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], requiring more intelligent power management of hardware, systems and applications. Application-level power management in HPC has been given relatively little treatment. In this work we harness application-specific knowledge from HPC application developers to achieve performance improvements of in-situ analytics applications.

In-situ analysis is of paramount importance to HPC. Through periodic synchronization, simulation-time analysis provides scientific insights into the simulation using computation or visualization while reducing the I/O needs. Figure 1 shows a power trace from the molecular dynamics simulation LAMMPS, demonstrating simulation-analysis activity. A simulation step corresponds to the regions between two analysis spikes in activity, and for each step, the analysis spends nearly half the time idling at ∼105 W to synchronize with simulation resulting in unused power.

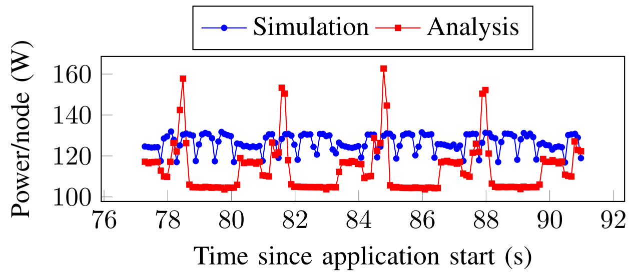


Fig. 1: Partial power trace of LAMMPS simulation and analysis processes running on separate nodes on the Theta supercomputer, exposing periodic synchronization. Power collected every 200 ms.

This idling period 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 differentiate faster from slower processes. However, detecting such differences and linking them to application-specific events is non-trivial due to complex process organization and workflows. In-situ analysis frameworks can organize MPI processes with both intra- and inter-dependencies of MPI sub-communicators. Different analyses can communicate with simulation at different intervals, resulting in complex workflows that can be difficult to learn using an online process. Therefore, by exposing HPC application developers' knowledge about processes organization and communication patterns, we can relate time and power measurements to programmatic events we are interested in: simulation-analysis compute and synchronization phases.

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.

To optimize performance of power-constrained in-situ analysis 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. 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.

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.

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 strategy as the strictly *power-aware* and GEOPM as the strictly *time-aware* approach. Using varying simulation and analysis compositions, we compare SeeSAw against both approaches on the Theta supercomputer system at Argonne National Laboratory relative to a static power allocation as the baseline on up to 1024 nodes. 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 Optimizations. Many in-situ analysis frameworks have been developed to date, enabling fast and scalable simulation-time analysis [6]–[16]. Few consider resource constraints. Malakar et al [17], [18] consider a fixed and offline memory, compute and I/O resource profile. However, these vary over time and the dynamic needs of the application were not 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 to compute phases, but 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 where analyses are not executable as stand-alone tasks. Related work in power-constrained scheduling [28]–[31] offer system-wide solutions complementary to our application-level approach.

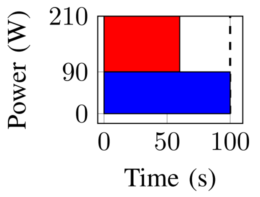
Strictly power-aware approach. The real-world example of a strictly power-aware approach is deployed in the SLURM scheduler. This approach aims to address power imbalances between nodes by shifting excess power from nodes that are not at the power cap to nodes that are at the power cap. The excess power is divided evenly among nodes that require more power. This redistribution is performed at fixed time intervals 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 nodes which arrived at the end of the iteration first, and speeds up the slower nodes by shifting a specific amount of power. The rate of change in power decreases over time until a user-configured minimum. Each node finds the median runtime of its respective ranks. A target runtime is designated 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.

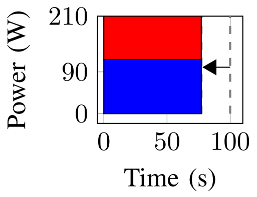
III. Assumptions

We assume the following conditions:

- Space-shared in-situ analysis. The time-shared mode with alternating simulation and analysis poses a simpler problem of managing a power budget: when one workload enters the critical section, power can be either kept at the budget or reduced to save energy. SeeSAw addresses the unique requirements of space-shared in-situ analysis.
- The in-situ analysis workflow is partitioned into two sets of tasks on separate power domains or voltage planes. For example, if per-core power can be controlled, simulation and analysis can be co-located on the same CPU. On our evaluation platform, power is controlled per node.
- Simulation (analysis) processes have equal work.



(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. Dashed lines show synchronization points.

- Simulation (analysis) compute units are equal, but compute units for simulation can be different from analysis.

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

SeeSAw balances power 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. With 210 W total power for all compute units (nodes, cores, etc.), the blue task requires 90 W and takes 100 s to reach the synchronization, while the red task needs 120 W and 60 s. In Figure 2a 120 W is unused for 40 s. By moving $\sim$3 W from the red to the blue task, SeeSAw reduces the iteration time to $\sim$77 s, as illustrated in Figure 2b.

Finding how much power to redistribute requires addressing 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 a linear function, SeeSAw accounts for non-linearity with a series of small linear steps each time simulation and analysis synchronize. SeeSAw addresses the second challenge 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 uses energy as the feedback metric from past time and power measurements to make decisions about power allocations, because energy captures the impact of changes in power on time and vice versa. Then, a fraction of the power budget is assigned to each task corresponding to the fraction of that task's energy needs with respect to the total energy required by simulation and analysis to reach a synchronization.

Furthermore, 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, thus 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, light-weight calculations incur negligible overhead.

A. SeeSAw Formulation

We formally describe how SeeSAw allocates power for power-constrained in-situ analysis. 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:

$$\min_i \, max\left(T_i^S, T_i^A\right)$$

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.

To account for noisy measurements of past time and power, we provide a configurable window w which determines after how many synchronizations to redistribute power. We take the average time and power over the last w intervals to obtain a value of simulation power P_j^S and time T_j^S at each allocation j, ie at the start of every w synchronizations:

$$P_j^S = \frac{1}{w}\sum_{i=j-w}^{j} p_i^S, \quad T_j^S = \frac{1}{w}\sum_{i=j-w}^{j} t_i^S$$

and likewise for analysis power P_j^A and T_j^A.

Using these past measurements, the goal is to find optimal powers $P_{j+i}^{OPT_S}$ and $P_{j+1}^{OPT_A}$ such that we obey the power budget C and such that S and A arrive at a new time t_{j+1}^* for the next w synchronizations.

To compute the new optimal power, we approximate the time and power relationship as linear using a parameter α:

$$\alpha_j^S = \frac{1}{T_j^S \times P_j^S}, \quad \alpha_j^A = \frac{1}{T_j^A \times P_j^A} \tag{1}$$

Using the power constraint C and the time equality property, we solve for $P_{j+1}^{OPT_S}$ and $P_{j+1}^{OPT_A}$ for the next allocation:

$$P_{j+1}^{OPT_S} = C\frac{\alpha_j^A}{\alpha_j^S + \alpha_j^A}, \quad P_{j+1}^{OPT_A} = C\frac{\alpha_j^S}{\alpha_j^S + \alpha_j^A} \tag{2}$$

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:

$$r_{j+1}^S = \frac{P_{j+1}^{OPT_S}}{C}, \quad r_{j+1}^A = \frac{P_{j+1}^{OPT_A}}{C} \tag{3}$$

We compute the total new allocated power for simulation and analysis for the next w steps:

$$\begin{aligned} P_{j+1}^{new_S} &= r_{j+1}^S \times P_{j+1}^{OPT_S} + (1 - r_{j+1}^S) \times P_{j+1}^{OPT_S} \\ P_{j+1}^{new_A} &= r_{j+1}^A \times P_{j+1}^{OPT_A} + (1 - r_{j+1}^A) \times P_{j+1}^{OPT_A} \end{aligned} \quad (4)$$

Since power is controlled per voltage plane, $P_{j+1}^{new_S}$ and $P_{j+1}^{new_A}$ are evenly divided by the number of compute units designated for simulation and analysis, respectively.

To account for the new power values being below or above what the hardware supports, we set a δ_{min} and δ_{max} corresponding to the lowest and highest supported power. If the simulation nodes are below δ_{min} (above δ_{max}), they are set at δ_{min} (δ_{max}) and analysis nodes at remaining power, and vice versa. In case of a tie, handling δ_{max} takes priority.

B. Harnessing Application Developers' Knowledge

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

First, to allocate power between the corect entities, a process' identity as simulation or analysis must be supplied. In-situ frameworks already make this distinction typically using sub-communicators. As only process membership is relevant, this information enables SeeSAw to navigate complex process organization in such frameworks with potentially many MPI sub-communicators.

In addition, SeeSAw must be invoked prior to a synchronization or point of communication between the simulation and analysis partitions. This enables differentiating independent work from communication between simulation and analysis, and correct power and time characterization of these events.

As illustrated in the case of LAMMPS in Section VI-C, the two requirements can be satisfied in just two lines of code. We make SeeSAw available[1] as part of an application-level power management library called PoLiMEr, discussed in Section VI-B.

V. LAMMPS: Overview

Molecular dynamics (MD) simulations explore the physics and chemistry governing systems such as liquids, biomolecules and materials. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) is a classical MD simulation and a key workload on HPC supercomputers [33], [34] cited in over 10,000 publications [35], and used by thousands of users world wide. It can simulate different systems containing different types of particles (atoms, molecules, ions, etc.) divided into sub-volumes assigned to individual MPI ranks.

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

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

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. At each Verlet 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 for our evaluation, followed by details of the necessary 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. Theta's nodes support power capping and monitoring via Intel's RAPL interface [39], accessible to users through the msr-safe module [40].

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]. 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. To measure simulation and analysis time and power for each interval between w synchronizations, we use the time of the slowest simulation (analysis) ranks, including the time to perform the power allocation, and the sum of power measurements from all simulation and analysis nodes.

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

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

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.

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.

C. Software Setup: LAMMPS

We use a custom benchmark for LAMMPS from prior works [17], [36], [37] simulating a box of water molecules solvating two types of ions. We consider analyses commonly used in scientific computing [44], [45] and as case studies [17], [18], [36]:

- 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, MSD2D is mostly memory-intensive (less than MSD), RDF is compute bound but with higher memory needs than VACF and MSD1D, both having 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.

We modified the LAMMPS *Splitanalysis* extension to initialize PoLiMEr's power management capability:

```
//universe->uworld : MPI communicator for all
  processes or MPI_COMM_WORLD
//universe->me : rank of current process
//master : 0 if simulation, 1 if analysis
//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. We then reallocate power before synchronization:

```
poli_power_alloc();
//synchronization
```

Detailed implementation documentation is provided in our code (see Section IV-B).

VII. Evaluation

We evaluate SeeSAw against the power- and time-aware approaches relative to the static baseline. The baseline equally divides the global power budget between simulation and analysis nodes. The power cap per node remains fixed (static) and is maintained by RAPL on our evaluation platform. We use the following parameters and settings in our experiments:

- dim – The problem size in LAMMPS is the number of atoms replicated within the simulation cube with dimension dim. Our benchmark has 1568 atoms, so the total number of atoms is $1568 \times dim^3$.
- j – LAMMPS parameter specifying after how many Verlet steps simulation and analysis synchronize. When $j = 1$ they synchronize at each step, otherwise Steps 2–4, 5 and 7 from Section V are skipped until every j^{th} step. We use a total of 400 Verlet steps.
- w – SeeSAw parameter specifying after how many synchronizations power should be allocated. Because the evaluated strategies allocate power at different intervals, for a fair comparison we set $w = 1$ so that each power management strategy is invoked at every synchronization.
- Global power budget of $110 \times n$ W, where n is the total number of nodes of the LAMMPS job. Section VII-D explains our choice of power cap.

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

A. Variability of Results

HPC systems are subject to run-to-run variability [32]. Table I shows the variability of 7 LAMMPS runs for different power caps. Since variability is low on repeated runs on the same node, to reduce the job sizes of our experiments, we compare the differences in runtime between jobs containing two runs: the power management algorithm of interest and the baseline. Simulation and analysis rank placement on nodes and cores is identical in both runs.

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 (1 s on Theta). A short-term power cap allows for brief (9.766 ms on Theta) violations of the long-term power cap. Capping both long- and short-term power 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. In contrast, SeeSAw does not depend on the application reaching the power cap. For or a fair evaluation, however, and to reduce variability, we show results with 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, problem sizes, and scales shown in Figure 3. Full MSD includes three components: MSD1D, MSD2D, and a final averaging of all particles. The *all* category includes RDF, MSD1D, MSD2D, final MSD averaging in case of full MSD, and VACF, executed in sequence at each synchronization. The full MSD analysis is a high-demand workload compared to VACF and RDF and its subcomponents MSD1D and MSD2D. Due to its high memory needs it is limited to problem size with $dim = 16$. In addition, for a comparison of full MSD and its subcomponents, we set $dim = 16$ for MSD1D and MSD2D.

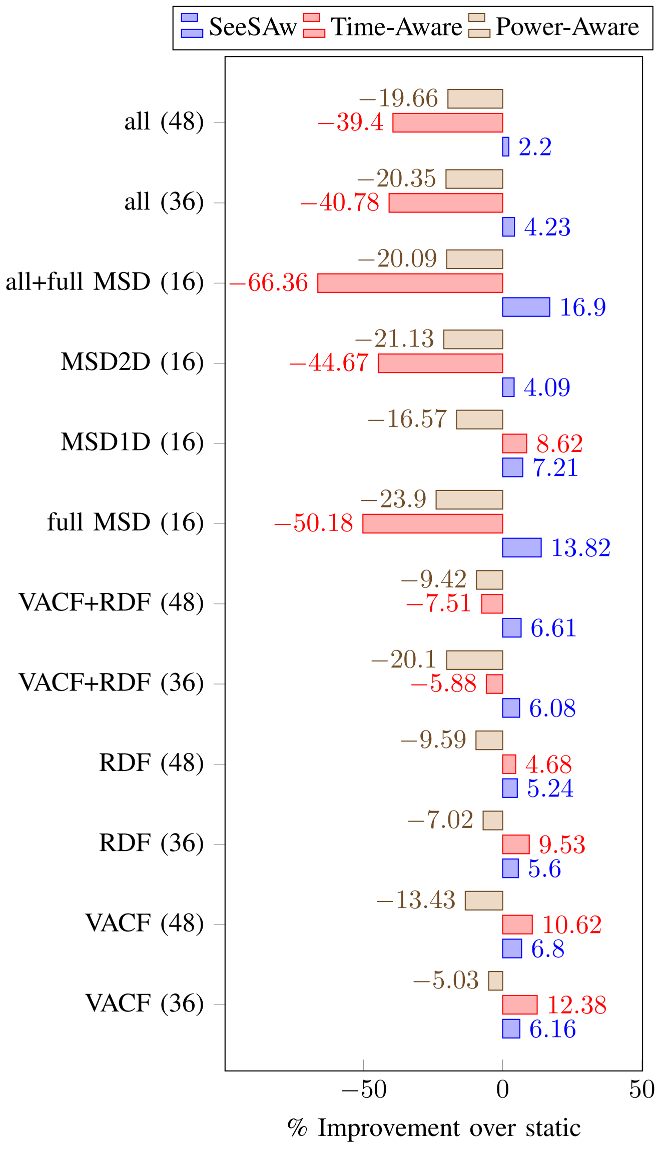
We observe the following: 1) SeeSAw outperforms with high-demand analyses, 2) the time-aware approach is competitive with low-demand analyses, 3) scale is a dominating factor over problem size for effectiveness of power management.

1) Impact of High-Demand Analyses: We demonstrate the performance differences between the three approaches on LAMMPS with full MSD in Figure 4 which shows the power allocated per node. The slack time (black graphs) is the difference between simulation and analysis time between synchronizations normalized to the total time the slowest process took to reach the next synchronization. For reference, Figures 4d and 4e show baseline simulation and analysis time and power between the first 10 synchronization points with 110 W power cap per node. In our implementation step 0 is ignored because it is outside of the main simulation loop. In the first couple steps the simulation has extra setup overhead, which is consistent in repeated runs with MSD. Onward MSD and LAMMPS are nearly identical in runtime with 4 seconds between synchronizations, whereas VACF, RDF, MSD1D, and MSD2D are 2–4$\times$ faster than simulation.

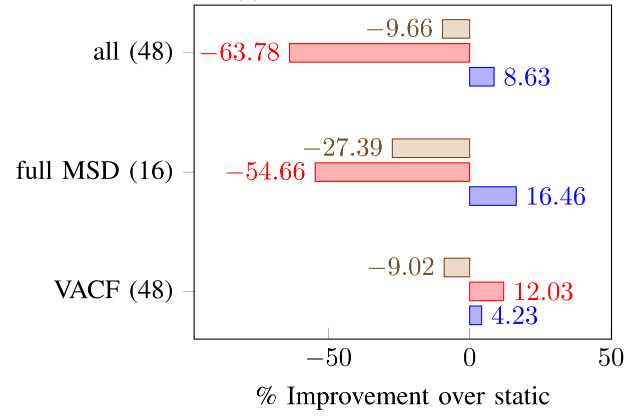
Figure 4a shows that SeeSAw settles on a power distribution within the first 20 steps, assigning analysis more power and bringing the slack time down to an average of 0.8% (calculated from the 10^{th} step).

By only referencing time, the time-aware approach can move power in the wrong direction and not correct the power allocation later on. This strategy picks a direction and gradually increases the gap in the power distribution in that direction. 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 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

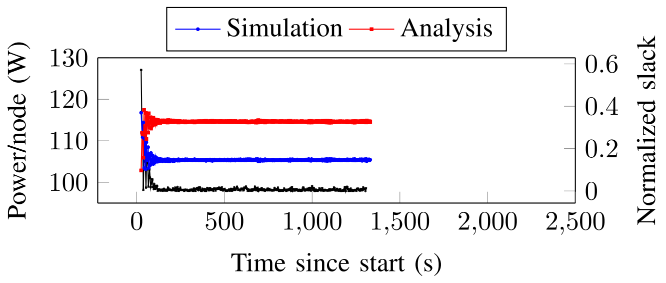


(a) 128 nodes

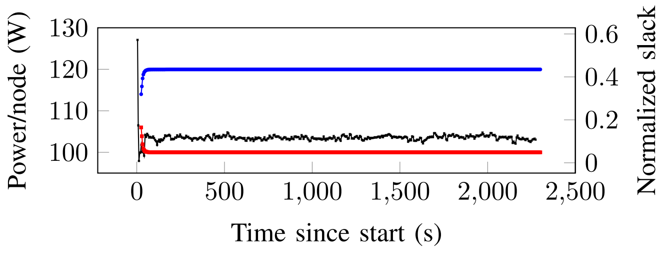


(b) 1024 nodes with representative workloads

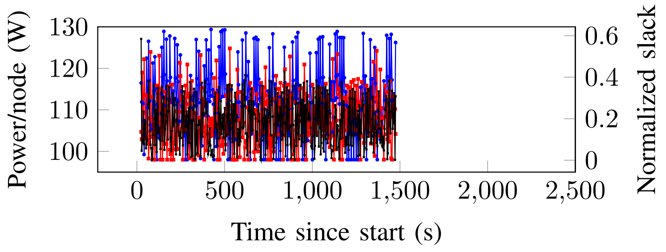
Fig. 3: Performance of SeeSAw, time- and power-aware approaches for different analyses, $w = 1$, $j = 1$. Median of 3 runs shown for each bar.



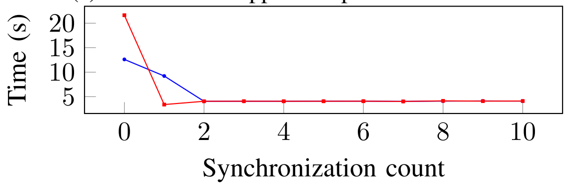
(a) SeeSAw: power allocated over time



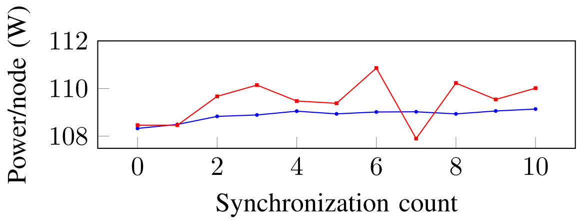
(b) Time-aware approach: power allocated over time



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

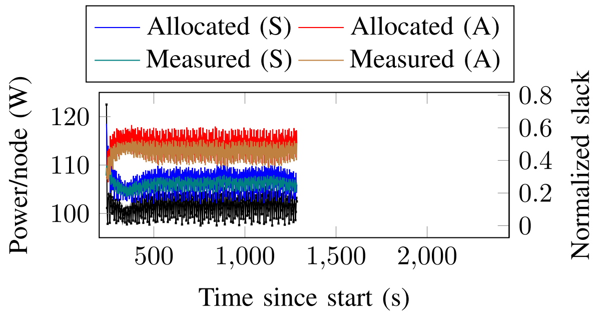


(d) Time between first 10 synchronizations for baseline

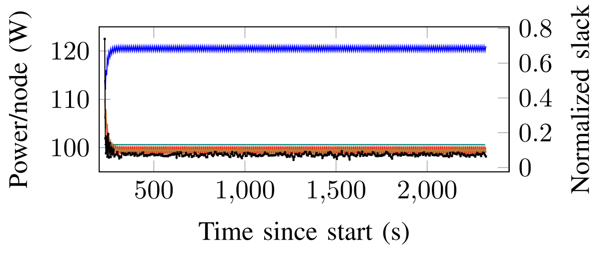


(e) Power between first 10 synchronizations for baseline

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 relative to the total time interval between synchronizations. Bottom two charts show time and power between first 10 synchronizations for simulation and analysis without any power management.



(a) SeeSAw



(b) Time-aware approach

Fig. 5: Power measured compared to allocated between synchronizations by SeeSAw and time-aware approach at 1024 nodes. Normalized slack shown in black.

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.

2) Impact of Low-Demand Analyses: 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 with LAMMPS+RDF and LAMMPS+VACF. These low-demand analyses do not benefit from more power, while the simulation does. SeeSAw and the time-aware strategy perform differently, due to settling at different power distributions. The time-aware approach settles 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 in-situ analysis where nodes do not have equal amount of work.

3) Impact of Scale: At larger scales an additional deciding power allocation factor are utilization limits due to communication overhead. Figure 3b shows the performance difference between the three power allocation algorithms in case of full MSD, all analyses and VACF, chosen as representative workloads out from Figure 3a. Figure 5 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 5a SeeSAw allocates 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.

Despite the normalized slack time in Figure 5b being nearly 0, the time-aware approach causes severe performance degradation. The time-aware approach chooses the wrong direction, increasing the gap in power distribution until reaching δ_{min}. Because the analysis is capped at δ_{min} and due to longer but low-power communication-intensive phases, the simulation is forced into a low-power state. As both tasks 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 and power feedback must be considered as well.

C. Sensitivity Analysis

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

1) Impact of SeeSAw Window Size and LAMMPS Synchronization Rate: Figure 6 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. 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 potential anomalies, so choosing $1 < w < 10$ 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. Optimal settings of w are empirically determined and may differ for different applications.

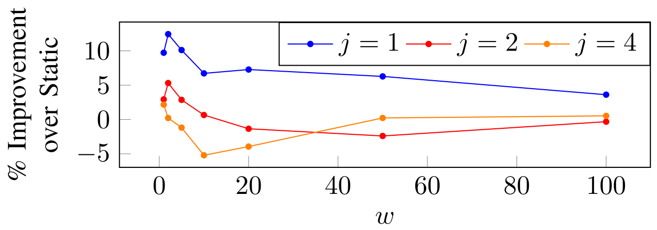


Fig. 6: SeeSAw w and LAMMPS synchronization rate j on 1024 nodes, $dim = 48$, mix of analyses.

2) Mixed Analysis Intervals: Different analyses can be configured to run at different time steps. Table II shows the impact of varying the frequency of a high-demand and low-demand analysis while the rest remain constant. One run varies full MSD while RDF and VACF synchronize with the simulation at each time step, another varies VACF while full MSD and RDF run every step. Power is allocated at every step. As the frequency of MSD decreases, and by fixing $w = 1$, SeeSAw is too reactive to the now anomalous MSD analysis. VACF, on the other hand, is a low-demand workload and does not skew SeeSAw's measurements. When properly tuned, SeeSAw tolerates variable analysis frequencies well. For high-demand but infrequent analyses, setting $w = 2$ or higher will not trigger sudden changes in power allocation.

TABLE II: SeeSAw runtime improvement over the baseline on 2 runs of LAMMPS with RDF, full MSD and VACF: 1) frequency of only full MSD is varied, 2) only VACF is varied. The rest synchronize at each time step. Median of 3 runs on 128 nodes shown, $w = 1$, $dim = 16$.

j	4	20	100
MSD % improvement over static	5.03	0.94	0.90
VACF % improvement over static	16.76	15.09	16.24

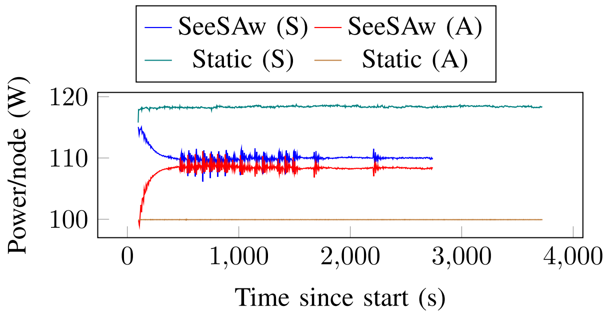
3) Unbalanced Initial Power Distribution: Finally, we consider different initial power distributions between simulation and analysis to reflect differences in power requirements between the two if, for instance, they are given different resources. Figure 7 shows that SeeSAw improves performance in such cases in comparison to keeping simulation and analysis at the initial power distribution. To account for noisy measurements, we chose $w = 2$. The median of three runs shows performance improvement of 28.26% when simulation starts with more power, 19.21% when analysis starts with more power, and 8.94% when the two start with the same power. In Figure 7b SeeSAw improves power utilization of the analysis, which in the static baseline does not utilize the assigned 120 W per node because it depends on the now much slower simulation restricted to run at 100 W per node.

D. Diminishing Returns with More Power Headroom

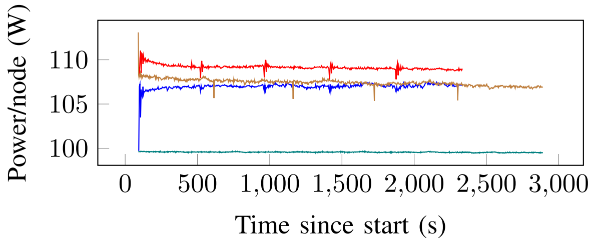
To justify our choice of 110 W in our evaluation, we preset Figure 8 which shows that SeeSAw is more effective at tighter power budgets. There are limited benefits with more liberal power budgets as LAMMPS fails to utilize additional power beyond 140 W per node. The minimum supported power cap by RAPL on Theta's nodes is 98 W, at which application performance is significantly reduced and run-to-run variability increases. For a stringent power cap that is not too close to the minimum, we chose 110 W. The trend shown in Figure 8 is consistent for different analyses, but the exact improvement over the static baseline varies across different power caps with highest improvements in the 110-120 W range per node.

E. Overhead

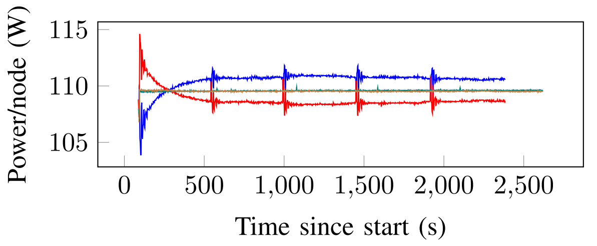
Overhead of computing a new power allocation with SeeSAw is primarily due to communication costs from exchanging power and time measurements between the PoLiMEr ranks, whereas RAPL requires 10ms to react to new power cap requests on Theta's CPUs. We report overhead as relative cost to LAMMPS in Figure 9a at the end of each synchronization, and the absolute time of stand-alone execution of SeeSAw



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



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



(c) Simulation and analysis both start at 110 W

Fig. 7: Simulation (S) and analysis (A) power traces with SeeSAw and static baseline with different initial power distributions. LAMMPS run on 128 nodes as three different jobs, all analyses, $dim = 36$, $w = 2$, $j = 1$.

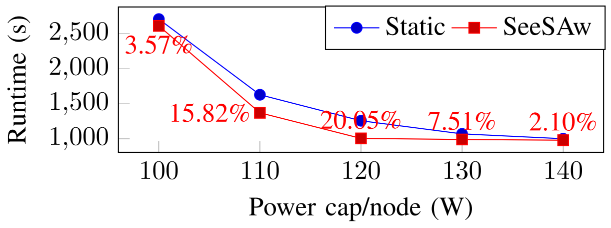
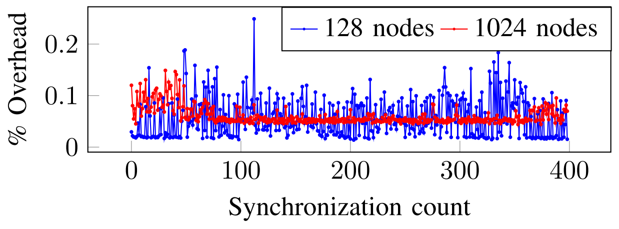
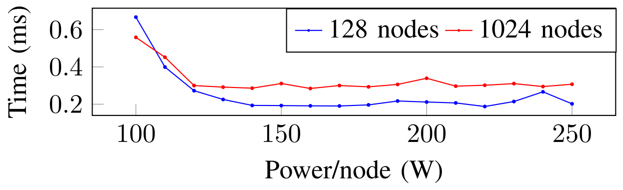


Fig. 8: Improvement of SeeSAw over static baseline for varying power caps per node. LAMMPS is run with all analyses with full MSD on 128 nodes, $dim = 16$, $w = 1$, $j = 1$. Each point is the median total runtime of 3 LAMMPS.

in Figure 9b. Communication costs dominate at 1024 nodes, causing a smaller relative overhead, but higher overhead in absolute measurements. Furthermore, communication costs depend on the underlying interconnect which is optimized for collective MPI communication routines on Theta. Nonetheless, overhead of allocating power itself is incorporated in the time and power measurements by SeeSAw.



(a) Overhead of SeeSAw as a percentage of total time at each synchronization in LAMMPS with all analyses on 128 and 1024 nodes, $dim = 48$, $w = 1$, $j = 1$.



(b) Average duration of SeeSAw in a loop of 10 iterations across different power caps on the Theta supercomputer.

Fig. 9: Overhead measurements of SeeSAw

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.

There are several avenues for future work. Methods to overcome local optima could be explored for more performance gains with low-demand analyses. To add support for heterogeneous hardware within the simulation (analysis) partition, power should be allocated through a hierarchical decision-making process that breaks down SeeSAw's power allocation to the individual compute units, or to different analyses when they are invoked at mixed intervals. Furthermore, SeeSAw could be 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.

Acknowledgements

We thank the anonymous reviewers for their insightful feedback. This research is primarily supported by a DOE Early Career award. Additional support comes from NSF (CCF-1439156, CCF-1823032, CNS-1764039) and the Proteus project under the DARPA BRASS program. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. We thank Preeti Malakar and Christopher Knight for their guidance on in-situ analysis and LAMMPS.

References

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

[2] V. Sarkar *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 *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 *et al.*, "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 *et al.*, "Datastager: scalable data staging services for petascale applications," *Cluster Computing*, vol. 13, no. 3, pp. 277–290, 2010.

[7] U. Ayachit *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 *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 *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 *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 *et al.*, "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 *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 *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 *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] ——, "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 *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 *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 *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 *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 *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 *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 *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 *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 *et al.*, "I/o aware power shifting," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 740–749.

[28] 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*. ACM, 2015, pp. 121–132.

[29] 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*. IEEE Press, 2014, pp. 807–818.

[30] G. Demirci *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] ——, "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 *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 *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] ——, "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 *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 *et al.*, "Whitelisting msrs with msr-safe," in *3rd Workshop on Exascale Systems Programming Tools, in conjunction with SC14*, 2014.

[41] I. Marincic *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 *et al.*, *Computer simulation of liquids*. Oxford university press, 2017.

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