrate it on real-world workloads tightly coupled as one LAMMPS job. Related work in scheduling under power constraints [28]–[31] offer system-wide solutions complementary to our application-level approach.

Strictly power-aware approach. The real-world example of a strictly power-aware approach is deployed in the SLURM scheduler. This approach aims to address power imbalances between nodes by shifting excess power from nodes that are not at the power cap to nodes that are at the power cap. The excess power is divided evenly among nodes that require more power. This redistribution is performed periodically for a fixed time interval for the duration of a job.

Strictly time-aware approach. The strictly time-aware approach is given by GEOPM's power balancing plug-in. Given a power budget and an application loop, this approach slows down fast nodes that arrived at the end of the iteration first, and speed up the slower nodes by shifting a specific amount of power. The rate of change in power decreases over time so that small amounts of power are moved, and such that power does not fall below a hardware-supported minimum. Each node finds the median runtime of its respective ranks. The new power allocation aims to meet a target runtime corresponding to some percentage below the maximum median runtime of all nodes. The higher the percentage, the more reactive the algorithm is. If there is slack power, it is redistributed to all nodes equally.



(a) Inefficient state



(b) Efficient state with SeeSAw

Fig. 2: Illustration of the SeeSAw goal to shift power from the red to the blue task such that both finish at an earlier time.

III. Assumptions

We assume an in-situ analysis workflow can be partitioned into two sets of tasks on separate power domains, or some other pairwise coupling can be achieved. Simulation and analysis can be co-located on the same node, or on different sockets, or other configurations, but for SeeSAw to work on real systems, we are constrained to hardware-supported power domains. Future architectures are expected to provide finer-grained ways to control power. We restrict our focus in this paper to scenarios wherein the simulation nodes have equal amount of work. Likewise, analysis nodes have equal amount of work. An in-situ analysis workflow consists of a sequence of iterations, with at least one synchronization, and a synchronization consists of at least one set of tasks sending data to other tasks.

IV. SeeSAw: Optimizing In-situ Analysis Under Power Cap

SeeSAw balances the power allocation between simulation and analysis so that the two reach points of synchronization at the same time. Figure 2 illustrates how power is allocated with SeeSAw. Given a total 210 W power budget, the blue task requires 90 W and takes 100 s to reach the synchronization (dashed line), while the red task needs 120 W and 60 s. The duration of the whole iteration is dictated by the slower task, as shown in Figure 2a, with 120 W unused for 40 s. By moving just over 3W from the red to the blue task, SeeSAw reduces the time of the iteration to $\sim$77 s, as illustrated in Figure 2b.

To find how much power needs to be rebalanced so that the two tasks synchronize at the same time, SeeSAw addresses two challenges: 1) the function between power and time is unknown and difficult to estimate, 2) there could be presence of system noise affecting the power and performance behavior of the application [32]. By approximating the relationship between power and time as linear function, SeeSAw accounts for non-linearity with a series of small linear steps each time simulation and analysis synchronize. The second challenge is addressed by taking the steps in a controlled way to guard against anomalies and noise, in which past information is consolidated with the present using an exponentially weighted moving average.

SeeSAw measures past time and power to obtain energy as a feedback metric, which captures the impact of changes in power on time and vice versa. There is a total amount of energy required by simulation and analysis to reach a synchronization point. SeeSAw assigns a fraction of the power budget to each task corresponding to the fraction of that task's energy needs with respect to the total energy required.

Energy enables finding a new power value in one step, rather than incrementally moving power to slow down the faster task (the time-aware approach) or move unused power to the more power-demanding task (the power-aware approach). By shifting specific amounts of power determined by heuristics, these incremental approaches may also miss the power distribution that makes simulation and analysis equal in time, resulting in a less optimal or worse state. Further benefits of SeeSAw include: no requirements for offline profiling, minimal code instrumentation to indicate points of synchronizations, lightweight calculations incur negligible overhead.

The following sections state how SeeSAw is allocating power formally, code instrumentation requirements, and we conclude with an outline of limitations.

A. SeeSAw Formulation

Let C be the global power budget available for an in-situ analysis job. Let S and A designate the simulation and analysis tasks, respectively. Let t_i^S and t_i^A be the time it takes for S and A to reach the synchronization at time step i. Our goal is to reduce the time it takes both S and A to arrive at the synchronization point, so we have the following objective:

$$minimize \max\left(t_i^S, t_i^A\right)$$

The solution to this objective is optimal when:

$$t_i^S = t_i^A$$

A proof is given by Zhang and Hoffmann [5], and Demirci et al [30], [31]. We paraphrase: when moving power from one task to the other in the optimal state, one task will slow down beyond the optimal time and the other speed up, and thus the overall runtime as determined by the slower task is longer.

Our goal is to find optimal powers P^{OPT_S} and P^{OPT_A} such that we obey the power budget C and such that S and A arrive at a new time t^*. In other words:

$$P^{OPT\_S} + P^{OPT\_A} = C \tag{1}$$

$$t^{*S} = t^{*A} \tag{2}$$

To compute P^{OPT}, we formulate time as function of power which, as described above, we approximate as a linear relationship. We define a parameter α:

$$\alpha = \frac{1}{t^* \times P^{OPT}}$$

The system of equations follows based on Equations 1 and 2:

$$P^{OPT\_S} + P^{OPT\_A} = C, \quad \alpha^S P^{OPT\_S} = \alpha^A P^{OPT\_A}$$

Solving for P^{OPT_S} and P^{OPT_A} yields:

$$P^{OPT\_S} = C\frac{\alpha^A}{\alpha^S + \alpha^A}, \quad P^{OPT\_A} = C\frac{\alpha^S}{\alpha^S + \alpha^A} \tag{3}$$

At each synchronization we measure simulation and analysis power p and time t, and since these measurements can be noisy, we allow for a configurable window w which determines after how many synchronizations power will be redistributed. We take the average time and power over the last w intervals:

$$p_i^S = \frac{1}{w}\sum_{j=i-w}^{i} p_j^S, \quad p_i^A = \frac{1}{w}\sum_{j=i-w}^{i} p_j^A,$$
$$t_i^S = \frac{1}{w}\sum_{j=i-w}^{i} t_j^S, \quad t_i^A = \frac{1}{w}\sum_{j=i-w}^{i} t_j^A$$

where i is the time step at which we re-allocate power, ie at the start of every w synchronizations.

We obtain α_i^S and α_i^A from the measured values:

$$\alpha_i^S = \frac{1}{t_i^S \times p_i^S}, \quad \alpha_i^A = \frac{1}{t_i^A \times p_i^A} \tag{4}$$

We then use Equation 3 to find the power allocations P^{OPT_S} and P^{OPT_A} for the next w steps.

To account for noise, anomalies and to reduce the rate at which we change power at each synchronization point, we use the exponentially weighted moving average to set the optimal power. The weight we place on the most recent data is determined by the ratios:

$$r^S = \frac{P^{OPT\_S}}{C}, \quad r^A = \frac{P^{OPT\_A}}{C} \tag{5}$$

We compute the total new allocated power P^S for simulation and P^A for analysis for the next w steps:

$$\begin{aligned} P^S &= r^S \times P^{OPT\_S} + (1 - r^S) \times P^{OPT\_S} \\ P^A &= r^A \times P^{OPT\_A} + (1 - r^A) \times P^{OPT\_A} \end{aligned} \tag{6}$$

Let n and m be the number of nodes running the simulation and analysis, respectively. We then divide the new allocated power for the next w steps evenly:

$$P^{new\_S} = P^S/n, \quad P^{new\_A} = P^A/m \tag{7}$$

To account for the new power values being below what the hardware can support, we set a δ_{min} corresponding to the lowest supported power. If the simulation nodes are below δ_{min}, they are set at δ_{min} and analysis nodes at $C - \delta_{min} \times n$, and vice versa.

B. Harnessing Developers' Knowledge for Application Awareness

Developers can enable SeeSAw through two pieces of application-specific information: the application's process organization, and simulation-analysis synchronization pattern.

The user must supply a process' identity as simulation or analysis so SeeSAw is able to allocate power between the correct entities. In-situ frameworks already make this distinction typically using sub-communicators. Making this information available to SeeSAw addresses the challenge of having complex process organization, as SeeSAw need not be concerned with all possible relationships between processes, only the membership of a process is relevant.

Finally, the user must invoke SeeSAw prior to a synchronization, so that SeeSAw can correctly characterize the time and power of key application events. Since application developers know where synchronizations are, this too is a trivial task. Enabling this power check-pointing provides the necessary differentiation between independent work and when simulation and analysis are communicating. As illustrated on the case of LAMMPS in Section VI-C1, the two requirements can be satisfied in just 2 lines of code.

C. Limitations

The formulation above addresses paired workloads, which reflects common in-situ analysis workflows. SeeSAw also performs best with highly periodic application patterns, however configuring w can make SeeSAw resistant to anomalies. Finally, SeeSAw is aimed to optimize in-situ analysis workflows, and is not a complete solution for global HPC power management, however it can be used in tandem with such approaches.

V. LAMMPS: Overview

~~Molecular dynamics (MD) simulations help us understand the physics and chemistry governing a vast array of systems such as liquids, biomolecules and materials~~. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is an open-source massively parallel classical MD simulation written in C++, and a key workload on HPC supercomputers [33], [34]. To date it has been cited in over 10,000 publications [35], and is used by thousands of users world wide. It can simulate different systems (liquids, solids, etc) containing different types of particles (atoms, molecules, ions, etc).

We consider the velocity-Verlet timestepping algorithm which drives the LAMMPS simulation. The Verlet algorithm invokes specific analyses at the end of every j steps, where j is configurable. Malakar et al propose an extension to the Verlet algorithm called Verlet-*Splitanalysis* which forms pyhsically separate partitions of simulation and analysis processes [18], [36], [37]. When a partition is created, an analysis process is paired with one or more simulation processes within one MPI subcommunicator (if the number of simulation and analysis processes is m, there are m subcommunicators). Each time step has the following flow:

1) simulation performs initial integration
2) simulation sends particle coordinates and velocities to analysis partition
3) both partitions rebuild a subset of data structures
4) simulation sends particle count to analysis for verification
5) both partitions update neighbor lists
6) simulation computes forces and final integration
7) simulation invokes analysis at the end of time step
8) optional output of state of simulation

Steps 2–4 constitute the synchronization phase between simulation and analysis. Otherwise, simulation and analysis are doing independent work. The rebuilding of neighbor lists concerns the update of positions of particles, and is a communication-intensive phase. In our LAMMPS runs we request output of thermodynamic data at end of each time step, which is also communication- and I/O-intensive. Overall, at each time step both simulation and analysis perform a series of actions with different resource utilization.

VI. Methodology and Experimental Setup

We describe the experimental hardware and software setup and the code instrumentation of LAMMPS.

A. Hardware Setup

All experiments were run on the Theta supercomputer at Argonne National Laboratory—a Cray XC40 system ranked #28 by the Top500 project [38]. Theta has 4392 single-socket compute nodes with second-generation 64-core Intel Xeon Phi 7230 CPUs. The base frequency of each node is 1.3 GHz with turbo frequency up to 1.5 GHz and TDP of 215 W. Each node on Theta supports power capping via Intel's RAPL interface [39], accessible to users through the msr-safe module [40]. Power capping and monitoring on Theta is at the node level.

B. Software Setup: Power Management

We extend a power monitoring and capping library for distributed MPI applications called PoLiMEr [41] to implement SeeSAw. We implement the power- and time-aware approaches as closely as possible to the original implementations provided in [42] and [43][1]. PoLiMEr supports power monitoring and capping via RAPL, and monitors time of each MPI rank. It uses the available MPI ranks of the application, and designates one rank per node for power monitoring. Power can be monitored and capped on specific code regions as well.

For each interval i between two synchronizations, t_i^S is the runtime of the slowest simulation rank, and t_i^A is the slowest runtime of the analysis rank. The measured power p_i^S and p_i^A is the sum of power measurements from all simulation and analysis nodes, respectively. The time and power of performing the power allocation itself is included in these measurements.

The SLURM-based power-aware approach checks if power can be shifted at a fixed time interval independent of the application. To give the power-aware approach an advantage, in our implementation we allocate power at the start of the simulation-analysis synchronization instead, because we expect a fixed time interval to perform badly with non-uniform workloads such as LAMMPS with *Splitanalysis*. The w window applies to our power-aware implementation, to account for potential noise in power measurements.

GEOPM's power balancer is invoked at each iteration of any application loop. Since analyses may not be invoked at each iteration, our implementation aids the time-aware algorithm by invoking it at each synchronization instead. Changing w does not have an effect, to mimic the original intended behavior.

[1]https://github.com/PoLiMEr-HPC/PoLiMEr-SeeSAw

C. Software Setup: LAMMPS

We use a custom benchmark for LAMMPS from prior works [17], [36], [37], which simulates a box of water molecules solvating two types of ions. The problem size in LAMMPS can be controlled with the dimension size dim of the cube within which the base number of atoms is replicated. The benchmark we use includes 1568 atoms, so the total number of atoms is $1568 \times dim^3$. In our results we report the setting for dim.

The analyses we consider are commonly used in scientific computing [44], [45], and used as case studies in [17], [18], [36]. They are:

- Hydronium and ion RDF – radial distribution functions, averaged over all molecules
- VACF – velocity auto-correlation function
- MSD, MSD1D, MSD2D – mean squared displacements for 1D and 2D spatial bins, averaged over all molecules

The analyses have different resource requirements [18]. MSD has high CPU and memory utilization, MSD1D and MSD2D are mostly memory-intensive, RDF is compute bound, and VACF has both low memory and CPU utilization. An analysis is invoked every j timesteps, and the number of analysis and simulation ranks is equal in all results in Section VII.

1) Modifications to LAMMPS: We modified the LAMMPS *Splitanalysis* extension described in Section V to accept additional parameters for purposes of evaluation: power cap per node, power cap per simulation node, power cap per analysis node (the latter two for examining unbalanced initial power caps, see Section VII-D).

Before the Verlet algorithm, we initialize PoLiMEr's power management capability:

```
//universe->uworld : MPI communicator for all
  processes (can be MPI_COMM_WORLD)
//universe->me : rank of current process
//master : 0 if simulation, 1 if analysis
  process
//power_cap : initial power cap on node of
  current process (specified by user)
poli_init_power_manager(universe->uworld,
  universe->me, master, power_cap);
```

This distinguishes simulation from analysis processes.

Before the synchronization, reallocate power by calling:

```
poli_power_alloc();
```

This indicates when a power manager should reallocate power.

VII. Evaluation

We evaluate SeeSAw against the power- and time-aware approaches relative to a static assignment of power caps, where the global power budget is divided equally between simulation and analysis nodes. This baseline represents the application's performance in a power-constrained environment. We use a power cap of 110 W per node (see Section VII-E).

First we outline steps taken to mitigate run-to-run variability, followed by comparing the power management algorithms on different analyses and scales. We then provide a sensitivity analysis specific to SeeSAw, followed by SeeSAw overhead measurements.

A. *Variability of Results*

Large-scale systems are subject to run-to-run variability. Table I shows the variability of 7 runs for different settings of power caps for LAMMPS on 128 nodes, ran for 1000 time steps. Variability is low on repeated runs on the same node. Therefore, to reduce the job sizes of our experiments, we compare the differences between jobs each containing a run with the power management algorithm of interest, followed by a baseline run on the same nodes.

TABLE I: Variability across 7 runs for different types of power caps and problem size for LAMMPS on 128 nodes.

Power Cap	dim	Variability Type	Variability %
None	36	run-to-run	0.8
	36	job-to-job	2.0
	48	run-to-run	0.2
	48	job-to-job	0.8
Long (110 W)	36	run-to-run	0.7
	36	job-to-job	6.0
	48	run-to-run	0.3
	48	job-to-job	5.7
Long and Short (110 W each)	36	run-to-run	2.1
	36	job-to-job	8.7
	48	run-to-run	5.5
	48	job-to-job	2.4

Variability is exacerbated by power caps. RAPL maintains a moving average of the requested power cap over a period of time. A short-term power cap can be specified to allow for brief violations of the long-term power cap. Limiting both the long- and short-term power cap guarantees the power budget will not be violated as RAPL limits the power slightly below the requested power, however, the variability increases. The power-aware algorithm takes action only if nodes are at the power cap, otherwise it assumes the application has available power. Since SeeSAw does not depend on the application reaching the power cap, it can make progress and allocate power regardless of what RAPL power cap settings are used. For a fair evaluation, however, we show results for long-term power capping only.

B. *Different Analyses Require Different Power Allocations*

We compare SeeSAw against the power- and time-aware approaches on LAMMPS with different sets of analyses, shown in Figure 3. Full MSD includes all three components: MSD1D, MSD2D and a final averaging of all particles. The *all* category includes RDF, MSD1D, MSD2D and final MSD averaging in case of full MSD, followed by VACF, executed one after another at each synchronization. The full MSD analysis is a more demanding workload compared to VACF and RDF and its subcomponents.

SeeSAw especially outperforms in the case of full MSD. Figure 4 illustrates the difference in performance between SeeSAw and the other two approaches, showing the power allocated per node throughout LAMMPS+MSD. The black graphs show the slack time – the difference between simulation and analysis time between synchronizations – normalized to

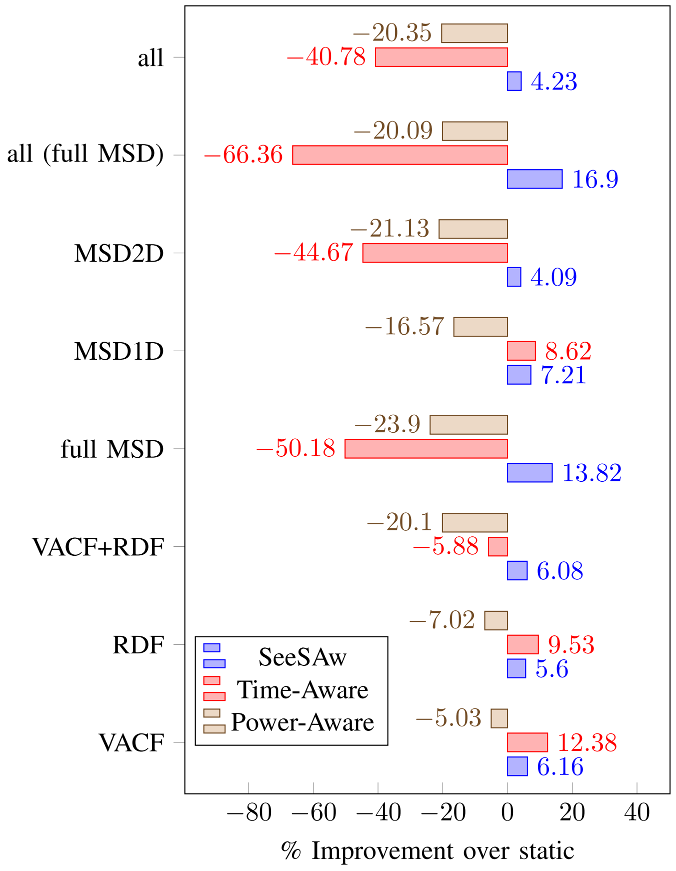


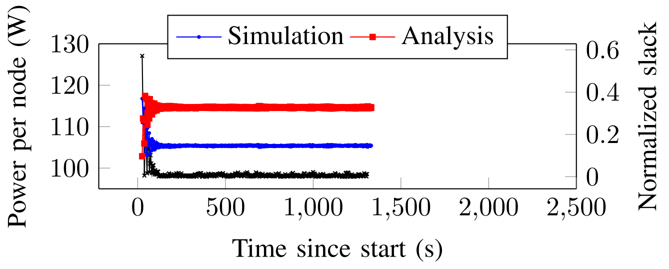
Fig. 3: Performance of SeeSAw, time- and power-aware approaches for different analyses with LAMMPS running on 128 nodes for 400 time steps, $dim = 36$ (for MSD $dim = 16$ due to memory constraints), $w = 1$, synchronization at every time step ($j = 1$). Median of 3 runs shown.

the total time it took for the longest process to reach each synchronization.

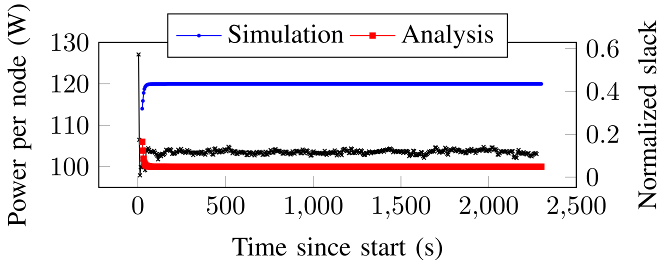
For reference, Figures 4d and 4e show baseline simulation and analysis time and power between the first 10 synchronization points when power is capped at 110 W per node. In our implementation, time step 0 is ignored, because it is outside of the main simulation loop. In the first couple synchronization steps the simulation has extra setup overhead, which is consistent with repeated runs when MSD is present. Onward MSD and LAMMPS are nearly identical in runtime with 4 seconds between synchronizations, whereas VACF and RDF are 3-4$\times$ faster than LAMMPS.

Figure 4a shows that SeeSAw settles on a power distribution within the first 20 out of 400 time steps, where the analysis is assigned more power, bringing the relative slack time down to an average of 0.8% (calculated from the 10th step).

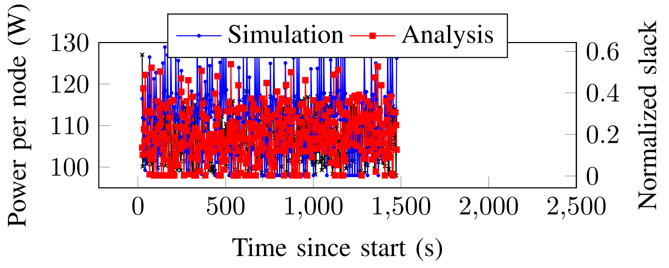
By only referencing time, the time-aware approach can move power in the wrong direction and not correct the power allocation later on. Because MSD is initially faster than simulation, as shown in Figure 4d, the time-aware approach assigns it more power too quickly. The power allocation flattens



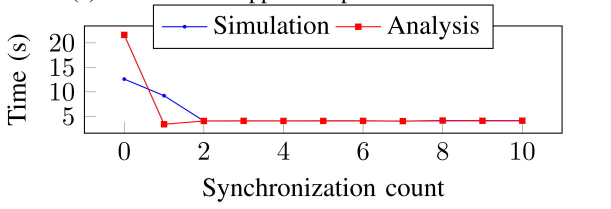
(a) SeeSAw: power allocated over time



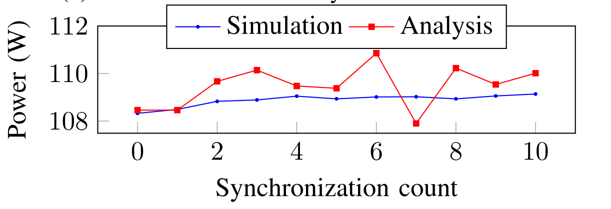
(b) Time-aware approach: power allocated over time



(c) Power-aware approach: power allocated over time



(d) Time between first 10 synchronizations for baseline



(e) Power between first 10 synchronizations for baseline

Fig. 4: Power allocation per simulation and analysis node at each synchronization step. Each row corresponds to a run of LAMMPS with the MSD analysis on 128 nodes, $j = 1, dim = 16$. The right y-axis (black) shows the normalized slack time between simulation and analysis relative to the total time interval between each synchronization. Bottom two charts show time and power between first 10 synchronizations for simulation and analysis without any power management.

out due to a reduction in the rate of change parameter and reaching δ_{min} (98 W). It is unable to return to a better power distribution, because at each subsequent synchronization point simulation and analysis nodes alternate in terms of being the slowest, causing no net power being shifted over time. Furthermore, the simulation is not able to utilize the assigned 120 W per node. Power measurements show that simulation consumes 102-104 W at each synchronization, due to the analysis nodes being capped low. The slack time of 12% reflects this inefficient distribution.

Finally, by only referencing power, the power-aware approach is unaware of performance impacts of its chosen power allocations. Under the power-aware approach, the slack time fluctuates between 0.2% and 40%. Without any metric for efficiency, it simply responds to potentially noisy differences in measured power, worsened by irregular application patterns.

The MSD result shows that SeeSAw is able to address the counter-intuitive need to give analysis more power even though simulation and analysis are both nearly identical in time as shown in the baseline time measurements in Figure 4d. The time-aware approach works well when the time and power characteristics of simulation and analysis are predictable, such is the case with LAMMPS+RDF and LAMMPS+VACF. This algorithm picks a direction and gradually increases the gap in the power distribution, settling at 120 W and 121 W for each simulation node, and 100 W and 101 W for each analysis node for LAMMPS+RDF and LAMMPS+VACF respectively. SeeSAw does not exceed 115 W per simulation node for LAMMPS+RDF and 117 W for LAMMPS+VACF. This suggests that SeeSAw may be susceptible to local optima. Finally, the power-aware approach is sensitive to noisy environments and is not suitable for applications where nodes do not have equal amount of work.

C. *Impact of Limited Power Utilization at Scale*

At larger scales an additional deciding power allocation factor are utilization limits due to communication overhead. Figure 5 shows the performance difference between the three power allocation algorithms in case of full MSD, all analyses and VACF, chosen as representative workloads out of the 8 cases from Figure 3. Figure 6 shows a detailed comparison between SeeSAw and the time-aware approach for all analyses. We omit the power-aware approach as its behavior is similar to Figure 4c.

In Figure 6a we see that SeeSAw has allocated more power to analysis. For the same workload and problem size on 128 nodes, SeeSAw fluctuates between 109-115 W per simulation node, which suggests that simulation at larger scale has lower power utilization.

Even though the normalized slack time in Figure 6b is nearly 0, the time-aware approach causes severe performance degradation. The time-aware approach chooses the wrong direction again, increasing the gap in power distribution until reaching δ_{min}. The simulation is forced into a low-power state as well, due to the analysis being capped at δ_{min} and longer but low-power communication-intensive phases. As both tasks

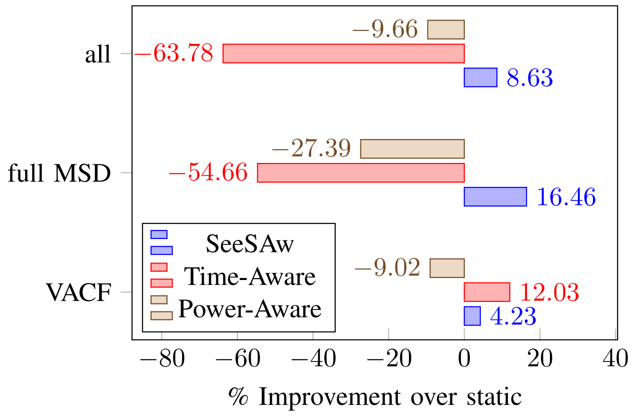
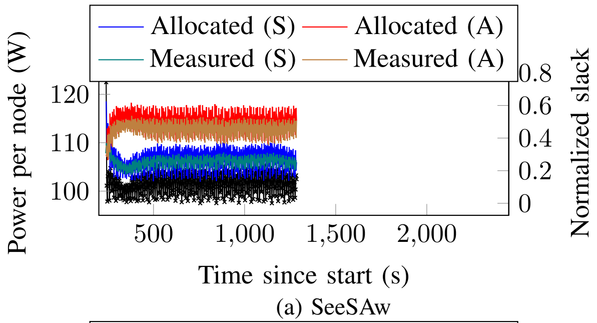
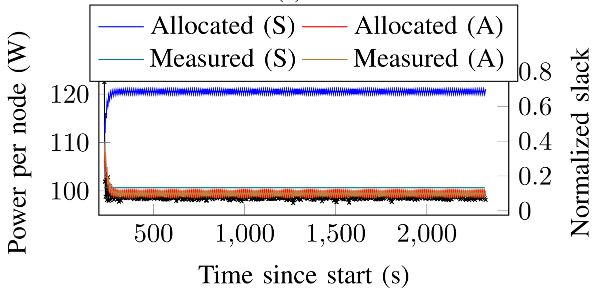


Fig. 5: Performance of SeeSAw, time- and power-aware approaches for different analyses with LAMMPS running on 1024 nodes for 400 time steps, $dim = 48$ (for MSD $dim = 16$ due to memory constraints), $w = 1$, synchronization at every time step ($j = 1$). Median of 3 runs shown.



(a) SeeSAw



(b) Time-aware approach

Fig. 6: Power measured compared to allocated between synchronizations by SeeSAw and time-aware approach at 1024 nodes. Normalized slack shown in black.

are running barely above the system operating power, the time difference between them is incidentally low. Therefore, low differences in time between simulation and analysis is not indicative of an energy-efficient state.

D. Sensitivity Analysis

In this section we discuss parameters that affect SeeSAw's performance and examine the impact of analyses with mixed intervals and initial unbalanced power between simulation and analysis.

1) Impact of SeeSAw Window Size and Synchronization Rate: Figure 7 shows the impact of the frequency j at which simulation and analysis synchronize, and the frequency w at which power is reallocated on SeeSAw's performance on 1024 nodes when LAMMPS is executed with all analyses. Overall, allocating power more frequently is favorable over infrequent re-allocations which miss past slack optimization opportunities. If invoked at each synchronization step, SeeSAw is more reactive to anomalies, and setting $w = 2$ mitigates that. When simulation and analysis synchronize less frequently, SeeSAw has fewer opportunities to correct inefficient power distributions causing LAMMPS to spend more time in inefficient states. In such configurations, we observe that allocating power as frequently as possible helps.

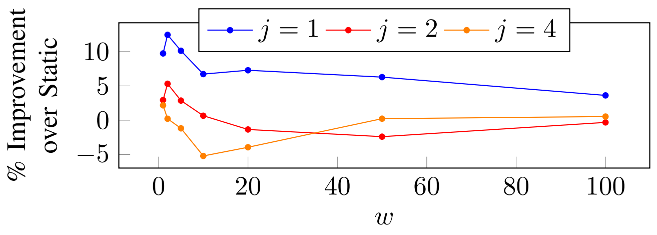


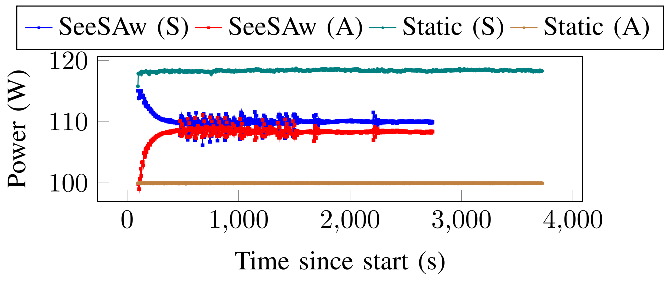
Fig. 7: SeeSAw w and LAMMPS synchronization rate j on 1024 nodes, $dim = 48$, mix of analyses.

2) Mixed Analysis Intervals: Certain analyses can be required to run less or more frequently than others. Table II shows the impact of varying the frequency of full MSD while RDF and VACF synchronize with the simulation at each time step, and power is allocated at each step. As the frequency of MSD decreases, and by fixing $w = 1$ SeeSAw becomes too reactive to the now anomalous MSD analysis. We find that SeeSAw tolerates variable analysis frequencies well. If developers require an analysis to run occasionally, by setting $w = 2$ or higher, those events will not trigger sudden changes in power allocation.

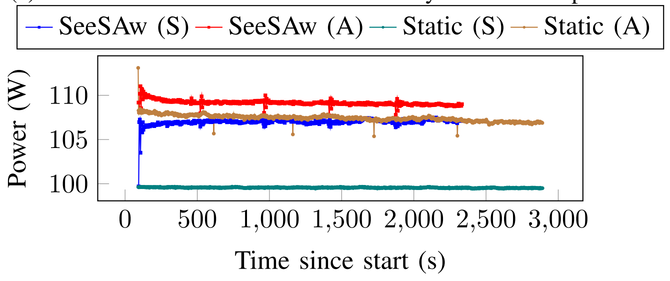
TABLE II: SeeSAw performance improvement over the baseline on 400 steps of LAMMPS with RDF, full MSD and VACF, where the frequency of full MSD is varied, while the rest synchronize at each time step. Median of 3 runs shown, performed on 128 nodes, $w = 1$, $dim = 16$.

j_{MSD}	4	20	100
% Improvement over static	5.03	0.94	0.90

3) Unbalanced Initial Power Distribution: Finally, we address the case wherein simulation and analysis start with uneven power distribution. If given different resources, or when analysis is partially co-located with simulation, or in cases of heterogeneous hardware on multiple nodes, there can be a greater difference in power requirements between simulation and analysis. Figure 8 shows that SeeSAw improves performance in such cases in comparison to keeping simulation and analysis at the initial power distribution. The figure depicts the measured simulation and analysis power per node



(a) Simulation starts with 120 W and analysis at 100 W per node



(b) Analysis starts with 120 W and simulation at 100 W per node

Fig. 8: Power measured under SeeSAw per simulation (S) and analysis (A) node compared static baseline, in case of unbalanced initial distribution of power. LAMMPS was run on 128 nodes, all analyses, $dim = 36$, $w = 2$.

for SeeSAw compared to the static allocation. The median of three runs shows performance improvement of 28.26% when simulation starts with more power, and 19.21% when analysis starts with more power.

E. Diminishing Returns with More Power Headroom

Figure 9 shows that SeeSAw is more effective the tighter the power budget. With more liberal power budgets, additional power management through SeeSAw yields limited benefits as LAMMPS fails to utilize additional power beyond 140 W per node. Since the minimum power allowed by RAPL on Theta is 98 W per node, in our experiments we choose 110 W as the baseline power cap. This allows for some power headroom above the minimum threshold while still being relatively low compared to LAMMPS' needs. In previous results we saw that performance improvements are attainable even for relatively low-power workloads.

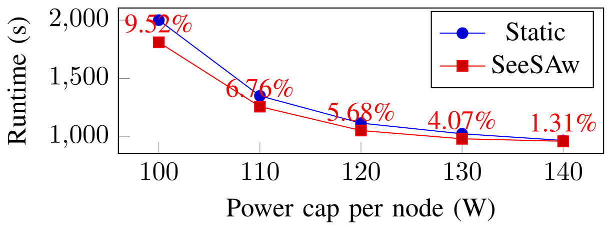


Fig. 9: Improvement of SeeSAw over static power allocation for varying power caps. LAMMPS is run on 128 nodes, $dim = 36$, $w = 10$. Each point is the median total runtime of 3 runs.

F. Overhead

We report the overhead of SeeSAw as the time to compute a new power allocation, measured for each power management rank in PoLiMEr. The overhead stems from communication costs involved in exchanging power and time measurements between ranks. Figure 10 shows the median overhead measured at the end of each synchronization.

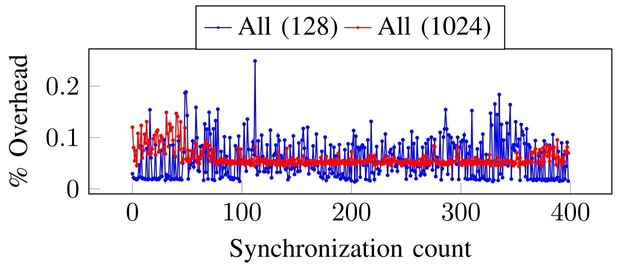


Fig. 10: Overhead of SeeSAw as a percentage of total time at each synchronization in case of all analyses ($dim = 48$) on 128 and 1024 nodes. LAMMPS was set to 400 time steps, synchronization and power allocation invoked at each step.

We show the percent overhead in the representative case when all analyses used from Figure 3 and Figure 5. Overall, less than 1% of the application time is devoted to power reallocation with SeeSAw. Furthermore, the added overhead is incorporated in the time measurements by SeeSAw, enabling it to address any power or time imbalance introduced by its own actions.

VIII. Conclusion and Future Work

SeeSAw is the first dynamic and completely online power management solution for coordinating HPC application developer knowledge with system power management for in-situ analysis workflows. Our results demonstrate that energy is the right feedback metric for optimizing performance of in-situ analysis workflows under power constraints. Existing strictly power- or time-aware solutions miss key information such as power utilization capabilities, whether measured feedback is noise, or when simulation or analysis benefits from more power than the other.

An avenue for future work involves adding elements of exploration to SeeSAw to detect and overcome local optima. This will enable even more performance gains in the case of highly uniform simulation-analysis workflows. Furthermore, SeeSAw could be implemented in or integrated with job schedulers and system-wide power management schemes.

Our results suggest that employing dynamic approaches which utilize application-specific knowledge is a promising direction for future research on power management in HPC.

References

[1] K. Bergman, S. Borkar *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems peter kogge, editor & study lead," 2008.

[2] V. Sarkar, S. Amarasinghe *et al.*, "Exascale software study: Software challenges in extreme scale systems," 2009, dARPA IPTO Study Report for William Harrod.

[3] "The slurm workload manager," https://slurm.schedmd.com.

[4] J. Eastep, S. Sylvester *et al.*, "Global extensible open power manager: a vehicle for hpc community collaboration toward co-designed energy management solutions," *Supercomputing PMBS*, 2016.

[5] H. Zhang and H. Hoffmann, "Performance & energy tradeoffs for dependent distributed applications under system-wide power caps," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: ACM, 2018, pp. 67:1–67:11. [Online]. Available: http://doi.acm.org/10.1145/3225058.3225098

[6] H. Abbasi, M. Wolf *et al.*, "Datastager: scalable data staging services for petascale applications," *Cluster Computing*, vol. 13, no. 3, pp. 277–290, 2010.

[7] U. Ayachit, A. Bauer *et al.*, "Performance analysis, design considerations, and applications of extreme-scale in situ infrastructures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 79.

[8] T. Bicer, D. Gursoy *et al.*, "Real-time data analysis and autonomous steering of synchrotron light source experiments," in *e-Science (e-Science), 2017 IEEE 13th International Conference on*. IEEE, 2017, pp. 59–68.

[9] M. Dorier, G. Antoniu *et al.*, "Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free i/o," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 2012, pp. 155–163.

[10] J. Y. Choi, C.-S. Chang *et al.*, "Coupling exascale multiphysics applications: Methods and lessons learned," in *2018 IEEE 14th International Conference on e-Science (e-Science)*. IEEE, 2018, pp. 442–452.

[11] M. Dreher and B. Raffin, "A flexible framework for asynchronous in situ and in transit analytics for scientific simulations," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 277–286.

[12] T. Kuhlen, R. Pajarola *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 (EGPGV)*, 2011.

[13] A. Luckow, G. Chantzialexiou *et al.*, "Pilot-streaming: A stream processing framework for high-performance computing," *arXiv preprint arXiv:1801.08648*, 2018.

[14] "Paraview catalyst," http://catalyst.paraview.org.

[15] F. Zheng, H. Abbasi *et al.*, "Predata–preparatory data analytics on peta-scale machines," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[16] F. Zheng, H. Zou *et al.*, "Flexio: I/o middleware for location-flexible scientific data analytics," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, 2013, pp. 320–331.

[17] P. Malakar, V. Vishwanath *et al.*, "Optimal scheduling of in-situ analysis for large-scale scientific simulations," in *High Performance Computing, Networking, Storage and Analysis, 2015 SC-International Conference for*. IEEE, 2015, pp. 1–11.

[18] ——, "Optimal execution of co-analysis for large-scale molecular dynamics simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 60.

[19] M. Dorier, O. Yildiz *et al.*, "On the energy footprint of i/o management in exascale hpc systems," *Future Generation Computer Systems*, vol. 62, pp. 17–28, 2016.

[20] M. Gamell, I. Rodero *et al.*, "Exploring power behaviors and trade-offs of in-situ data analytics," in *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*. IEEE, 2013, pp. 1–12.

[21] G. Haldeman, I. Rodero *et al.*, "Exploring energy-performance-quality tradeoffs for scientific workflows with in-situ data analyses," *Computer Science-Research and Development*, vol. 30, no. 2, pp. 207–218, 2015.

[22] I. Rodero, M. Parashar *et al.*, "Evaluation of in-situ analysis strategies at scale for power efficiency and scalability," in *Cluster, Cloud and Grid Computing (CCGrid), 2016 16th IEEE/ACM International Symposium on*. IEEE, 2016, pp. 156–164.

[23] O. Yildiz, M. Dorier *et al.*, "A performance and energy analysis of i/o management approaches for exascale systems," in *Proceedings of the sixth international workshop on Data intensive distributed computing*. ACM, 2014, pp. 35–40.

[24] V. Adhinarayanan, W.-c. Feng *et al.*, "On the greenness of in-situ and post-processing visualization pipelines," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*. IEEE, 2015, pp. 880–887.

[25] ——, "Characterizing and modeling power and energy for extreme-scale in-situ visualization," in *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*. IEEE, 2017, pp. 978–987.

[26] S. Labasan, M. Larsen *et al.*, "Power and performance tradeoffs for visualization algorithms," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 325–334.

[27] L. Savoie, D. K. Lowenthal *et al.*, "I/o aware power shifting," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 740–749.

[28] T. Patki, D. K. Lowenthal *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*. ACM, 2015, pp. 121–132.

[29] O. Sarood, A. Langer *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*. IEEE Press, 2014, pp. 807–818.

[30] G. Demirci, H. Hoffmann *et al.*, "Approximation algorithms for scheduling with resource and precedence constraints," in *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[31] G. Demirci, I. Marincic *et al.*, "A divide and conquer algorithm for dag scheduling under power constraints," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 466–477.

[32] S. Chunduri, K. Harms *et al.*, "Run-to-run variability on xeon phi based cray xc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 52.

[33] "Lammps," http://lammps.sandia.gov.

[34] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[35] "Lammps publications," https://lammps.sandia.gov/papers.html, accessed: 2019-09-26.

[36] P. Malakar, C. Knight *et al.*, "Scalable in situ analysis of molecular dynamics simulations," in *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*. ACM, 2017, pp. 1–6.

[37] P. Malakar, T. Munson *et al.*, "Topology-aware space-shared co-analysis of large-scale molecular dynamics simulations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. IEEE Press, 2018, p. 24.

[38] "Top500 list - june 2019," https://www.top500.org/list/2019/06/, accessed: 2019-07-20.

[39] H. David, E. Gorbatov *et al.*, "Rapl: memory power estimation and capping," in *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 2010, pp. 189–194.

[40] K. Shoga, B. Rountree *et al.*, "Whitelisting msrs with msr-safe," in *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*, 2014.

[41] I. Marincic, V. Vishwanath *et al.*, "Polimer: An energy monitoring and power limiting interface for hpc applications," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*. ACM, 2017, p. 7.

[42] "Slurm power management documentation," https://slurm.schedmd.com/power_mgmt.html, accessed: 2019-09-16.

[43] "Geopm power balancer source code," https://github.com/geopm/geopm/blob/dev/src/PowerBalancer.cpp, accessed: 2019-09-16.

[44] M. P. Allen and D. J. Tildesley, *Computer simulation of liquids*. Oxford university press, 2017.

[45] N. Michaud-Agrawal, E. J. Denning *et al.*, "Mdanalysis: a toolkit for the analysis of molecular dynamics simulations," *Journal of computational chemistry*, vol. 32, no. 10, pp. 2319–2327, 2011.