Energy-Efficient and Power-Constrained Techniques for Exascale Computing

Area Exam Position Paper

Stephanie Labasan
Department of Information and Computer Science
University of Oregon

Abstract— **The future of computing will be driven by constraints on power consumption. Achieving an exaflop will be limited to no more than 20 MW of power, forcing co-design innovations in both hardware and software to improve overall efficiency. On the hardware side, processor designs are shifting to many-core architectures to increase the ratio of computational power to power consumption. Research and development efforts of other hardware components, such as the memory and interconnect, further enhance energy efficiency and overall reliability. On the software side, simulation codes and parallel programming models will need modifications to adapt to the increased concurrency and other new features of future architectures. Developing power-aware runtime systems is key to fully utilizing the limited resources. In this paper, we survey the current research in energy-efficient and power-constrained techniques in software, then present an analysis of these techniques as they apply to a specific high-performance computing use case.**

Keywords— **High-performance computing, HPC, dynamic voltage and frequency scaling, dynamic voltage scaling, dynamic concurrency throttling, DVFS, DVS, DCT, energy-efficient, power-efficient, power-aware, power-constrained**

I. Introduction

Power is a key challenge for achieving the next generation of computing. Simply scaling current technologies to exascale will produce a machine that consumes over a staggering 100 MW of power. Such a power-hungry machine results in unsustainable energy utility costs. The U.S. Department of Energy (DOE) has set a challenging goal of reaching an exaflop at 20 MW, requiring supercomputers to stay at or below the power bound for cost feasibility [1], [2], [3]. Said another way, transitioning a present system to exascale requires computational power be increased by a factor of 10, while power requirements be increased by only a factor of 1. The future of computing will challenge the community to innovate all system domains, including hardware, parallel programming models, runtime infrastructures, and simulation codes.

slabasan@cs.uoregon.edu

Addressing the power challenge will require co-design solutions in both hardware and software computing aspects. Processor designs are shifting towards many-core and accelerator-based architectures to achieve higher performance at a lower power consumption. Similarly, more power-efficient subcomponents — memory, interconnect, etc. — are being considered in support of the overall power goal. The entire software stack is also being reconsidered with power in mind. Simulation codes and parallel programming models are being modified to fully leverage the features of upcoming hardware architectures, such as increased concurrency and reduced memory per core. Power-aware runtime systems are under development to dynamically reallocate scarce resources such that performance is optimized.

Additional solutions to reaching exascale are software techniques for savings in energy and power in HPC. Traditionally, energy-efficient techniques are used to evaluate the trade-offs between performance, power savings, and energy savings. Lower processor operating frequencies and voltages result in reduced power consumption, but the impacts on performance and savings in power and energy are highly volatile. At exascale, power is the limited resource, necessitating effective power management. Future research in power-constrained HPC will need to understand the trade-offs between the complex measures of performance — time-to-solution, power consumption, energy usage, accuracy of results, etc. — such that the ultimate goal of maximizing performance under a power bound is met.

In this paper, we survey several energy-efficient and power-constrained approaches in the context of HPC. This paper presents a qualitative analysis of such approaches in order to understand what opportunities exist for designing a power-aware visualization framework.

Terminology	Symbol	Definition
High-performance computing	HPC	Technology leveraging high computational capacity of parallel processing.
HPC system	—	Computing system containing parallel compute resources to provide high computational capacity.
Supercomputer	—	Specific class of HPC system containing a large number of compute nodes.
Floating-point operations per second	FLOPS	Standard metric for measuring performance of supercomputers. Used to rank the top 500 supercomputers in the world.
Energy	—	In physics, energy is defined as the capacity to do work.
Power	—	In physics, power is the rate at which work is applied. It is the derivative of energy usage over time.
Power bound, Power cap	—	Upper limit on power consumption for a particular domain. Power bounds are imposed hierarchically from the supercomputing facility down to the individual processors.
Power-constrained	—	Configuration where power is the limited resource.
Energy-constrained	—	Configuration where energy is the limited resource.
Dynamic voltage and frequency scaling	DVFS	Dynamically change voltage and CPU frequency to reduce power.
Dynamic voltage scaling	DVS	Dynamically change voltage, keeping CPU frequency constant.
Dynamic frequency scaling	DFS	Dynamically change CPU frequency, keeping voltage constant.
Dynamic concurrency throttling	DCT	Dynamically change the number of threads used in a parallel region.
Peak power	—	Maximum power a component can consume.
Thermal design power	TDP	Manufacturer-specified maximum power consumption given a typical application. Often used interchangeably with peak power.
Worst-case provisioned	—	Traditional approach in HPC where all components in the system are able to consume peak power.
Hardware overprovisioning	Overprovisioning	Possible solution for future systems, whereby more components exist than can be simultaneously executing at peak power without exceeding specified power bounds.
Power-aware	—	Objective focused on maximizing performance under a power bound.
Temporal granularity	—	Power monitoring objective specifying the (time) frequency of measurement samples.
Spatial granularity	—	Power monitoring objective specifying the resolution of per-component measurement samples.
Bulk-synchronous applications	—	Massively distributed parallel applications with global synchronization points at key milestones in the computation. The computation cannot move forward until all compute units have reached the synchronization point.
Phase	—	Particular region in an application with uniform execution characteristics.
In situ	—	Scientific visualization model where the data exploration and analysis occurs alongside the simulation.

TABLE I

SUMMARY OF TERMINOLOGY USED IN THIS PAPER.

The rest of this paper is organized as follows:

- Section II provides the necessary foundation for understanding the material in this survey paper, such as challenges at exascale and the differences between energy and power.
- Section III discusses the different classifications of computing and their overarching goals to be energy-efficient or power-aware.
- Section IV provides a comprehensive overview of offline and online DVFS approaches used to reduce energy usage.
- Section V discusses current research endeavors targeted at the power-constrained environment at exascale.
- Section VI discusses present efforts looking at scientific visualization with power consumption in mind.
- Section VII reviews the findings of this survey paper and discusses future work.

II. Background

A. High-Performance Computing

High-performance computing (HPC) is an enabling technology allowing domain scientists to solve a wide variety of complex scientific problems through simulations. Simulations are a compelling alternative to constructing physical experiments as it may be infeasible to build the actual system due to safety or cost reasons, for example. Applications codes facilitate a mechanism for simulating complex real-world phenomenon, often requiring high computational throughput. Typically, these codes require more than one order of magnitude more computational resources than are available on a desktop or workstation, so we look to HPC systems to satisfy computational demands. A specific class of HPC systems are supercomputers, which are large machines consisting of tens of thousands of interconnected compute nodes providing high computational capacity to collective solve the problem at hand.

The performance of supercomputers is often measured by the number of floating-point operations completed in a single second (FLOPS). The IBM Roadrunner transitioned the community from teraflop to petaflop in 2008, capable of completing 10^{15} (or 1 quadrillion) operations per second. The next generation of supercomputers, expected in 2023, will be three orders of magnitude more powerful than Roadrunner, and will expose new challenges related to power consumption.

Twice a year, the Top500 [4] ranks the most powerful systems in the world by the performance achieved on a standardized benchmark known as Linpack [5]. Linpack is a dense solver of linear equations that has been heavily optimized to stress all system components from the processor to the memory subsystem to the interconnect. As of June 2016, the most powerful system in the world is the Sunway TaihuLight located at the National Supercomputing Center in Wuxi, which achieves just over 93 petaflops and consumes 15 MW of power [4]. Scaling current technologies from 93 petaflops to an exaflop will require just one more order of magnitude increase in computational power.

There are several challenges in achieving an exaflop, such as reduced memory per processor, increased memory latency, and higher failure rates to name a few [6]. In 2010, the DOE projected the first exascale system to arrive in 2018. Table II summarizes the expected design changes from a petascale system in 2010 to an exascale system. Due to power consumption limitations, previous supercomputing design trends cannot be applied to transition from petascale to exascale. We discuss this power challenge in more detail in the following subsection.

A.1 Power Challenges

Historically, HPC has largely focused on building the fastest system to achieve the highest job throughput. As a result, computer architecture trends show higher clock frequencies and increased hardware parallelism in the form of transistors, cores, and nodes (Moore's Law [7]). While this trend continues to dominate the chip industry, as of late, it has brought rise to some critical problems. Higher clock frequencies and more hardware parallelism increase the amount of heat generated due to transistor switching. This heat must be dissipated to keep components at a reasonable temperature and to avoid hardware failure. This translates to a high power consumption, as power is directly correlated to the amount of heat generated. The higher the power consumption, the higher the energy usage, and the more expensive the annual operational cost to the supercomputing center. At high power consumption rates, the operational cost becomes expensive and unsustainable for a supercomputing center to afford. The U.S. DOE imposed a maximum power consumption rate of 20 MW for a first-generation exascale system [1], [2], [3]. Assuming 1 MW of power costs a supercomputing center $1M annually in energy bills, the U.S. DOE

Parameter	2010	2018	Ratio
System Peak	2 PF	1 EF	500
Power Consumption	6 MW	20 MW	3
System Memory	0.3 PB	10 PB	33
Node Performance	0.125 GF	10 TF	80
Node Memory Bandwidth	25 GB/s	400 GB/s	16
Node Concurrency	12 CPUs	1K CPUs	83
Interconnect Bandwidth	1.5 GB/s	50 GB/s	33
System Size (Nodes)	20K	1M	50
Total Concurrency	225K	1B	4,444
Storage	15 PB	300 PB	20
I/O Bandwidth	0.2 TB/s	20 TB/s	100

TABLE II
PREDICTED EXASCALE DESIGN TARGETED FOR 2018 (NOW TARGETED FOR 2023), AND THE RESULTING CHANGE FROM A PETASCALE SYSTEM IN 2010. THE LARGE IMBALANCE BETWEEN PERFORMANCE AND POWER AT EXASCALE IS HIGHLIGHTED IN RED [1].

has essentially imposed an annual budget of $20M on power consumption. Comparing a petascale machine in 2010 to an exascale machine in 2023, this means power consumption must only increase by one order of magnitude, while performance will increase by three. Thus, solutions will need to be 100X more power-efficient at exascale than petascale.

In 2007, the Green500 list [8] was created to complement the Top500 list. It ranks supercomputers on the metric of performance-per-watt (*i.e.,* FLOPS/W) instead of performance (*i.e.,* FLOPS). The complementary list promotes new metrics of performance, such as performance-per-watt and energy-efficiency for improved system reliability. With the system-wide power cap implemented by the U.S. DOE, research will need to look into more sophisticated power management strategies at all levels of the system.

Looking forward to the next generation of computing, power consumption will drive the operation and design decisions of these systems from hardware architecture and system design to application development. The challenge will be how to maximize performance of the system while operating under a system-wide power budget.

B. Energy vs. Power

Energy and power are two (often misused) terms related to how much work has been done. We explore the subtle differences in the following subsections.

B.1 Energy

Energy, commonly expressed in Joules, quantifies the amount of work done. It can also be expressed in Watt-hours or more commonly, kiloWatt-hours (1 kWh = 3.6 MJ). An electric company charges consumers for the amount of energy used in a given month. As an example, say a consumer used 910 kWh in the month of July. The electric company charges $0.12/kWh, so the total charge to the consumer would be 910 kWh $\times$ $0.12/kWh = $109.20.

Calculating the energy usage from a time-power curve by is done by taking the integral of the power consumption over time, or equivalently, by calculating the area under the curve as shown in Equation 1 and Fig. 1 (shaded in red), respectively.

$$E = \int_0^\infty P dt \tag{1}$$

B.2 Power

Power, on the other hand, is a rate expressed in Joules per second (J/s). This quantity represents how much work is done in a given interval of time. Calculating the power consumption from a time-energy curve is done by taking the derivative of energy over time as shown in Equation 2:

$$P = \frac{dE}{dt} \tag{2}$$

At the hardware level, the total power consumption of a device is composed of leakage and dynamic power, shown in Equation 3:

$$P_{total} = P_{leakage} + P_{dynamic}, \tag{3}$$

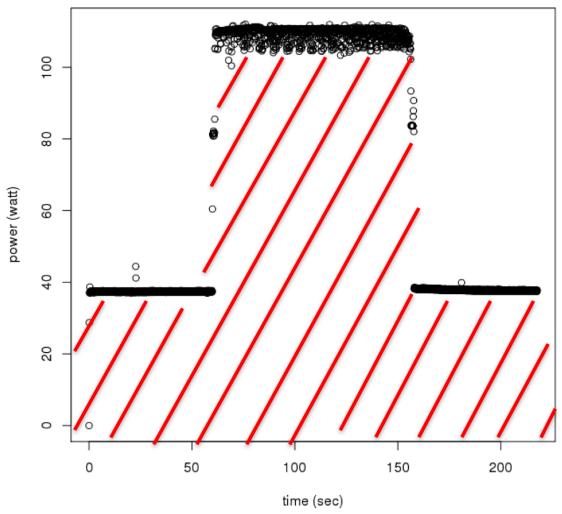

Fig. 1
POWER PROFILE FOR LINPACK EXECUTING ON A SINGLE NODE. THE ENERGY USAGE OF THIS EXPERIMENT IS DETERMINED BY CALCULATING THE AREA UNDER THE POWER VS. TIME CURVE.

where the leakage power is power lost while the device is in steady-state (*i.e.,* idle state), and the dynamic power is attributed to the switching of transistors. $P_{dynamic}$ is the dominating factor in Equation 3, and is dependent on the given application and current thermal environment. Fig. 1 shows a real-world power profile for the Linpack benchmark running on a single node. For the first 50 seconds of the experiment, the power consumption is at 40W. This is the minimum power necessary to run the system in its idle state. Once the application begins, more system components become engaged, such as the cache, cooling infrastructures, and arithmetic units, and the power consumption rises to 110W. Once the application has completed, the system returns to the idle power of 40W.

Song et al. [9] further break down the components in Equation 3 as:

$$P = I_{leakage}V_{dd} + ACV_{dd}^2 f, \quad (4)$$

where AC is the capacitance of the device, V_{dd} is the supply voltage, and f is the clock frequency. Equation 4 implies that the total power consumption can be managed by adjusting the clock frequency and/or the supply voltage, which are directly proportional to one another.

C. Power Monitoring Tools

There are a wide range of hardware and software tools for power monitoring. More commonly, these tools are capable of monitoring power rather than controlling power. As such, there is an increasingly high demand for tools that enable power control at fine-grained temporal and spatial granularities in moving forward to exascale, where power will be a severely constrained resource and will need to be allocated wisely. In the following subsections, we will discuss a small subset of these tools at a high-level to provide an overview of the available infrastructures. For more details, see Appendix A. We summarize the different power monitoring solutions in Table III.

C.1 Coarse-Grained Monitoring

Coarse-grained monitoring tools, such as WattsUp? Pro and Cray XC30 measure power consumption with low spatial granularity. The WattsUp? Pro [10] is an external monitor connected between the power supply at the wall and the component. While such a device beneficially provides physical power measurements rather than modeled ones, it has some pitfalls. The first being that it reports the power consumption of the entire system, which does not reveal how the power is being consumed across different components, such as the processor and memory infrastructure. Similarly, Cray XC30 systems have the same shortcomings in that it reports power usage per-node, but does not indicate how power is being consumed by node subcomponents. Measuring power consumption of a large-scale HPC system would render this solution cost-ineffective as multiple meters would need to be purchased. An additional problem is the frequency update rate of twice per second, which will be too coarse-grained to attribute power consumption to particular phases in the application.

C.2 Vendor-Specific Component-Level Monitoring

Processor manufacturers, such as Intel, AMD, and IBM, have released software technologies for measuring, monitoring, and controlling power consumption of the processor and peripherals. Intel's Running Average Power Limit (RAPL) technology [12] was first introduced with Sandy Bridge processors. RAPL provides privileged access to a wide variety of 64-bit model-specific registers (MSRs) to monitor and control various power and thermal features of the processor and pe-

Power Monitoring Tool	Type	Sample Freq	Scope	Power Capping?	Cost	Ref
WattsUp? Pro	Coarse-Grained	0.5 sec	AC outlet	No	$$$	[10]
Cray	Coarse-Grained	100 ms	Server, Node	No	—	[11]
Intel RAPL	Vendor-Specific	1 ms	Core, DRAM per socket	Yes	—	[12]
AMD APM TDP Power Cap	Vendor-Specific	10 ms	Socket	Yes	—	[13]
IBM EnergyScale	Vendor-Specific	10 ms	Server	Yes	—	[14]
NVIDIA NVML	Vendor-Specific	16 ms	GPU	No	—	[15]
PowerPack	Fine-Grained	1 sec	DC components	No	$$	[16]
Penguin PowerInsight	Fine-Grained	1 ms	DC components	No	$	[17]
PowerMon PowerMon2	Fine-Grained	0.3 ms-20 ms	DC components	No	$	[18]
Ilsche et al.	Fine-Grained	1 ms	DC components	No	$$$	[19]
HDEEM	Fine-Grained	1 ms	DC components	No	$$	[20]

TABLE III
SUMMARY OF SOME AVAILABLE POWER MONITORING TOOLS.

ripherals. One of these features is predicting the power consumption with internal models based on counters in order to make decisions on how to spend any available power headroom wisely. With the introduction of Haswell processors, fully integrated voltage regulators enabled actual power measurements to vastly improve the accuracy of RAPL energy measurements [21].

Fig. 14 highlights four RAPL power domains, which can vary across architectures:

- Package (PKG): All components on the processor
- Power Plane 0 (PP0): Cores and local private caches
- Power Plane 1 (PP1): Uncore devices, such as graphics cards on client processors or QuickPath Interconnect (QPI) on server processors
- DRAM: Memory subsystem

Traditionally, the client processors support the *PKG*, *PP0*, and *PP1* domains, while the server processor supports *PKG*, *PP0*, and *DRAM* domains.

Each RAPL domain has registers for the following parameters:

- Power Limit and associated Time Window for enforcing a power cap
- Energy Status for monitoring energy usage (and deriving power consumption)
- Perf Status for monitoring the performance impact of the enforced power cap
- RAPL Info defines the minimum and maximum power supported by the given domain, the max time window, and the thermal design power (TDP)

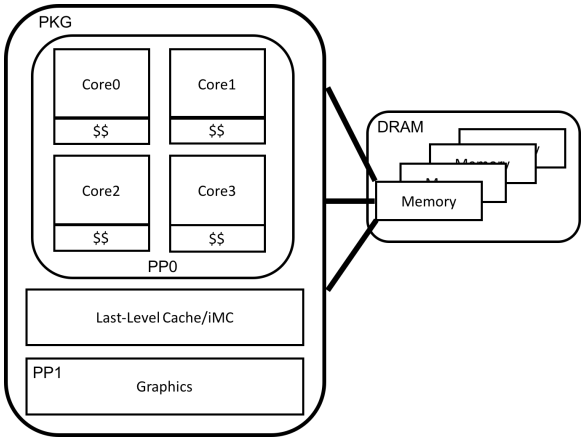

Fig. 2
RAPL DOMAINS FOR A GIVEN PROCESSOR.

Each MSR bit field is expressed in terms of power, energy, or time units, which are fused into a read-only register called `MSR_RAPL_POWER_UNITS`. For most architectures, the power unit is expressed in $\frac{1}{8}W$, the energy unit is expressed in $15.3\mu J$, and the time unit is expressed in $976\mu s$. MSRs in any domain can be accessed with `rdmsr` and `wrmsr`, assuming the `msr` kernel confirms the presence of and appropriate permis-

sions for the `/dev/cpu/X/msr` file.

AMD Application Power Management [13] provides similar functionality to Intel's SpeedStep, TurboBoost, and RAPL technologies. Starting with Bulldozer, AMD introduced TDP Power Cap technology, enabling software to limit power consumption of the processor by enforcing a desired TDP limit. IBM EnergyScale [14] also enables software to monitor and control the power consumption starting with POWER6 systems. One notable feature is processor folding, whereby tasks are migrated among cores to maximize energy usage. For example, if the application demands resources, migrate tasks to a larger number of cores. When the demand decreases, reduce the number of working cores, putting superfluous cores to idle or sleep.

With the growing demand for fine-grained power measurement capabilities, other vendors are releasing software-based solutions to monitor power of targeted components. NVIDIA's Management Library (NVML) provides GPU core temperature, current effective frequency of the GPU, and real-time power consumption measurements [15]. This library is limited in scope, only targeting the server processors, such as the Tesla and Quadro lines. Power readings in milliWatt granularity are gathered with a simple API call.

C.3 Fine-Grained Infrastructures

To overcome the limited information provided by vendors about their respective power management infrastructures, several tools have been created to physically instrument compute nodes and individual components irrespective of the underlying architecture. PowerPack [16] is a hardware and software co-design solution to automate collection of power data from component-level sensors.

Penguin PowerInsight [17] is a commercially-available hardware and software solution designed in close collaboration with Sandia National Laboratory. The hardware sensor modules integrate onto the motherboard by attaching to the voltage rail between the power supply and the motherboard components, such as the CPU or memory.

PowerMon and PowerMon2 [18] are very similar solutions to Penguin's PowerInsight discussed previously. It is a low cost — at most $150 per device — fine-grained power monitoring device for commodity systems capable of collecting data from eight channels simultaneously at a rate of 1000 samples per second. The PowerMon2 was a more optimized solution to its predecessor, the PowerMon, which had six measurement channels and a sample collection frequency of 20 ms. This optimized device provided eight measurement channels to sample additional peripherals, such as GPUs, and a maximum collection rate of 3000 samples per second.

There is a complex relationship between several design goals — scalability, fine-grained per-component measurements, fine-grained power attribution to applications behaviors, and accuracy. Each solution discussed in this section has its benefits and drawbacks, and no one monitoring infrastructure satisfies all of design goals.

III. System Objectives

In this paper, we define two specific types of design constraints: energy-efficient and power-aware. These two constraints are similar, but aim to satisfy two different objectives based on the target domain. This section details the constraints and explains how they impact the design of various computing domains in more detail in the following subsections. The different classifications of computing and their respective design constraints are summarized in Table IV.

A. Energy Efficiency

Designing for energy efficiency has dominated the computing world for several decades. It has been the driving factor behind mobile and desktop computing, where energy efficiency translates into longer battery life (mobile computing) and lower monthly energy bills (desktop computing). In particular, energy efficiency is of concern to areas of computing where the amount of energy is finite. The goal of *mobile computing* is to extend battery life, so the user does not have to recharge the device in the middle of the day. Energy-efficient techniques to maximize battery life include putting the processor and other components to different levels of deep sleep when not in use [22], [23], [24], [25], [26], [27], [28], [29]. On the software side, mobile application developers are finding intelligent ways of minimizing energy usage, further extending the battery life of the device.

Desktop computing architects have adapted techniques from the mobile world, such as putting the processor to sleep when not in use, and have implemented

Computing Classification	Energy- vs. Power-Constrained	Driver	System Scale	Response Time	Perf Degradation Allowed?
HPC	Power-constrained	Organization	Large	Long	Yes
Data Center	Energy-constrained	User	Large	Short	No
Mobile	Energy-constrained	User	Small	Short	No
Desktop	Energy-constrained	User	Small	Short	No
Workstation	Energy-constrained	Organization	Medium	Long	Yes

TABLE IV
COMPUTING TAXONOMY AND THEIR ASSOCIATED ENERGY-CONSTRAINED OR POWER-CONSTRAINED OBJECTIVES.

energy saving heuristics to save energy during computation. DVFS enables dynamic reduction of the current operating frequency and voltage, in order to conserve power or reduce the amount of heat being generated by the chip. Lower frequencies result in performance degradation particularly to CPU-bound applications, since the rate of instruction completion is proportional to the operating frequency of the processor. *Workstations* provide higher computational capacity than desktop computers, and are driven to be energy-efficient to reduce energy utility costs. As such, slowdown to applications are acceptable.

Data centers, such as those housed by Amazon and Google, are another class of computing distinct from those targeted for HPC. The goal of a data center is to provide users with cheap, scalable solutions. Data center applications are transactional — users submit a request to the data center, and the data center is expected to provide an immediate answer. In several cases, data centers are constrained by an energy budget, so measures need to be taken to minimize energy usage [30], [31], [32], [33]. Performance slowdown is not acceptable in this case, as the user can satisfy their demands by submitting their request to another data center.

In an *HPC* setting, minimizing energy usage on a per-job basis is necessary for maximizing job throughput of the machine under power limitations. The central goal of energy-efficient computing is to reduce energy usage, while incurring some slowdown in performance. The slowdown is often acceptable because it reduces energy costs. Lots of research has been dedicated to analyzing the trade-offs between energy and performance in the HPC domain. Jitter [34] and CPU MISER [35] are two algorithms designed to save energy with some impact to performance by leveraging DVFS to exploit application imbalances. Most algorithms are particularly focused on phases where the CPU may be memory-bound and lacking computational work, while others focus on compute resources with less work assigned, as consequently, they will spend time idling at a global barrier waiting for other compute resources to finish. Adagio [36] was the first dynamic runtime system developed to increase energy savings without impacting performance by using previous task performance and energy usage behaviors to influence the subsequent execution of this same task.

With the impending challenges of next generation supercomputers, using solely energy-efficient approaches will not satisfy the overall power bound goals at exascale. That is to say, meeting an energy target will not necessarily guarantee a specific sustained power. Innovations are needed for all components of the HPC system with power in mind.

B. Power-Aware

Power-aware computing is mostly targeted towards the HPC domain, where power constraints will pose a large challenge in achieving the next generation of computing. Up until now, nodes have been the primary scarce resource shared among various jobs in order to increase throughput, and power consumption and energy usage were a secondary concern to performance. At exascale, the U.S. DOE has enforced a 20 MW power cap, bringing power to the forefront as the primary concern.

In order to meet the specified power bound, all components of the HPC system will need to be re-assessed with power in mind. Hardware architectures are being redesigned with power efficiency in mind, paving the way for many-core architectures, such as NVIDIA's GPU and Intel's Xeon Phi. Power measurement tools, such as Intel's Running Average Power Limit, are improving their accuracy and temporal and spatial mea-

surement granularity to monitor the current power consumption of various system components from the socket to the individual cores to the memory subsystem.

Understanding and optimizing HPC application performance under a power bound is a research area that needs continued exploration. Rountree et al. [37] were the first to look at the implications to performance while under a power bound enforced on a per-processor basis. Their work highlights the notable impact of processor manufacturing variations on overall performance when under a power bound, noting that variations in power will no longer be acceptable given a system-wide power bound. In the past, applications have been power-oblivious, and power has been allocated based on maximum capacity. However, this strategy does not always lead to the best performance and research has shown better performance can be achieved with a smaller power allocation [38], [39], [40], [41]. Since power will be a limited resource at exascale, it will be necessary to allocate the available power wisely. Algorithms will need to be re-designed to be power-aware, such that they can dynamically adapt to the available power, though resulting performance will degrade. More specifically, applications will need to understand instances when more power is required to achieve end goals and instances when power and performance could be reduced without impacting accuracy.

A tractable solution to achieving an exaflop under a power bound is *hardware overprovisioning*, where a system cannot run all components at peak power simultaneously without exceeding the power bound [38], [37], [40], [42], [43]. Patki et al. [39] found that an average 50% speedup can be achieved with an overprovisioned system.

C. Energy and Power Management Technologies

Typically, the CPU is the dominating factor in overall power consumption of the system (see Fig. 3). By applying energy and power reduction techniques, such as DVFS, DCT, and power capping to applications, significant savings can result. We detail these different techniques in more detail in the following subsections.

C.1 Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is the generic name for the technology to adjust the clock frequency and supply voltage to control power consumption and energy usage. In other words, a lower frequency reduces power consumption, since the processor is operating at a lower speed and is consequently generating less heat. Assuming DVFS does not degrade the performance to a large degree, this technology has also resulted in a savings in energy usage, which is cost-effective to the supercomputing center. However, it can take several thousand clock cycles for the change in voltage and frequency to take effect, so the time step in which the DVFS is being applied needs to be long enough to amortize this change [45].

Up until now, DVFS has been the primary method for reducing energy usage and operational costs. At exascale, where power is the scarce resource, DVFS will not be sufficient in meeting power goals. DVFS guarantees a specific frequency and voltage, which are discrete pairs fused into the processor during the manufacturing process. But, DVFS does not guarantee a specific power consumption, which is impacted by a myriad of factors including the current temperature of the system and the number of components currently engaged (*i.e.,* application-dependent). In order to meet power constraints, tools are needed to enforce a strict power bound, with the underlying firmware managing the voltage and frequency to guarantee a specific power consumption.

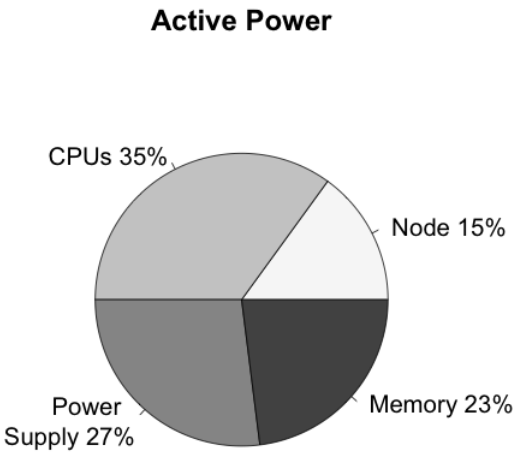

Fig. 3
THE CPU IS A LARGE POWER CONSUMER IN A SYSTEM. THE MAGNITUDE OF POWER IT CONSUMES WILL VARY DEPENDING ON THE APPLICATION BEHAVIORS [44].

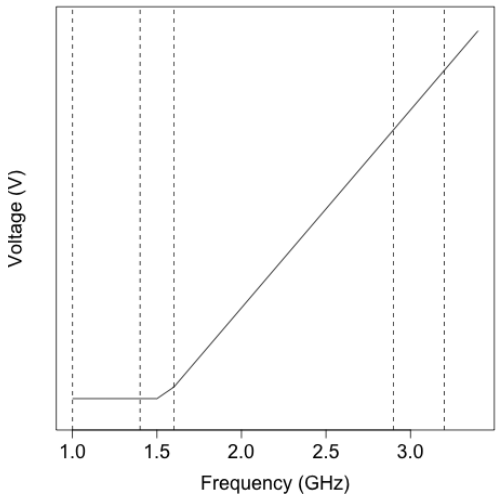

Fig. 4

SAMPLE POWER-EFFICIENT VOLTAGE-FREQUENCY CURVE FOR A CPU. FREQUENCIES BETWEEN 1.0 GHZ AND 1.4 GHZ CAN BE ACHIEVED AT A CONSTANT VOLTAGE VALUE. THE FREQUENCIES BETWEEN 1.6 GHZ AND 2.9 GHZ ARE THE TYPICAL VOLTAGE-FREQUENCY RANGE EXPOSED TO SOFTWARE. FREQUENCIES ABOVE 2.9 GHZ ARE CONSIDERED TURBO RANGE, WHERE EFFECTIVE FREQUENCIES WILL VARY BASED ON THE APPLICATION AND THERMAL CONSTRAINTS.

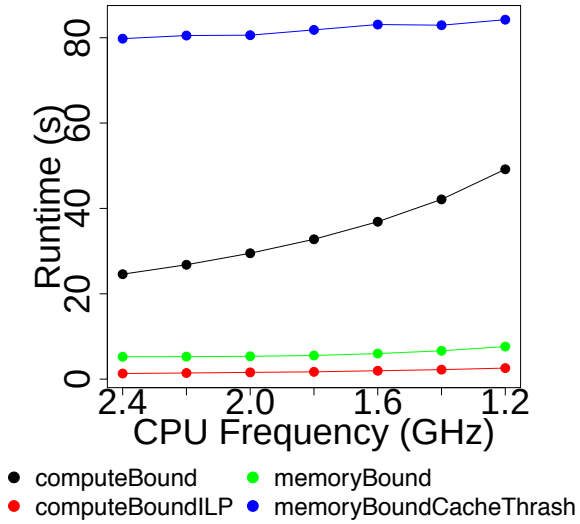

Fig. 5

APPLICATIONS WITH VARYING DEGREES OF COMPUTE AND MEMORY INSTRUCTIONS WILL BE IMPACTED TO DIFFERENT LEVELS BY FREQUENCY [52].

In the literature, this technology has also been referred to as dynamic frequency scaling (DFS) [46] and dynamic voltage scaling (DVS) [34], [36], [47], [48], [49], [50], [51]. In reality, dynamically increasing voltage in DVS (while keeping a constant frequency) is not useful to HPC, but is more a technology found in desktop computing (overvolting). DFS implies a change in frequency without a change in voltage as demonstrated in Fig. 4, but does not necessarily provide any energy or power savings. In this paper, we will refer to this energy- and power-saving technology as DVFS, where both voltage and frequency need to change to result in savings.

Discrete voltage-frequency pairs are known as performance states or p-states. The default frequency or nominal frequency of the processor is known as P1, and results in the best performance, maximum power. The lowest frequency available to software is known as Pn. There are, however, frequencies lower than Pn available to firmware as a protection mechanism in case of thermal events. The impact of different frequencies to performance is heavily dependent on the application as shown in Fig. 5. Reducing the CPU frequency from P1 to Pn for a compute-bound application produces a negative linear correlation between frequency and performance. In the case of the black curve in Fig. 5, a reduction in the CPU clock frequency by two causes a 2X increase in runtime. The remaining curves experience roughly no change in runtime as the frequency changes as these applications were waiting on data to return from memory.

Every processor manufacturer has branded their own version of this technology. On Intel processors, this CPU throttling technology is known as SpeedStep. Intel processors also have a technology known as Turbo-Boost, whereby additional performance can be gained by subsequently running the processor at higher than the normal frequency range when thermal limits allow and when there is a demand for high computational capacity. On AMD processors, it is known as Application Power Management (APM) [13] and TurboCore. On IBM processors, this technology is known EnergyScale [14].

C.2 Dynamic Concurrency Throttling

The number of cores in a processor is increasing exponentially, however, most applications are unlikely to make effective use of the available cores. Dynamic concurrency throttling (DCT) is a software knob for adapting the concurrency level based on application execution behaviors in multi-core processors [53], [54]. DCT can be a better solution to DVFS within a node as some DVFS implementations affect all cores the same (*e.g.,* Intel Ivy Bridge), and they have high overhead in switching from one voltage and frequency to another [55]. While advanced processors have made great strides in providing tools for finer-grained energy efficiency, such as enabling each core to run at a different frequency (*e.g.,* Intel Haswell) and reducing the overhead, DCT provides a throttling mechanism with less overhead.

Combining the techniques of DVFS and DCT into a prediction model is shown to be beneficial in maximizing performance and energy savings [56], [57]. DCT will be another key component in reaching an exaflop at 20 MW. In particular, it will enable future power-aware resource managers to dynamically reconfigure the number of active threads within the job to fit within the available power budget.

C.3 Power Capping

At exascale, systems will be severely power limited, so techniques are needed to manage power consumption across all components in the system. Power capping provides fine-grained software control over the power consumption of the processor, and the underlying hardware uses DVFS to manage the specific voltage and frequency to guarantee the specified power. In 2011, the first component with power capping capabilities appeared on the market. Currently, power capping mechanisms target the CPU, which is typically the dominating factor in node-level power consumption. Processor manufacturers, such as Intel, AMD, and IBM, have provided capabilities to enforce a cap on the power consumption of the processor, and Intel also enables control over the power consumption of the DRAM. Research in power-aware HPC is shifting away from using DVFS to using power capping to manage power usage under a hard power constraint. While understanding performance due to DVFS has been extensively studied, understanding performance under a power cap exposes new challenges concerning the relationship between performance and power consumption.

IV. Energy-Efficient HPC

Research in power-aware HPC has been dominated by trading performance for a savings in energy. Solutions have leveraged DVFS to minimize performance degradation by targeting specific application characteristics, such as communication barriers, I/O delays, load imbalance, or iterative behaviors. While saving energy is necessary to reaching power goals, they will be insufficient as the overarching goal at exascale will be managing power consumption, not energy usage. Initially, offline approaches were proposed, serving as a basis for evaluating future online approaches. Online approaches are challenging, since making decisions requires accurate models and predictions of the impact of different CPU frequencies on application phases without any a priori knowledge of the application. This section will provide an in-depth overview of the previous and current research endeavors of energy-efficient techniques for HPC. First, we discuss various metrics used to characterize applications as compute-bound, memory-bound, or I/O-bound, which identify good candidates for energy savings.

A. Characterizing Applications

Traditionally, applications are categorized into one of three classifications – compute-bound, memory-bound, or I/O-bound — indicating the bottleneck resource. For most processors, a reduction in frequency on a compute-bound workload is inversely proportional to the overall execution time, and for memory-bound and I/O-bound applications, a change in frequency should result in no change to the execution time. For this reason, DVFS algorithms tend to target the latter types of applications, since they provide the most energy-saving opportunities without a time penalty. Predicting the energy-time trade-offs of DVFS can be done by collecting performance metrics to classify each application.

One common metric is instructions per cycle (IPC), or inversely cycles per instruction (CPI), which indicates the CPU-boundedness of an application. The higher the IPC, the more dependent the application is on the CPU, and the more impacted it will be by a change in frequency. As such, IPC varies greatly with frequency, making it a non-ideal candidate for predicting the energy-time trade-off. A more reliable metric is

DVFS Algorithm	DVFS Classification(s)	Scope	Mod Src Code?	Ref
Freeh et al.	Offline	Intra-Node	No	[58], [59]
Cameron et al.	Offline	Intra-Node	No	[48]
Linear Programming	Offline, Slack	Inter-Node	Yes	[60]
Minimization Solution	Offline	Inter-Node	Yes	[49]
Springer et al.	Offline	Inter-Node	Yes	[61]
Green Queue	Offline	Inter-/Intra-Node	No	[62]
Jitter	Regular Iterations	Inter-Node	Yes	[34]
Laurenzano et al.	Regular Iterations	Inter-Node	Yes	[63]
Energy Aware MPI	Non-Regular Iterations, Slack	Inter-Node	No	[64]
CPUFreq	Fixed Intervals	Intra-Node	No	[65]
CPU MISER	Fixed Intervals	Intra-Node	No	[35]
Ge et al.	Fixed Intervals	Intra-Node	No	[51]
Criticality-Aware Governor	Fixed Intervals	Intra-Node	No	[66]
β-Adaptation	Fixed Intervals	Intra-Node	No	[50], [67]
Adagio	Slack	Inter-Node	No	[36]
Thrifty Barrier	Slack	Intra-Node	No	[68]
Lim et al.	Slack	Inter-Node	No	[69]
ORA	Slack	Intra-Node	No	[70]

TABLE V
SUMMARY OF ENERGY EFFICIENCY ALGORITHMS.

misses per operation (MPO), derived from number of instructions retired and last-level cache misses [59], [71]. MPO is indicative of the memory pressure of a given application — as the MPO value increases, the application becomes more memory-bound because the number of memory accesses per operation is increased. Unlike IPC, MPO remains constant with varying frequencies. Executing the same application with different frequencies results in a maximum 1% difference in MPO.

Hsu et al. proposed the β metric to indicate the intensity level of off-chip (*i.e.,* memory) accesses [50], [67]. β is a value between 0 and 1, where 1 indicates that the overall execution time will scale with a reduction in frequency, and 0 indicates that the execution time is independent of the frequency. Computing this coefficient is done at runtime, and is based on the rate of instruction completion (in millions) per second (MIPS) at each CPU frequency:

$$\beta = \frac{\sum_i (\frac{f_{max}}{f_i} - 1)(\frac{mips(f_{max})}{mips(f_i)} - 1)}{\sum_i (\frac{f_{max}}{f_i} - 1)^2} \qquad (5)$$

Here, the total number of instructions in a given application is assumed constant irrespective of frequency, and thus the MIPS value will be proportional to the CPU frequency.

Rountree et al. [72] survey current hardware performance counters to predict application performance sensitivity to frequency. They advocate for an additional counter to track the number of cycles spent executing leading loads. A load is a non-speculative read that results in a last-level cache miss. The leading load distinction identifies the first load. It is assumed that the results of the load will be needed for future instructions, but these are not tracked. To implement the additional counter, an extra bit is added to the load instruction register. The bit is set to 1 when the leading load occurs. The counter is incremented for every cycle that this bit is high. The bit is cleared once the load instruction completes. Leading loads expresses a new relationship between clock frequency, memory accesses, and performance, irrespective of independent application characteristics.

There are several performance metrics that can be

used to form better predictions of the energy-time tradeoffs, but at some point the overhead of collecting an exhaustive set of metrics can outweigh the energy savings.

B. Classifications of DVFS Approaches

This section will define the different classes of DVFS approaches, which we summarize in Table V.

B.1 Offline

Offline approaches are a good first step in determining the potential for energy and power savings with DVFS, though are not ideal for production due to high costs. They are particularly useful as a baseline to compare future DVFS algorithms to as they become more advanced. The large downside of offline approaches is the huge overhead in executing an application at each available voltage-frequency pair for a particular hardware architecture in order to generate a profile of execution time and energy usage for a particular architecture. This section will detail the offline DVFS approaches.

Freeh et al. [58] performed an extensive study on the cross-product of all available CPU frequencies and parallel benchmarks with varying behaviors to understand the energy and performance trade-offs. They ran the NAS parallel benchmark suite [73] at all frequencies on varying number of nodes — the single node results highlight the influence of the memory bottleneck on energy and time, while the multiple node results highlight the influence of the communication bottleneck. For each execution, the power consumption of the entire system is recorded at frequent intervals on external meters, and energy usage is derived by integrating this measurement over the interval time. In addition, the total execution time is recorded. The NAS parallel benchmarks, which vary in terms of communication and computation phases, showcase differences in power and execution time at scale. Communication phases stress different components of the processor than computation phases, resulting in different power distributions. The execution time for computation phases, *i.e.,* the CPU is the bottleneck, scales at a different rate than communication phases when varying the number of nodes. In some cases, energy and time can be reduced by executing the application at a lower frequency on more nodes than executing at a higher frequency on fewer nodes. This is because of delays in the processor due to memory stalls or synchronization points. Executing at a higher frequency during phases with processor delays does not decrease the execution time since the processor is not the bottleneck, but does waste energy. Unfortunately, this approach is very coarse-grained, limiting the amount of energy savings that can be achieved.

Freeh et al. extended on their findings and demonstrated higher energy savings when applying multiple energy gears per phase [59]. Here, an energy gear is simply another term for a discrete voltage and frequency pair, *i.e.,* available CPU frequency. While showing that energy savings could result without degrading performance, there is a large effort up front to collect trace data from each application, then manually instrumented each application into phases. A solution that automatically detects phase changes within the application (from hardware performance counters) and changes frequency appropriately would be better suited for exascale.

Hsu et al. [49] found an optimal frequency schedule per phase that solves the following minimization problem: given an application, identify a phase and a frequency such that if the phase is executed at the specified frequency and the rest of the application is executed at the nominal frequency, the total execution time and the overhead in changing frequencies does not degrade overall performance by the specified threshold. The solution is derived from an offline trace, and the resulting per-phase frequencies are implemented into the application. While the findings are useful, the scope of this study was limited in scale.

PowerPack (see Section II-C.3) is a power framework used to profile, analyze, and control energy usage of an application [48]. The PowerPack software can apply DVFS schemes during slack time in distributed applications to conserve energy, but profiling needs to occur first to determine the potential energy savings. Slack time is defined as the time spent waiting for an MPI communication call to complete due to dependencies, such as receiving data or running a collective. These dependencies may include waiting for data to arrive from another process in an MPI communication call or waiting for the last process to arrive in an MPI collective call. In order to see energy savings, phases need to be long enough to outweigh the overhead of changing frequencies.

Rountree et al. [60] proposed a solution that determines the maximum energy savings for an application. By determining an upper bound, future DVFS solutions can be quantified in terms of proximity to the optimum.

Total energy savings is determined after the comple-

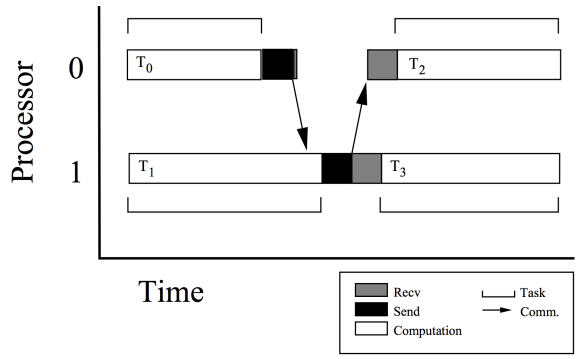

Fig. 6
A TASK IS IDENTIFIED AS A REGION BETWEEN TWO MPI COMMUNICATION CALLS. IN THIS FIGURE, A TASK IS DEFINED BEFORE AN MPI RECEIVE AND MPI SEND CALL [60].

tion of the application, so this solution maximizes its energy savings by reclaiming slack time across all processors, particularly during memory- or communication-bound phases. This approach includes three steps. The first step generates a communication trace for an application at all available CPU frequencies. These traces provide the execution time and power consumption of each task at every available frequency, which are often application-dependent. Here, a task is defined as the region between two MPI communication calls, such as a receive and send call, shown in Fig. 6. At the nominal or highest frequency available, the per-processor timestamp, source, and destination for every MPI communication call are recorded.

The second step in this approach combines all the resulting traces into a system-wide task graph and transforms the graph into a linear programming (LP) matrix. Using the LP matrix, the maximum execution time for the application, and the execution time for each task at each frequency as input, a frequency schedule for each task is outputted. The schedule defines how long each task should run at a given frequency in order to minimize overall energy usage. For tasks with a very short execution time, the default solution is to run it at the nominal frequency as there is a penalty for changing frequencies, which in turn, can negatively impact overall application performance. The third and final step is to validate the resulting frequency schedule by re-running the same application using this schedule, and changing the frequency for each task.

One obvious limitation of this approach, and common to all offline approaches, is the high overhead of executing the applications at each frequency to generate a profile of various information to determine an optimal energy savings solution. The number of application executions scales linearly with the number of available CPU frequencies, which can result in a long first step. This high up-front cost keeps this solution from being used in production environments, but is a useful solution in quantitatively evaluating the effectiveness of a DVFS solution. Further energy savings could be achieved by instrumenting tasks within each MPI call.

Springer et al. [61] seek to find a schedule of CPU allocation and CPU frequencies satisfying a maximum energy usage, while minimizing execution time. Because communication and computation phases differ in their corresponding power consumption and execution time, their model separate execution time and energy usage into these components:

$$Time = T_{comp} + T_{comm} \tag{6}$$

$$Energy = T_{comp}P_{comp} + T_{comm}P_{comm} \tag{7}$$

First, the power consumption due to the computation component of the application is determined at each frequency, executing on a single node. A single node is used here to avoid power consumption being impacted by communication overhead. Power is derived by dividing energy usage by execution time. Power usage due to computation will vary across applications due to intrinsic behavioral differences.

B.2 Regular Iterations and Non-Regular Iterations

HPC applications tend to be iterative in nature, meaning history can be a good indicator of predicting future behaviors. Several DVFS runtime systems have been developed to leverage this iterative nature to develop near-optimal energy savings. An increasingly growing number of HPC applications are falling into a non-iterative category, where repeated events over several iterations will differ. In the following subsections, we discuss the different runtime systems developed for both types of applications.

Regular Iterations. These runtime systems assume the target application is iterative in nature, and thus history is leveraged to predict future behaviors [34], [35], [36],

[69]. At each iteration, the algorithm computes the total slack per processor, and schedules an ideal frequency for the subsequent iteration. The target here are nodes on the critical path, exploiting their time waiting at synchronization points in the computation for energy savings such that all nodes arrive at the same time.

Jitter [34] requires manual instrumentation of the application with a special `MPI_Jitter` call to determine the iteration boundary. To reduce overhead, Jitter waits several iterations before taking any action if the iteration is too short. PMPI is used to instrument MPI primitives and determine slack time. It records the timestamp when the call starts, the timestamp when the call ends, and derives the wait time for that iteration as well as the global wait time of the node. To determine which nodes are ahead and which nodes are behind, the slack of each node is compared to the global minimum slack time, and an appropriate frequency is scheduled. Unfortunately, making CPU scaling decisions at each iteration (or a set of multiple iterations) leaves the potential for further energy savings. It assumes the critical node will be constant throughout the current iteration, when in reality, nodes are likely to move on and off the critical path in a given iteration.

Laurenzano et al. [63] perform an extensive set of benchmark tests for each target system, which are included in a database. The tests provide the expected power consumption behavior and execution time for different application execution patterns at various CPU frequencies. Each application is instrumented at the loop-level and is compared to loop results in the benchmark test suite on six characteristics, hit rates for L1, L2, and L3 cache, the ratio of the number of floating-point operations to the number of memory operations, and the average look-ahead values for floating-point and integer computation. The resulting power consumption and execution time behaviors for the benchmark loop are used to predict how CPU frequency will impact the application's loop.

Porterfield et al. [74] highlight a further complication when determining energy savings and designing runtime systems to save energy — something that has been significantly overlooked — temperature. Temperature will affect energy usage variation across different granularities (*e.g.,* per-iteration, per-node, per-socket, etc.) and will need to be accounted for in future approaches.

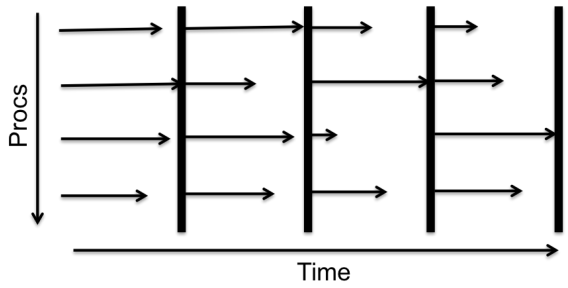

Fig. 7
BULK SYNCHRONOUS APPLICATION DISTRIBUTED ACROSS SEVERAL THREADS WORKING TOWARDS GLOBAL SYNCHRONIZATION POINTS (DENOTED BY VERTICAL BARS) IN THE COMPUTATION. DUE TO OS JITTER, MANUFACTURING VARIATIONS, OR LOAD IMBALANCE, THREADS MAY NOT REACH THE SYNCHRONIZATION POINT AT THE SAME TIME. AS A RESULT, THREADS THAT REACH THE BARRIER FIRST MUST STOP COMPUTING AND WAIT UNTIL ALL THREADS HAVE REACHED THE BARRIER. THE OVERALL PERFORMANCE HERE IS DETERMINED BY THE SLOWEST THREAD TO REACH THE SYNCHRONIZATION POINT. REDUCING THE FREQUENCY OF THREADS WAITING AT THE SYNCHRONIZATION POINT ARE IDEAL CANDIDATES FOR ENERGY SAVINGS.

Non-Regular Iterations. Several DVFS runtime systems have been developed assuming applications are iterative in nature. However, an increasing number of HPC applications are having non-temporal communication behaviors [75], [76], [77]. Said another way, the amount of time spent in the same MPI call across different iterations will vary based on application parameters for a given iteration. These differences become exacerbated due to intrinsic system properties, such OS jitter, manufacturing variations, and network contention, and could move an application from temporal to non-temporal and vice versa. Energy Aware MPI (EAM) was developed to save energy across a wider variety of application characteristics and behaviors [64]. It uses models of common MPI communication primitives to predict the minimum execution time, conducts an online assessment of slack time, and applies some frequency correlated to the amount of slack present.

B.3 Fixed-Time Intervals

These DVFS algorithms make energy-saving decisions at fixed-time intervals by predicting the upcom-

ing interval based on previous intervals (*i.e.,* feedback system). For traditional bulk-synchronous HPC applications, the overall performance of the application is determined by the last thread to reach the synchronization point (*i.e.,* on the critical path). This intra-node approach implements various heuristics to determine which threads should receive higher frequencies for better performance, and which threads would benefit from lower frequencies, such that all threads arrive at the same time (see Fig. 7). While this approach can result in significant energy savings without any instrumentation of the application source code, it has no method for intelligently tracking the critical path and can make an incorrect decision and degrade overall performance. By allocating frequencies appropriately to each thread, significant energy savings and better performance can be achieved.

CPUFreq is an infrastructure consisting of frequency-scaling governors [65], which are DVFS algorithms for energy and power savings supported by the Linux kernel. *cpuspeed*, *thermald*, and *cpupower* are some userspace utilities that make use of the CPUFreq governor to vary frequency. While most DVFS-enabled products on the market today are per-socket scope, these frequency governors are per-core or per-thread scope. Currently, there are five available governors, three statically schedule the frequency, and two dynamically schedule the frequency:

- *Performance*: Run at the maximum allowable frequency.
- *Powersave*: Contrary to performance, run at the lowest allowable frequency.
- *Userspace*: Run at the user-specified frequency.
- *Ondemand*: Dynamically scale the frequency to the current load.
- *Conservative*: Similar to ondemand, scale the frequency to the current load dynamically, but do so at a more gradual rate.

Traditionally, the default governor is *ondemand* for desktop systems, saving the user energy costs when not in use, while on HPC systems, the default is *performance*. The dynamic algorithms, *ondemand* and *conservative*, make frequency decisions at regular time intervals, assessing the current load of the system as a factor in its next decision.

Ge et al. [51] developed one of the earliest runtimes, comparing the potential for energy savings of a set of microbenchmarks exhibiting common behaviors of distributed HPC applications. Their results showed that computation-bound applications are not good candidates for energy savings since the CPU is the bottleneck, while memory-bound and communication-bound applications exhibit high slack times, which can be leveraged for energy savings with DVFS strategies.

In [66], Zhang et al. developed a criticality-aware DVFS governor, showing it consumes 15% and 25% less power than the *ondemand* and *performance* governors currently available in the Linux kernel. Identifying critical threads at runtime is a challenging problem. A simple approach is to track the progress of each thread towards the synchronization point, using the number of instructions completed as a metric for progress. This basic approach assumes each thread executes the same number of instructions between barriers, which [79] shows is not always a good indicator of criticality as it can have low variability across threads. As a solution to more intelligently predicting critical threads, two heuristics were created to identify lagging threads: Thread Criticality Predictors [79] and Criticality Stacks [80], further referred to as CS and TCP, respectively. TCP determines thread criticality based on cache hierarchy statistics, namely cache misses, while CS determines criticality based on the time each thread is performing useful work and the number of dependent threads. The TCP and CS predictors evaluate the current state of the system, and the new governor uses this information to make a better informed decision regarding which threads to scale or descale in frequency. By providing additional intelligence to the governor about criticality, Zhang et al.'s governor results in better energy usage than dynamic Linux governors [65].

Another online algorithm, known as CPU MISER — short for CPU Management Infrastructure for Energy Reduction — identifies application behavior at fixed intervals and schedules the optimal frequency for minimal performance degradation [35]. At each interval, CPU MISER collects a set of performance events to calculate a metric representing CPU intensity. Prediction models are used to estimate application behavior in the next interval. To mitigate degradation from mispredictions in the model, CPU MISER adapts the evaluation interval, such that the interval becomes less frequent at high frequencies and more frequent at lower frequencies.

The β-adaptation algorithm [50], [67] uses a user-specified performance threshold to select a schedule of frequencies such that the overall performance does not

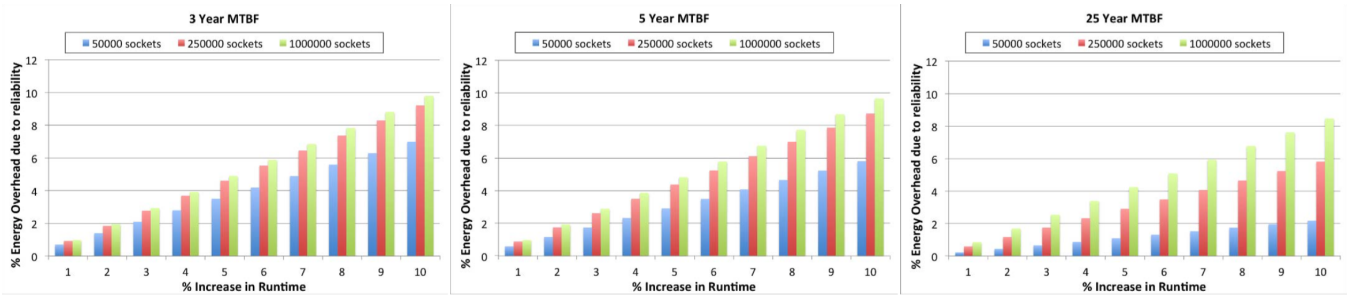

Fig. 8
ENERGY USAGE OVERHEAD OF RELIABILITY MECHANISMS FOR VARYING INCREASES TO OVERALL EXECUTION TIME [78].

exceed the threshold. At the beginning of each interval, the runtime system calculates the coefficient β described in Section IV-A, indicating the volume of off-chip memory access. Then, the ideal frequency for the subsequent interval is computed based on the β and performance threshold value, and the number of instructions per second is updated for computing the β value for the following interval.

B.4 Slack-Based

Slack time can be introduced into an application if processes are stalled at communication or collective calls waiting on dependencies. The DVFS approaches discussed in this section reduce energy usage with a minimal increase in overall execution time by reducing the frequency when slack exists at MPI communication and collective calls. These approaches are perhaps the simplest, as they target MPI communication calls and reduce the frequency of nodes with high slack time (*i.e.,* not on the critical path) [34], [36], [64], [69].

Rountree et al.'s solution using linear programming [60] resulted in close-to-optimal energy savings with negligible performance degradation. This approach switches between frequencies at MPI communication calls, identifying the critical path, so as not to delay the overall execution time, and approximating ideal frequencies over available frequencies. Unfortunately, linear programming is unfeasible to solve at runtime, making it a non-optimal approach for production.

Thrifty Barrier [68] puts non-critical threads to deep sleep modes while waiting at barriers, which is a technology more geared towards client processors.

Adagio [36] requires no modification to the source code and results in maximum energy savings with negligible (<1%) slowdown. Its algorithm leverages the findings of Jitter to determine the ideal frequency for a given task, which consists of computation, communication, and slack time as shown in Fig. 6. Here, a task is more fine-grained than Jitter's iteration granularity, which likely consists of multiple tasks. In this manner, Adagio can react better to the critical path as it moves across nodes.

For each instance of a repeated task, Adagio records the number of instructions and IPC for the current operating frequency on the specific node. This enables the algorithm to detect a change in execution between repeated tasks and to relate frequency to execution time. To maximize its energy saving ability, Adagio implemented several optimizations. Most notably, it addresses the limited number of discrete CPU frequencies available on the platform by running tasks in nearby frequencies to approximate the ideal frequency. Other DVFS algorithms select the optimal frequency closest to the discrete frequency that is available. This could mean forgoing energy savings by running at a faster frequency or suffering a performance penalty by running at a slower frequency. Adagio uses a timer to approximate the calculated ideal frequency by leveraging the nearby discrete frequencies. For example, if the ideal frequency is 1.5 GHz, but the processor only exposes 1.6 GHz and 1.4 GHz, then Adagio would run half the task at 1.6 GHz and the other half at 1.4 GHz, such that the ideal frequency is achieved. There is a penalty for each change in frequency, so Adagio has implemented a minimum time for the processor to stay in a constant frequency.

Green Queue [62] is the current largest demonstration of a DVFS runtime algorithm on a production system with 1024 nodes. It requires an initial profile of the application for computational (intra-node) and communication bottlenecks (inter-node). Unlike other offline approaches, Green Queue stores the trace (and resulting DVFS schedule) in a database. If the application is executed again with a similar input configuration, the DVFS schedule is pulled from the database and used to run the application. Similar to Jitter, Green Queue assumes iterations at the inner-most level are homogeneous, and phase transitions occur at each iteration of the outer loop, which Adagio showed is too coarse-grained for maximum energy savings.

Slack-time in OpenMP programs are the main contributors to high power consumption [70]. The authors used the OpenMP Runtime API (ORA) to collect various performance metrics at the OpenMP event-level during runtime.

C. Efficacy Evaluation Metrics

Several DVFS approaches have been developed to save energy in distributed HPC applications, but evaluating which approach is better is ambiguous, normally answered by the user's tolerance, which may not be acceptable to another. For example, one approach may save 5% energy savings, but cause a 2% increase in performance, and another approach may save 8% in energy, and cause a 4% increase in performance. Both approaches satisfy the goal of saving energy, but the acceptable slowdown is largely user dependent. As such, evaluation metrics have been used to make quantitative conclusions on how beneficial the approach is when comparing one to another. In other words, these metrics take a two-dimensional space, energy usage and performance, and translate it to a single metric.

Energy-delay product and energy-delay^2 product, EDP and ED^2P, respectively, are common metrics for low power circuits to quantify the trade-offs between power and performance:

$$EDP = E \times D^1, \tag{8}$$

and

$$ED^2P = E \times D^2, \tag{9}$$

where E is the energy, and D is the circuit delay. The exponent represents the percentage increase to energy usage (*i.e.,* qualitative) given a 1% slowdown in the circuit [81]. Several research endeavors leverage this metric to evaluate the energy-performance trade-offs of their techniques [69], [82], [83]. In [47], ED^2P and ED^3P are used to determine the optimal CPU frequency that results in the lowest ED^2P and ED^3P values. ED^3P is more restrictive on performance degradation than ED^2P, so a smaller performance loss results when using ED^3P.

In [51], a generalized version of EDP was proposed to determine the trade-offs between performance and energy savings, enabling the user to determine some threshold X for performance degradation to achieve Y savings in energy. The metric includes the percentage of energy that must be saved given some increase to overall execution time and a user-specified weight factor:

$$ED^2P_{generalized} = E^{(1-\delta)} \times D^{2(1+\delta)} \tag{10}$$

Here, δ is a weight value between $-1 \leq \delta \leq 1$. If $-1 \leq \delta < 0$, then the user favors energy savings over performance degradation, else if $0 < \delta \leq 1$, then the user favors minimal performance degradation over energy savings, and if $\delta = 0$, then the user favors them equally.

A survey of evaluation metrics provided by [78] sought to find an acceptable metric for comparing DVFS techniques. They found shortcomings in current state-of-the-art metrics for evaluating energy savings approaches. First, current metrics are unaware of the energy usage spent providing reliability (*e.g.,* cooling, checkpoint/restart, additional backup components, etc.). Second, current metrics do not account for the increased likelihood of failure during application execution due to longer runtimes associated with energy saving approaches. In response, they proposed a new metric to better evaluate energy savings techniques in the realm of large-scale HPC systems, which are not only associated with high power consumptions, but also increased failure rates. Energy savings techniques may decrease failure rates by providing lower temperatures, while minimizing the increase to overall execution time. On the other hand, DVFS techniques save energy by lowering voltages, which can increase failure rates. Referring back to the two examples mentioned at the beginning of this section, a 2% or 4% increase to overall performance for an application running for several days may translate into the application encountering another failure, and

subsequent restart, which may negate any energy savings. The proposed metric is *energy-reliability*, which better quantifies the effectiveness of an energy savings technique with respect to reliability mechanisms. Fig. 8 illustrates the growing dominance of reliability mechanisms in overall energy usage in large-scale systems. Energy-reliability encompasses the default application energy usage where no failures occur and the energy usage associated with reliability mechanisms, demonstrating that expending more energy to complete an application faster and reduce the likelihood of encountering a failure, can result in lower energy usage.

Another energy efficiency metric is the vector-valued metric developed by Hsu et al [84]. It differs from previous metrics described in this section in that it places equal weight on both energy and performance in a two-dimensional vector, whereas many other metrics condense these two objectives into one scalar value. With a single scalar value representing energy efficiency, supercomputers can be fully ordered, but with the vector-valued metric only partial order can be achieved. In other words, a supercomputer is better if both energy efficiency and performance are higher than its comparator. A potential downside of partial order is that multiple supercomputers will be similarly ranked in terms of best energy efficiency or highest performance. Additionally, there will be a cluster of supercomputers that will fall into the middle of these two axes, and determining where the “best" supercomputer should fall is highly subjective.

The search for an ideal energy efficiency and performance metric continues forward.

V. Power-Constrained HPC

Future HPC systems will be significantly power constrained. The U.S. DOE has imposed a 20 MW power budget in achieving an exaflop [1], [2], [3]. This will force power budgets to be imposed across all levels of the system as illustrated in Fig. 9. The facility-wide power bound will constrain the amount of power that can be given to each system in the machine room. Each system will have a system-wide scheduler to manage the power across all jobs on the system. Each job will have a resource manager to ensure the job power bound is not exceeded. The job-level power bound translates into a node-level and socket-level power bound. The power bounds at all levels in the hierarchy must be respected in order to ensure reliable operation.

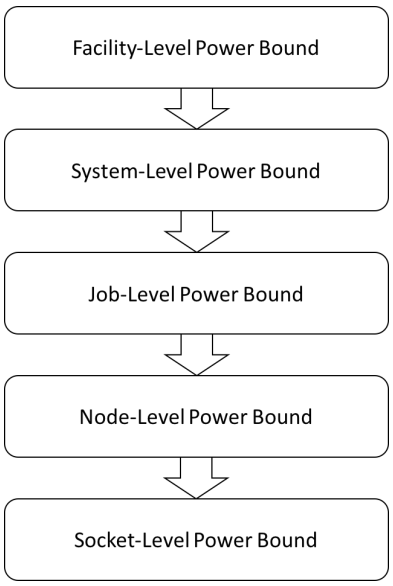

Fig. 9
Hierarchical power bounds.

Power capping has become a tractable solution to meeting power goals at exascale. Rather than managing the CPU frequency directly with DVFS, software mechanisms are enabling users to specify a specific power bound, and the firmware will modulate between discrete frequencies to guarantee the power bound is not exceeded. While performance under DVFS has been extensively studied for years and is well-understood, performance under a power cap will be difficult as differences in manufacturing variability become exposed. This section will provide an in-depth overview of the previous and current research endeavors of power-constrained techniques for HPC.

A. Hardware Overprovisioning

In order to achieve an exaflop at 20 MW, HPC systems are likely to contain more nodes than can be powered at peak simultaneously without exceeding the system-wide bound. This is known as hardware overprovisioning [39], or overprovisioning for short. Overprovisioning is being driven by the utilization wall [85] initially proposed by Venkatesh et al. Increasing transistor switching rates are producing more heat than can be efficiently dissipated, limiting the portion of the processor that can run at peak power. The remaining fraction of the processor left unpowered is known as *dark silicon* [86].

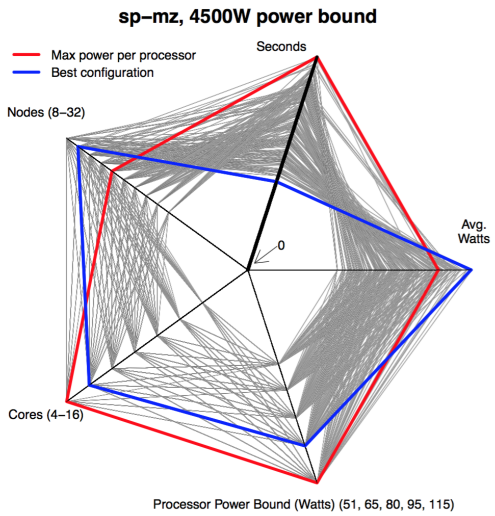

Fig. 10
EXHAUSTIVE CONFIGURATION SEARCH SPACE FOR AN APPLICATION AT A 4500W SYSTEM POWER BOUND [88].

Processor designs are already leveraging overprovisioning. The operating frequency of each core is dependent on the number of active cores, and all cores cannot simultaneously be operating at the highest frequency. By applying this same idea to HPC, customizing the hardware configuration on a per-application basis depending on its trace and component utilization can result in better performance [39], [87]. For example, if an application has high memory utilization, reducing the number of cores per node or allocating more power to the memory subsystem may result in better performance. Fig. 10 shows a configuration with fewer cores per node running at a lower power cap results in the best overall execution time, significantly outperforming the default case where each processor runs at TDP.

Most HPC applications stress the system components far below their peak capacity [38], so allocating maximum power is an inefficient use of the limited power at exascale. Fig. 11 demonstrated an inefficient use of system-wide power as almost 1 MW of power was left unused.

New technologies, such as Intel's RAPL, enable software to impose a hardware-enforced power bound on the processor. In the simplest case, RAPL could statically impose the same power bound on each processor, but this would also be poor use of the power as some applications may not run steadily at the power bound. In the alternative case, RAPL could dynamically impose a power limit based on the current utilization of the hardware components. Reaching an exaflop under the power bound will require resource managers to make well-informed decisions on how to best allocate power among the components in order to maximize performance.

B. *Performance Under a Power Bound*

Rountree et al. [37] paved the way for the future of power-aware supercomputing by leveraging RAPL to enforce a power bound on the processor. Their work was the first to use this technology exposing the new challenges that come with applying a power bound across a large number of processors. Fig. 12 shows the power draw across 64 homogeneous processors over six different parallel benchmarks. The higher the power consumption for a given processor, the less efficient it is, while the lower the power consumption, the more efficient the processor. Fig. 12 also indicates the severe level of processor heterogeneity that will exist in exascale systems — a challenge HPC will need to overcome. Work by Fukazawa et al. [89] also looked at the impacts of a power bound on the execution time of applications, finding a non-linear correlation between execution time and power consumption when under a power cap.

Traditionally, manufacturing variations are mitigated by grouping the processors by similar performance behaviors. Then, HPC systems are composed of processors from the same bin. This approach ignores variations in power consumption across duplicate processors (see Fig. 12). Applying a power bound further exposes variations in CPU frequencies, which impact overall performance. An initial quantification of the

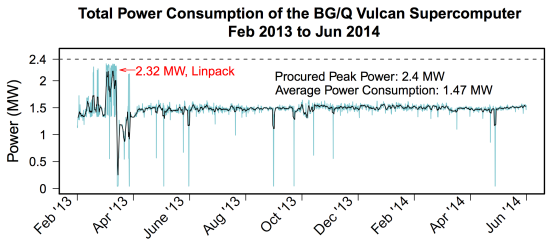

Fig. 11
SYSTEM-WIDE POWER CONSUMPTION ON PRODUCTION SYSTEM AT LLNL [38].

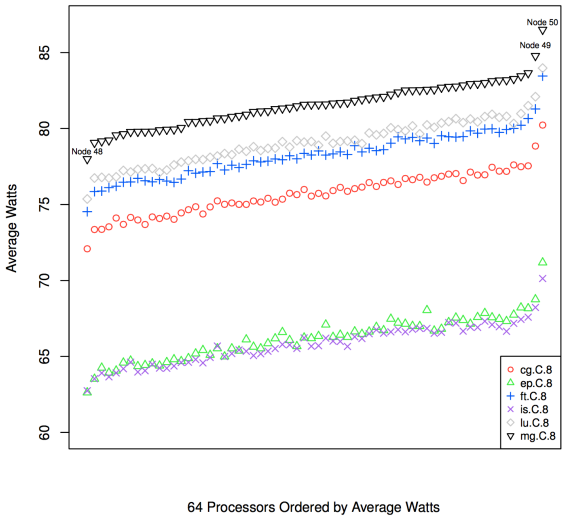

Fig. 12
POWER DRAW OF 64 PROCESSORS OVER SIX PARALLEL BENCHMARKS [37].

manufacturing variations in four current production systems showed as much as 23% variation in power consumption and as much as 17% in performance variation across distributed ranks [43]. Manufacturing variations are a large challenge, irrespective of the underlying architecture, and correlations between performance and power consumption are almost impossible to predict. Further, when the non-homogeneous processors are under a power bound, the performance variation across ranks worsens to 64%.

C. Power-Aware System-Level Schedulers

System-level schedulers are necessary to ensure the power consumption of all running jobs does not exceed the power bound imposed by the facility. At present, these schedulers are aware of the number of nodes available and the number of nodes allocated to other jobs. At exascale where power is a limited resource, schedulers will need to also be aware of the available power and allocate compute nodes appropriately. RMAP, short for Resource Manager for Power [38], [42], schedules jobs based on the available power in order to improve turnaround time for jobs and to minimize the amount of unused system power at any given time. Traditionally when submitting a job, the user specifies the number of nodes and the estimated runtime. In this power-aware scheduler, users will additionally specify a threshold value, identifying a maximum slowdown tolerance factor. In some cases, RMAP may apply a lower power cap on the processor, degrading the performance of a job, but enabling the job to fit within the current available resources.

Power capping technologies alone will not be sufficient at enforcing a system-wide power bound. In addition to power capping individual components, a global agent must ensure the global power bound is not exceeded. POWSched schedules power between jobs in a cluster [90], [91]. It is independent of the application behaviors, making reallocation decisions based on current per-socket power consumption, while abiding by the system-wide power bound. With this dynamic approach, system-wide throughput is increased up to 14% by reallocating excess power on processors where applications are not fully utilizing the available power.

D. Power-Aware Job-Level Resource Managers

Power-aware resource managers will be critical to making effective use of the limited power such that performance is maximized.

In a power-constrained setting, the overall performance is determined by the slowest node under the power cap. To maximize performance, Power Balancer is shown to be a viable solution in maintaining the same performance while using less power, since the faster nodes are allocated less power, freeing up power to speed up the slower nodes [92]. Power balancer requires executing the application once at the desired power budget per-node to determine the slowest processor. Then, for all other packages, power balancer determines the minimum power cap, where the performance is still better than the slowest processor. Because these metrics are collected offline and without a global view of the application behaviors running on each node, power balancer is limited in its effectiveness and is not ideal for production.

Similarly, the work by [43] also requires an initial run of the application, one at the minimum frequency and one at the maximum frequency, in order to collect CPU and DRAM power consumption.

Conductor [93] combines DVFS and power capping to manage power consumption of a single job. Power budgets are enforced from the top-down. The supercomputing center has a maximum power limit enforced either by the power capacity that can be drawn from the wall or budgetary constraints. This system-wide

power limit is divided up across the number of nodes in the system. Assuming the system is worst-case provisioned, the budget assumes every node is running at peak power. Then, the per-node power limit is divided among the components in the node, such as the processor and memory subsystem. The per-job power limit is derived from the resource allocations. Conductor initially configures the level of concurrency and CPU operating frequency to minimize slack time. RAPL is used to enforce a per-processor power limit. During application execution, Conductor shifts power to maximize performance under a job-level power bound, dynamically changing the level of concurrency and operating frequency on a per-task basis.

An integer linear programming solution shows the theoretical upper bound of performance benefits in a power-constrained environment [41]. Results demonstrate that current power-aware runtime systems are leaving further performance improvements on the table. By selecting an optimal configuration, up to 41.1% better performance could be gained. At this time, Conductor provides close-to-optimal performance improvement as determined by the ILP solution.

E. Per-Component Power Allocations

Much of the previously discussed work has focused on power capping of the processor. While the CPU is traditionally the heavy power consumer in an HPC system, other components, such as the memory, interconnect, accelerators, etc., will need to be accounted for as well in the global power bound.

Sarood et al. [40] used an interpolation scheme to determine the optimal configuration for running an application in an overprovisioned setting. Finding the best configuration would require an offline phase of profiling the application for each valid configuration $(n \times p_c, p_m)$, where n is the number of nodes, p_c is the power cap for the CPU, and p_m is the power cap for the memory. However, this can result in a huge overhead as the number of nodes or available number of discrete power caps for the CPU and memory increases. To mitigate the overhead, [40] used curve fitting to predict performance. In some instances, capping the CPU and memory frees up enough power for the application to use an additional node for computation.

Similarly, work by Tiwari et al. [94] also target capping the CPU and DRAM to maximize performance under a power bound. Their work differs from that of [40] as it does not require offline profiling of the application itself. Rather, it uses profiles of computation execution behaviors from representative HPC kernels. The resulting empirical data set is used to develop a prediction model on the optimal power cap configuration for the CPU and DRAM to optimize performance.

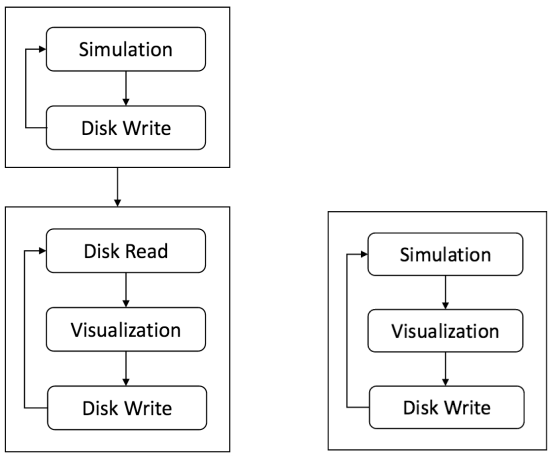

Fig. 13
TWO TYPES OF VISUALIZATION PIPELINES. THE LEFT FIGURE SHOWS THE TRADITIONAL POST-PROCESSING VISUALIZATION MODEL, WHILE THE RIGHT SHOWS THE IN SITU MODEL [95].

VI. SCIENTIFIC VISUALIZATION AND POWER

In addition to high power consumption at exascale, another critical problem will be I/O bandwidth rates. The rate at which scientific simulations are producing large amounts of data is far exceeding the ability to write out that data and store it to disk. This will require new methodologies to analyze simulation data with higher temporal sparsity.

The traditional scientific visualization workflow follows a post-processing model with two stages as shown in Fig. 13. First, the scientific simulation writes out data to disk at discrete time steps. Then, the data is read from disk and post-processed with scientific visualization tools, such as VisIt [96] and ParaView [97]. As simulations evolve in complexity, the amount of data per time step is increasing exponentially, making it unfeasible to write out data at frequent rates. Advanced solutions write out fewer time steps, or more recently, follow an in situ processing model, where data is processed alongside the simulation. The challenge here is

saving enough data from the simulation without losing accuracy or key areas of interest. As the I/O bottleneck increases, simulation data must be written to disk less frequently and the ability to gain insight from simulation codes decreases.

Scientific visualization routines will need to work not only within the I/O bandwidth limitations, but also within the framework of a power-constrained system at exascale. Currently, research at the crossroads between power-constrained HPC and scientific visualization is minimal [52], [95], [98], [99], [100], [101].

For in situ strategies, power will need to be shared in some fashion between the scientific simulation and visualization [102]. In one scenario, the simulation and visualization may share the compute resources without any data movement necessary. In another scenario, compute cores may be split among the simulation and visualization, necessitating intra-node data movement. In a third scenario, compute nodes may be split among the simulation and visualization, requiring inter-node data movement. Managing power allocations for in situ strategies will be a critical advancement in the field of power-aware scientific visualization as it will enable continued data exploration and analysis of large-scale scientific simulations. Within this context, memory capacity and bandwidth will be shared between the simulation and visualization, so understanding how to allocate power to the memory hierarchy will be another important area of research. This area is unique from previous power-constrained research endeavors that aim to benefit a wide range of individual applications.

Data movement costs have become a large factor in continuing to gain insight from scientific data. The scientific visualization community has moved away from a post-processing model to an in situ model, where data is processed alongside the simulation, potentially reducing the overheads of data storage and data movement. Additionally, research endeavors have targeted NVRAM as an intermediate data staging area for in situ processing. This technology lessens the latency and performance gap between main memory and disk, but the energy costs have not been studied. Gamell et al. [103] conducted a study of the power consumption behaviors of different data management and data movement strategies of a configuration with NVRAM in the memory hierarchy. They found NVRAM to be a viable solution as it has high bandwidth for data transfer, particularly for large block sizes. However, NVRAM has a higher power consumption than the SSD.

Labasan et al. [52] provided an initial survey of the various factors that impact power and performance trade-offs for an isosurfacing algorithm implemented in two frameworks. First, the algorithm was self-implemented to specifically perform Marching Cubes on tetrahedral data. The optimized implementation was efficient for this particular data type, especially in the number of instructions executed. Second, the isosurfacing algorithm was implemented with a general-purpose framework known as VTK [104]. General-purpose frameworks such as VTK sacrifice performance for the ability to handle a wide variety of configurations and data types. Aside from implementation, other factors impacting the trade-off include concurrency, parallel programming paradigm, and architecture.

VII. Conclusion and Future Work

Scientific visualization is a key component in making scientific discoveries. It enables the exploration and analysis of scientific data, and the ability to communicate findings through a comprehensible image. Visualization at exascale will encounter challenges not only from I/O bandwidth rates, but will also from power constraints. The visualization community is actively moving towards in situ processing in order to reduce the I/O costs associated with moving large amounts of data in and out of memory. At present, there is a very limited set of work in addressing the power constraints at exascale and the challenges associated with an in situ pipeline. Several open questions must be addressed, such as:

- Are there distinct phases with visualization applications that are good candidates for power-saving techniques?
- What are the power behaviors of different data movement strategies?
- How should power be scheduled between the simulation and visualization? Are there times power can be shifted away from the visualization, resulting in degraded image quality or analysis accuracy, thus freeing up power for the simulation?

The community community will need to explore and address this space in order to reach exascale. Our research will investigate the relatively new space of power-aware visualization and understand the trade-offs between power and performance for visualization algorithms.

REFERENCES

[1] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, Tony Mezzacappa, Parviz Moin, Mike Norman, Robert Rosner, Vivek Sarkar, Andrew Siegel, Fred Streitz, Andy White, and Margaret Wright, "The Opportunities and Challenges of Exascale Computing," Tech. Rep., Advanced Scientific Computing, Fall 2010.

[2] Pete Beckman, Ron Brightwell, Bronis R. de Supinski, Maya Gokhale, Steven Hofmeyr, Sriram Krishnamoorthy, Mike Lang, Barney Maccabe, John Shalf, and Marc Snir, "2013 Exascale Operating and Runtime Systems," Tech. Rep., Advanced Scientific Computing Research (ASCR), Nov 2012.

[3] Vivek Sarkar, William Harrod, and Allan E Snavely, "Software Challenges in Extreme Scale Systems," in *Journal of Physics: Conference Series*, 2009, p. 012045.

[4] Erich Strohmaier, "TOP500 Supercomputer," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006, SC '06, ACM.

[5] J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, 1979.

[6] Steve Ashby, Pete Beckman, Jackie Chen, Phil Colella, Bill Collins, Dona Crawford, Jack Dongarra, Doug Kothe, Rusty Lusk, Paul Messina, and Others, "The Opportunities and Challenges of Exascale Computing," *Summary report of the advanced scientific computing advisory committee (ASCAC) subcommittee at the US Department of Energy Office of Science*, 2010.

[7] Gordon E. Moore, "Readings in Computer Architecture," chapter Cramming More Components Onto Integrated Circuits, pp. 56–59. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[8] Tom Scogland, Balaji Subramaniam, and Wu-Chun Feng, "The Green500 List: Escapades to Exascale," *Comput. Sci.*, vol. 28, no. 2-3, pp. 109–117, May 2013.

[9] Shuaiwen Leon Song, Kevin Barker, and Darren Kerbyson, "Unified Performance and Power Modeling of Scientific Workloads," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, New York, NY, USA, 2013, E2SC '13, pp. 4:1–4:8, ACM.

[10] "WattsUp? Pro," `http://www.wattsupmeters.com`, Accessed: 2016-09-08.

[11] Alastair Hart, Harvey Richardson, Jens Doleschal, Thomas Ilsche, Mario Bielert, and Matthew Kappel, "User-level Power Monitoring and Application Performance on Cray XC30 Supercomputers," *Proceedings of the Cray User Group (CUG)*, 2014.

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

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

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

[15] NVIDIA, *NVML Reference Manual*, May 2015.

[16] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron, "PowerPack: Energy Profiling and Analysis of High-Performance Systems and Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2010.

[17] James H Laros, Phil Pokorny, and David DeBonis, "PowerInsight-A Commodity Power Measurement Capability," in *Green Computing Conference (IGCC), 2013 International*. IEEE, 2013, pp. 1–6.

[18] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-grained and Integrated Power Monitoring for Commodity Computer Systems," in *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, March 2010, pp. 479–484.

[19] T. Ilsche, D. Hackenberg, S. Graul, R. SchÃűne, and J. Schuchart, "Power Measurements for Compute Nodes: Improving Sampling Rates, Granularity and Accuracy," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, Dec 2015, pp. 1–8.

[20] D. Hackenberg, T. Ilsche, J. Schuchart, R. SchÃűne, W. E. Nagel, M. Simon, and Y. Georgiou, "HDEEM: High Definition Energy Efficiency Monitoring," in *Energy Efficient Supercomputing Workshop (E2SC), 2014*, Nov 2014, pp. 1–10.

[21] D. Hackenberg, R. SchÃűne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An Energy Efficiency Feature Survey of the Intel Haswell Processor," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, May 2015, pp. 896–904.

[22] Krisztián Flautner, Steve Reinhardt, and Trevor Mudge, "Automatic Performance Setting for Dynamic Voltage Scaling," in *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, 2001, MobiCom '01, pp. 260–271, ACM.

[23] Jason Flinn and M. Satyanarayanan, "Energy-aware Adaptation for Mobile Applications," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 1999, SOSP '99, pp. 48–63, ACM.

[24] Kinshuk Govil, Edwin Chan, and Hal Wasserman, "Comparing Algorithm for Dynamic Speed-setting of a Low-power CPU," in *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking*, New York, NY, USA, 1995, MobiCom '95, pp. 13–25, ACM.

[25] Dirk Grunwald, Charles B. Morrey, III, Philip Levis, Michael Neufeld, and Keith I. Farkas, "Policies for Dynamic Clock Scheduling," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, Berkeley, CA, USA, 2000, OSDI'00, USENIX Association.

[26] Flavius Gruian, "Hard Real-time Scheduling for Low-energy Using Stochastic Data and DVS Processors," in *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, New York, NY, USA, 2001, ISLPED '01, pp. 46–51, ACM.

[27] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker, "Scheduling for Reduced CPU Energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, Berkeley, CA, USA, 1994, OSDI '94, USENIX Association.

[28] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat, "ECOSystem: Managing Energy As a First Class Operating System Resource," *SIGPLAN Not.*, vol. 37, no. 10, pp. 123–132, Oct. 2002.

[29] Nathaniel Pettis, Le Cai, and Yung-Hsiang Lu, "Dynamic Power Management for Streaming Data," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, New York, NY, USA, 2004, ISLPED '04, pp. 62–65, ACM.

[30] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle, "Managing Energy and Server Resources in Hosting Centers," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 103–116, Oct. 2001.

[31] E. N. Elnozahy, Michael Kistler, and Ramakrishnan Rajamony, "Energy-efficient Server Clusters," in *Proceedings of the 2Nd International Conference on Power-aware Computer Systems*, Berlin, Heidelberg, 2003, PACS'02, pp. 179–197, Springer-Verlag.

[32] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony, "The Case for Power Management in Web Servers," in *Power Aware Computing*, Robert Graybill and Rami Melhem, Eds., chapter 1, pp. 261–289. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[33] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller, "Energy Management for Commercial Servers," *Computer*, vol. 36, no. 12, pp. 39–48, Dec. 2003.

[34] Vincent W. Freeh, Nandini Kappiah, David K. Lowenthal, and Tyler K. Bletsch, "Just-in-Time Dynamic Voltage Scaling: Exploiting Inter-node Slack to Save Energy in MPI Programs," *J. Parallel Distrib. Comput.*, vol. 68, no. 9, pp. 1175–1185, Sept. 2008.

[35] Rong Ge, Xizhou Feng, Wu chun Feng, and K.W. Cameron, "CPU MISER: A Performance-Directed, Run-Time System for Power-Aware Clusters," in *Parallel Processing, 2007. ICPP 2007. International Conference on*, Sept 2007, pp. 18–18.

[36] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch, "Adagio: Making DVS Practical for Complex HPC Applications," in *Proceedings of the 23rd International Conference on Supercomputing*, New York, NY, USA, 2009, ICS '09, pp. 460–469, ACM.

[37] B. Rountree, D.H. Ahn, B.R. de Supinski, D.K. Lowenthal, and M. Schulz, "Beyond DVFS: A First Look at Performance under a Hardware-Enforced Power Bound," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 947–953.

[38] Tapasya Patki, David K. Lowenthal, Anjana Sasidharan, Matthias Maiterth, Barry L. Rountree, Martin Schulz, and Bronis R. de Supinski, "Practical Resource Management in Power-Constrained, High Performance Computing," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2015, HPDC '15, pp. 121–132, ACM.

[39] Tapasya Patki, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. de Supinski, "Exploring Hardware Overprovisioning in Power-constrained, High Performance Computing," in *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, New York, NY, USA, 2013, ICS '13, pp. 173–182, ACM.

[40] O. Sarood, A. Langer, L. Kale, B. Rountree, and B. de Supinski, "Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems," in *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, Sept 2013, pp. 1–8.

[41] Peter E. Bailey, Aniruddha Marathe, David K. Lowenthal, Barry Rountree, and Martin Schulz, "Finding the Limits of Power-Constrained Application Performance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, SC '15, pp. 79:1–79:12, ACM.

[42] O. Sarood, A. Langer, A. Gupta, and L. Kale, "Maximizing Throughput of Overprovisioned HPC Data Centers Under a Strict Power Budget," in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, Nov 2014, pp. 807–818.

[43] Yuichi Inadomi, Tapasya Patki, Koji Inoue, Mutsumi Aoyagi, Barry Rountree, Martin Schulz, David Lowenthal, Yasutaka Wada, Keiichiro Fukazawa, Masatsugu Ueda, Masaaki Kondo, and Ikuo Miyoshi, "Analyzing and Mitigating the Impact of Manufacturing Variability in Power-Constrained Supercomputing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, SC '15, pp. 78:1–78:12, ACM.

[44] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, Aug 2010, pp. 189–194.

[45] H. Kimura, M. Sato, Y. Hotta, T. Boku, and D. Takahashi, "Empirical Study on Reducing Energy of Parallel Programs using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster," in *2006 IEEE International Conference on Cluster Computing*, Sept 2006, pp. 1–10.

[46] Ami Marowka, "Modeling the Effects of DFS on Power Consumption in Hybrid Chip Multiprocessors," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, New York, NY, USA, 2013, E2SC '13, pp. 3:1–3:8, ACM.

[47] Rong Ge, Xizhou Feng, and K.W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 34–34.

[48] Kirk W. Cameron, Rong Ge, and Xizhou Feng, "High-Performance, Power-Aware Distributed Computing for Scientific Applications," *Computer*, vol. 38, no. 11, pp. 40–47, Nov. 2005.

[49] Chung-Hsing Hsu and Ulrich Kremer, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, New York, NY, USA, 2003, PLDI '03, pp. 38–48, ACM.

[50] Chung-Hsing Hsu and Wu-Chun Feng, "Effective Dynamic Voltage Scaling Through CPU-Boundedness Detection," in *Proceedings of the 4th International Conference on Power-Aware Computer Systems*, Berlin, Heidelberg, 2005, PACS'04, pp. 135–149, Springer-Verlag.

[51] Rong Ge, Xizhou Feng, and K.W. Cameron, "Improvement of Power-Performance Efficiency for High-End Computing," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 8 pp.–.

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

[53] Matthew Curtis-Maury, Filip Blagojevic, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos, "Prediction-Based Power-Performance Adaptation of Multithreaded Scientific Codes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 10, pp. 1396–1410, 2008.

[54] Matthew Curtis-Maury, James Dzierwa, Christos D. Antonopoulos, and Dimitrios S. Nikolopoulos, "Online Power-performance Adaptation of Multithreaded Programs Using Hardware Event-based Prediction," in *Proceedings of the 20th Annual International Conference on Supercomputing*, New York, NY, USA, 2006, ICS '06, pp. 157–166, ACM.

[55] A. K. Porterfield, S. L. Olivier, S. Bhalachandra, and J. F. Prins, "Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 884–891.

[56] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz, "Prediction Models for Multi-Dimensional Power-Performance Optimization on Many Cores," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2008, PACT '08, pp. 250–259, ACM.

[57] D. Li, B. R. de Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for Energy-Efficient Resource Management of Hybrid Programming Models," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 144–157, Jan 2013.

[58] V. W. Freeh, Feng Pan, N. Kappiah, D. K. Lowenthal, and R. Springer, "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, April 2005, pp. 4a–4a.

[59] Vincent W. Freeh and David K. Lowenthal, "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster," in *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2005, PPoPP '05, pp. 164–173, ACM.

[60] B. Rountree, D.K. Lowenthal, S. Funk, Vincent W. Freeh, B.R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, Nov 2007, pp. 1–9.

[61] Robert Springer, David K. Lowenthal, Barry Rountree, and Vincent W. Freeh, "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster," in *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2006, PPoPP '06, pp. 230–238, ACM.

[62] Ananta Tiwari, Michael Laurenzano, Joshua Peraza, Laura Carrington, and Allan Snavely, "Green Queue: Customized Large-Scale Clock Frequency Scaling," in *Proceedings of the 2012 Second International Conference on Cloud and Green Computing*, Washington, DC, USA, 2012, CGC '12, pp. 260–267, IEEE Computer Society.

[63] Michael A. Laurenzano, Mitesh Meswani, Laura Carrington, Allan Snavely, Mustafa M. Tikir, and Stephen Poole, "Reducing Energy Usage with Memory and Computation-Aware Dynamic Frequency Scaling," in *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, Berlin, Heidelberg, 2011, Euro-Par'11, pp. 79–90, Springer-Verlag.

[64] Akshay Venkatesh, Abhinav Vishnu, Khaled Hamidouche, Nathan Tallent, Dhabaleswar (DK) Panda, Darren Kerbyson, and Adolfy Hoisie, "A Case for Application-oblivious Energy-Efficient MPI Runtime," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, SC '15, pp. 29:1–29:12, ACM.

[65] "Linux CPUFreq Governors," `https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt`, Accessed: 2016-09-14.

[66] H. Zhang, W. Han, F. Li, S. He, Y. Cheng, H. An, and Z. Chen, "A Criticality-Aware DVFS Runtime Utility for Optimizing Power Efficiency of Multithreaded Applications," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, pp. 841–848.

[67] C.-H. Hsu and Wu chun Feng, "A Power-Aware Run-Time System for High-Performance Computing," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 1–1.

[68] Jian Li, Jose F. Martinez, and Michael C. Huang, "The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, Washington, DC, USA, 2004, HPCA '04, pp. 14–, IEEE Computer Society.

[69] Min Yeol Lim, Vincent W. Freeh, and David K. Lowenthal, "Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 2006, SC '06, ACM.

[70] A. Nandamuri, A.M. Malik, A. Qawasmeh, and B.M. Chapman, "Power and Energy Footprint of OpenMP Programs Using OpenMP Runtime API," in *Energy Efficient Supercomputing Workshop (E2SC), 2014*, Nov 2014, pp. 79–88.

[71] Vincent W. Freeh, D.K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B.L. Rountree, and M.E. Femal, "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 18, no. 6, pp. 835–848, June 2007.

[72] B. Rountree, D.K. Lowenthal, M. Schulz, and B.R. de Supinski, "Practical Performance Prediction under Dynamic Voltage Frequency Scaling," in *Green Computing Conference and Workshops (IGCC), 2011 International*, July 2011, pp. 1–8.

[73] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks&Mdash;Summary and Preliminary Results," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, New York, NY, USA, 1991, Supercomputing '91, pp. 158–165, ACM.

[74] Allan Porterfield, Rob Fowler, Sridutt Bhalachandra, and Wei Wang, "OpenMP and MPI Application Energy Measurement Variation," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, New York, NY, USA, 2013, E2SC '13, pp. 7:1–7:8, ACM.

[75] D. J. Kerbyson, A. Vishnu, and K. J. Barker, "Energy Templates: Exploiting Application Information to Save Energy," in *2011 IEEE International Conference on Cluster Computing*, Sept 2011, pp. 225–233.

[76] Ian Karlin, Abhinav Bhatele, Jeff Keasler, Bradford L. Chamberlain, Jonathan Cohen, Zachary Devito, Riyaz Haque, Dan Laney, Edward Luke, Felix Wang, David Richards, Martin Schulz, and Charles H. Still, "Exploring Traditional and Emerging Parallel Programming Models Using a Proxy Application," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, Washington, DC, USA, 2013, IPDPS '13, pp. 919–932, IEEE Computer Society.

[77] M. Adams, P. Colella, D. T. Graves, J. N. Johnson, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. McCorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. Van Straalen, "Chombo Software Package for AMR Applications - Design Document," Tech. Rep. LBNL-6616E, EECS Department, University of California, Berkeley, December 2015.

[78] R. E. Grant, S. L. Olivier, J. H. Laros, R. Brightwell, and A. K. Porterfield, "Metrics for Evaluating Energy Saving Techniques for Resilient HPC Systems," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, pp. 790–797.

[79] Abhishek Bhattacharjee and Margaret Martonosi, "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 290–301, June 2009.

[80] Kristof Du Bois, Stijn Eyerman, Jennifer B. Sartor, and Lieven Eeckhout, "Criticality Stacks: Identifying Critical Threads in Parallel Programs Using Synchronization Behavior," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2013, ISCA '13, pp. 511–522, ACM.

[81] Chung-Hsing Hsu, Wu-chun Feng, and Jeremy S. Archuleta, "Towards Efficient Supercomputing: A Quest for the Right Metric," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 11 - Volume 12*, Washington, DC, USA, 2005, IPDPS '05, pp. 230.1–, IEEE Computer Society.

[82] Pradip Bose, Margaret Martonosi, and David Brooks, "Modeling and Analyzing CPU Power and Performance: Metrics, Methods, and Abstractions," *Tutorial, ACM SIGMETRICS*, vol. 9, 2001.

[83] Xizhou Feng, Rong Ge, and Kirk W. Cameron, "Power and Energy Profiling of Scientific Applications on Distributed Systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*, Washington, DC, USA, 2005, IPDPS '05, pp. 34–34, IEEE Computer Society.

[84] Chung-Hsing Hsu, Jeffery A. Kuehn, and Stephen W. Poole, "Towards Efficient Supercomputing: Searching for the Right Efficiency Metric," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA, 2012, ICPE '12, pp. 157–162, ACM.

[85] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor, "Conservation Cores: Reducing the Energy of Mature Computations," in *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2010, ASPLOS XV, pp. 205–218, ACM.

[86] "UCSD Center for Dark Silicon," `http://darksilicon.org/`, Accessed: 2016-09-19.

[87] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. d. Supinski, "Adaptive Configuration Selection for Power-Constrained Heterogeneous Systems," in *Parallel Processing (ICPP), 2014 43rd International Conference on*, Sept 2014, pp. 371–380.

[88] Tapasya Patki, *The Case for Hardware Overprovisioned Supercomputers*, Dissertation, University of Arizona, 2015.

[89] K. Fukazawa, M. Ueda, M. Aoyagi, T. Tsuhata, K. Yoshida, A. Uehara, M. Kuze, Y. Inadomi, and K. Inoue, "Power Consumption Evaluation of an MHD Simulation with CPU Power Capping," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, May 2014, pp. 612–617.

[90] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, and Martin Schulz, "Dynamic Power Sharing for Higher Job Throughput," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2015, SC '15, pp. 80:1–80:11, ACM.

[91] Daniel A. Ellsworth, Allen D. Malony, Barry Rountree, and Martin Schulz, "POW: System-Wide Dynamic Reallocation of Limited Power in HPC," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, New York, NY, USA, 2015, HPDC '15, pp. 145–148, ACM.

[92] Matthias Maiterth, Martin Schulz, Dieter Kranzlmuller, and Barry Rountree, "Power Balancing in an Emulated Exascale Environment," in *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops*, New York, NY, USA, 2016, IPDPSW '16, pp. 1142–1149, ACM.

[93] Aniruddha Marathe, Peter E. Bailey, David K. Lowenthal, Barry Rountree, Martin Schulz, and Bronis R. Supinski, *High Performance Computing: 30th International Confer-*

ence, ISC High Performance 2015, Frankfurt, Germany, July 12-16, 2015, Proceedings, chapter A Run-Time System for Power-Constrained HPC Applications, pp. 394–408, Springer International Publishing, Cham, 2015.

[94] A. Tiwari, M. Schulz, and L. Carrington, "Predicting Optimal Power Allocation for CPU and DRAM Domains," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, May 2015, pp. 951–959.

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

[96] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil, "VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data," in *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, pp. 357–372. Oct 2012.

[97] James Ahrens, Berk Geveci, and Charles Law, "ParaView: An End-User Tool for Large Data Visualization.," 01 2005.

[98] Marc Gamell, Ivan Rodero, Manish Parashar, Janine C. Bennett, Hemanth Kolla, Jacqueline Chen, Peer-Timo Bremer, Aaditya G. Landge, Attila Gyulassy, Patrick McCormick, Scott Pakin, Valerio Pascucci, and Scott Klasky, "Exploring Power Behaviors and Trade-offs of In-situ Data Analytics," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2013, SC '13, pp. 77:1–77:12, ACM.

[99] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil, and M. Mohiyuddin, "Energy-Efficient Computing for Extreme-Scale Science," *Computer*, vol. 42, no. 11, pp. 62–71, Nov 2009.

[100] J. Krueger, D. Donofrio, J. Shalf, M. Mohiyuddin, S. Williams, L. Oliker, and F. J. Pfreundt, "Hardware/Software Co-Design for Energy-Efficient Seismic Modeling," in *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov 2011, pp. 1–12.

[101] D. Kopta, K. Shkurko, J. Spjut, E. Brunvand, and A. Davis, "Memory Considerations for Low Energy Ray Tracing," *Comput. Graph. Forum*, vol. 34, no. 1, pp. 47–59, Feb. 2015.

[102] I. Rodero, M. Parashar, A. G. Landge, S. Kumar, V. Pascucci, and P. T. Bremer, "Evaluation of In-Situ Analysis Strategies at Scale for Power Efficiency and Scalability," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2016, pp. 156–164.

[103] M. Gamell, I. Rodero, M. Parashar, and S. Poole, "Exploring Energy and Performance Behaviors of Data-Intensive Scientific Workflows on Systems with Deep Memory Hierarchies," in *High Performance Computing (HiPC), 2013 20th International Conference on*, Dec 2013, pp. 226–235.

[104] William J. Schroeder, Kenneth M. Martin, and William E. Lorensen, "The Design and Implementation of an Object-oriented Toolkit for 3D Graphics and Visualization," in *Proceedings of the 7th Conference on Visualization '96*, Los Alamitos, CA, USA, 1996, VIS '96, pp. 93–ff., IEEE Computer Society Press.

[105] S. Huang, S. Xiao, and W. Feng, "On the Energy Efficiency of Graphics Processing Units for Scientific Computing," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, Washington, DC, USA, 2009, IPDPS '09, pp. 1–8, IEEE Computer Society.

[106] Vaibhav Sundriyal and Masha Sosonkina, "Initial Investigation of a Scheme to Use Instantaneous CPU Power Consumption for Energy Savings Format," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, New York, NY, USA, 2013, E2SC '13, pp. 1:1–1:6, ACM.

[107] C. H. Hsu, J. Combs, J. Nazor, F. Santiago, R. Thysell, S. Rivoire, and S. W. Poole, "Application Power Signature Analysis," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, May 2014, pp. 782–789.

[108] B. Austin and N. J. Wright, "Measurement and Interpretation of Micro-benchmark and Application Energy Use on the Cray XC30," in *Energy Efficient Supercomputing Workshop (E2SC), 2014*, Nov 2014, pp. 51–59.

[109] "msr-safe," `http://software.llnl.gov/msr-safe/`, Accessed: 2016-09-22.

[110] "libmsr," `http://software.llnl.gov/libmsr/`, Accessed: 2016-09-21.

[111] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver, "A Validation of DRAM RAPL Power Measurements," in *Proceedings of the 2016 International Symposium on Memory Systems (MEMSYS)*, New York, NY, USA, Oct 2016.

[112] Jack Dongarra, Hatem Ltaief, Piotr Luszczek, and Vincent M. Weaver, "Energy Footprint of Advanced Dense Numerical Linear Algebra Using Tile Algorithms on Multicore Architectures," in *Proceedings of the 2012 Second International Conference on Cloud and Green Computing*, Washington, DC, USA, 2012, CGC '12, pp. 274–281, IEEE Computer Society.

[113] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge," *IEEE Micro*, vol. 32, no. 2, pp. 20–27, March 2012.

[114] D. Hackenberg, T. Ilsche, R. SchÃűne, D. Molka, M. Schmidt, and W. E. Nagel, "Power Measurement Techniques on Standard Compute Nodes: A Quantitative Comparison," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, April 2013, pp. 194–204.

[115] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig, "Measuring Energy Consumption for Short Code Paths Using RAPL," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 3, pp. 13–17, Jan. 2012.

[116] James Demmel and Andrew Gearhart, "Instrumenting Linear Algebra Energy Consumption via On-chip Energy Counters," Tech. Rep. UCB/EECS-2012-168, EECS Department, University of California, Berkeley, Jun 2012.

[117] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, and G. D. Peterson, "Power Aware Computing on GPUs," in *Application Accelerators in High Performance*

Computing (SAAHPC), 2012 Symposium on, July 2012, pp. 64–73.

[118] Luigi Brochard, Raj Panda, and Sid Vemuganti, "Optimizing Performance and Energy of HPC Applications on POWER7," *Computer Science - Research and Development*, vol. 25, no. 3, pp. 135–140, 2010.

[119] Michael Knobloch, Maciej Foszczynski, Willi Homberg, Dirk Pleiter, and Hans Böttiger, "Mapping Fine-grained Power Measurements to HPC Application Runtime Characteristics on IBM POWER7," *Computer Science - Research and Development*, vol. 29, no. 3, pp. 211–219, 2014.

APPENDIX

I. POWER MONITORING TOOLS

There are a wide range of hardware and software tools for power monitoring. More commonly, these tools are capable of monitoring power rather than controlling power. As such, there is an increasingly high demand for tools that enable power control at fine-grained temporal and spatial granularities in moving towards exascale, where power will be a severely constrained resource and will need to be allocated wisely. In the following appendix, we provide more details on the power monitoring tools dicussed in Section II-C. For reference, the different power monitoring solutions discussed here are summarized in Table III.

A. Coarse-Grained Monitoring

The **WattsUp? Pro** [10] is an external monitor that polls the power consumption of the target device at frequent intervals. The power meter is connected between the power supply at the wall and the target device, so the meter is effectively measuring the total power draw from the wall power supply. The actual power measurement reported by the meter will be below the actual power at the wall due to leakage between the wall and the WattsUp? device.

There have been several research efforts using this external monitor [35], [63], [103], [105], [106], [107]. While such a device beneficially provides physical power measurements rather than modeled ones, it has some pitfalls. The first being that it reports the power consumption of the entire system, which does not reveal how the power is being consumed across different components, such as the processor and memory infrastructure. Measuring power consumption of a large-scale HPC system would render this solution cost-ineffective as multiple WattsUp? power meters would need to be purchased. Further, each power meter would only provide power consumption of the node to which it is connected, rather than giving the power consumption of the entire system, which is not as linear as simply computing the aggregate sum of the power consumption per node. An additional problem is the frequency update rate of twice per second, which will be too coarse-grained to attribute power consumption to particular regions or phases in the application.

Software can monitor power consumption and energy usage on **Cray XC30** systems at the node-level. The interface to these hardware counters are via files in userspace. Hart et al. [11] demonstrated the functionality of Cray's infrastructure on two different architectures, one with an accelerator and one without. Cray's new power measurement capabilities available beginning with the Cray Cascade provide additional counters that monitor the power consumption due to the accelerator. All counters are updated every 0.1 sec. Austin et al. [108] used Cray's power management infrastructure on the Edison system at NERSC to investigate the trade-offs between power and performance for large-scale HPC applications at the node-level.

B. Vendor-Specific Component-Level Monitoring

Intel's Running Average Power Limit (RAPL) technology [12] was first introduced with Sandy Bridge processors. RAPL provides privileged access to a wide variety of 64-bit model-specific registers (MSRs) to monitor and control various power and thermal features of the processor and peripherals. One of these features is predicting the power consumption with internal models based on counters in order to make decisions on how to spend any available power headroom wisely. With the introduction of Haswell processors, fully integrated voltage regulators enabled actual power measurements to vastly improve the accuracy of RAPL energy measurements [21].

Fig. 14 highlights four RAPL power domains, which can vary across architectures:

- Package (PKG): All components on the processor
- Power Plane 0 (PP0): Cores and local private caches
- Power Plane 1 (PP1): Uncore devices, such as graphics cards on client processors or QuickPath Interconnect (QPI) on server processors
- DRAM: Memory subsystem

Traditionally, the client processors support the *PKG*, *PP0*, and *PP1* domains, while the server processor supports *PKG*, *PP0*, and *DRAM* domains. Here, the assumption is that the power consumption, P, of the *PKG* domain is less than or equal to the sum of the *PP0* domain and the *PP1* domain:

$$P_{PKG} \leq P_{PP0} + P_{PP1} \tag{11}$$

Then, the assumption is that the power consumption attributed to the last-level cache (LLC) and integrated memory controller (iMC) is calculated by subtracting the power consumption of the *PKG* domain, the power

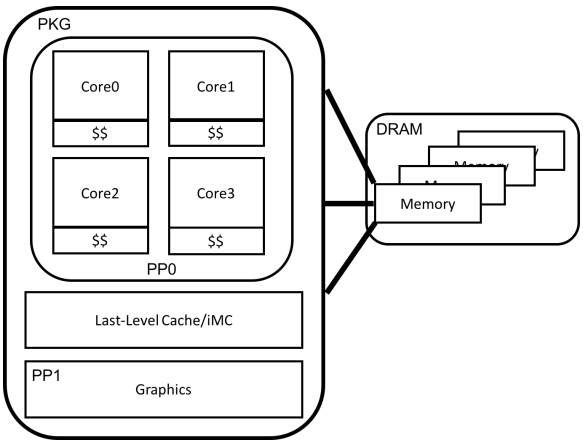

Fig. 14
RAPL DOMAINS FOR A GIVEN PROCESSOR.

consumption of the *PP0* domain, and the power consumption of the *PP1* domain:

$$P_{LLC,iMC} = P_{PKG} - P_{PP0} - P_{PP1} \qquad (12)$$

Each RAPL domain has registers for the following parameters:

- Power Limit and associated Time Window for enforcing a power cap
- Energy Status for monitoring energy usage (and deriving power consumption)
- Perf Status for monitoring the performance impact of the enforced power cap
- RAPL Info defines the minimum and maximum power supported by the given domain, the max time window, and the thermal design power (TDP)

Each MSR bit field is expressed in terms of power, energy, or time units, which are fused into a read-only register called `MSR_RAPL_POWER_UNITS`. For most architectures, the power unit is expressed in $\frac{1}{8}W$, the energy unit is expressed in $15.3\mu J$, and the time unit is expressed in $976\mu s$.

MSRs in any domain can be accessed with `rdmsr` and `wrmsr`, assuming the `msr` kernel confirms the presence of and appropriate permissions for the `/dev/cpu/X/msr` file. Giving users such privileged access to underlying hardware can give rise to major security issues, so Lawrence Livermore National Laboratory (LLNL) has been spearheading an effort known as `msr-safe` [109]. This creates a new `msr-safe` kernel module, which creates a whitelist of MSRs, providing a more secure method to enforce power caps.

RAPL was the first tool of its kind to enable software enforcement of a particular power cap. Since then, other processor manufacturers, such as AMD and IBM, have followed suit and are providing tools for capping power. This will be a vital research platform for future power-constrained research moving forward. Achieving exascale power goals will require assigning power caps to different system components to ensure power is being used wisely, while staying at or below a given system-wide power bound as well as maximizing machine throughput. The RAPL package power domain provides two fields to enforce power limits (and two associated fields to specify the time window), while all other domains provide a single field. By default, the upper power limit (PL2) has a higher-than-TDP power limit over a very short time window to protect the power supply from failing due to high power spikes in application phases. The lower power limit (PL1) is typically set to TDP over a larger time window. Having two power limits provides finer granularity of managing power over an application.

While RAPL and MSRs in general provide a large capability for monitoring and controlling different hardware features of Intel architectures, it is rather difficult for software to handle these MSRs in an efficient manner. One problem is that MSR encodings are architecture-specific. That is to say, the translation between bits and actual values varies slightly between one generation and the next. Additionally, some MSRs are not enabled on all architectures. One other problem is that the same MSR may be of different scope across architectures. For example, on an Ivy Bridge processor, functionality to enforce a CPU frequency is for the package, but on Haswell, functionality changes to a per thread scope. To ease these challenges, LLNL has an ongoing development effort of a library called *libmsr* [110], which provides a wrapper API to directly access various RAPL MSRs agnostic of the target architecture. There are also some optimizations that are implemented in the library, such as batching read and write operations into a single operation to mitigate the overhead of issuing an interrupt per operation per MSR.

RAPL is positioned as the most tractable platform for understanding the constraints of exascale under a power bound. Several studies have shown the accuracy of RAPL's power models as compared to actual measurements [21], [111], [112], [113], [114], [115], [116].

The authors in [44] propose modifying the current RAPL algorithm for the DRAM power domain for smoother power limiting. For each time window, The RAPL algorithm calculates the available power by measuring the current power consumption at discrete time intervals and subtracting this value from the target power utilization. Once the available power is calculated, RAPL looks at a table of M-states corresponding to a particular bandwidth and maximum power consumption, selecting the state that provides the highest bandwidth given the available power. Unfortunately, the M-states define the bandwidth at the worst-case power consumption. Said another way, the power consumption is defined assuming the highest memory traffic, but is not representative of typical application behaviors. This limits the RAPL algorithm's ability to gracefully adapt to the bandwidth behavior of the current phase, wasting power allocation that could be better used somewhere else in the system. To improve this, the authors suggest adding an additional limit to the DRAM RAPL algorithm such that one limit guarantees a specified power cap over the time window, while a second limit adapts the power consumption to the current bandwidth utilization over smaller time intervals.

With the growing demand for fine-grained power measurement capabilities, other vendors are releasing software-based solutions to monitor power of targeted components. **NVIDIA's Management Library (NVML)** provides GPU core temperature, current effective frequency of the GPU, and real-time power consumption measurements [15]. This library is limited in scope, only targeting the server processors, such as the Tesla and Quadro lines. Power readings in milliWatt granularity are gathered with a simple API call to `nvmlDeviceGetPowerUsage`. According to [15], Fermi and Kepler GPU lines will have power readings within $\pm 5\%$ of the physical power reading. Kasichayanula et al. introduced the Activity-based Model for GPUs (AMG) [117], which estimates activity and power consumption with software models similar to those used by RAPL, APM, and EnergyScale, which use prediction models for CPU power consumption.

AMD Application Power Management [13] provides similar functionality to Intel's SpeedStep, TurboBoost, and RAPL technologies. It is included with its processors and uses internal models to make intelligent decisions about how performance should be allocated. Starting with Bulldozer, AMD introduced **TDP Power Cap** technology, enabling software to limit power consumption of the processor by enforcing a desired TDP limit.

IBM EnergyScale [14] provides tools to measure, monitor, and control the power consumption starting with POWER6 systems. Some of the features include the following:

- Power/thermal monitoring: Collect power consumption and temperature of the server, inlet, and exhaust
- Static/dynamic power saver: Reduce frequency by a fixed amount or allow internal algorithms based on performance counters to determine frequencies all in favor of saving power
- Power capping: Enforce power bound on server
- Processor folding: Migrate tasks among cores to maximize energy usage. For example, if the application demands resources, migrate tasks to a larger number of cores. When the demand decreases, reduce the number of working cores, putting superfluous cores to idle or sleep.

In order to collect the power data, Brochard et al. and Knobloch et al. [118], [119] used two IBM-internal system management tools, Amester to measure component-level power and Active Energy Manager (AEM) to measure server-level and chassis-level power.

C. Fine-Grained Infrastructures

To overcome the limited information provided by vendors about their respective power management infrastructures, several tools have been created to physically instrument compute nodes and individual components irrespective of the underlying architecture. **PowerPack** [16] is a hardware and software co-design solution to automate collection of power data from component-level sensors. PowerPack also enables us to do fine-grained attribution of application behavior to the power consumption and energy usage of the system. Further details of the inner workings of PowerPack are described in [83].

Penguin PowerInsight [17] is a commercially-available hardware and software solution designed in close collaboration with Sandia National Laboratory. The hardware sensor modules integrate onto the motherboard by attaching to the voltage rail between the power supply and the motherboard components, such as the CPU or memory. A software utility known as *getRawPower* is provided to gather readings from the sensors via the `ioctl` system call.

PowerMon and PowerMon2 [18] are very similar solutions to Penguin's PowerInsight discussed previously. It is a low cost — at most $150 per device — fine-grained power monitoring device for commodity systems capable of collecting data from eight channels simultaneously at a rate of 1000 samples per second. The PowerMon2 was a more optimized solution to its predecessor, the PowerMon, which had six measurement channels and a sample collection frequency of 20 ms. This optimized device provided even more fine-grained instrumentation with eight measurement channels to sample additional peripherals, such as GPUs, and a maximum collection rate of 3000 samples per second. The PowerMon2 has a smaller form factor than PowerMon, enabling it to better integrate onto the motherboard.

Ilsche et al. [19] created an infrastructure enabling more frequent and fine-grained power monitoring of components. In doing so, cost, size, and scalability were low-level goals. This solution provides power readings for applications executing as quickly as tens of microseconds, and samples are collected every $100\mu s$. While Ilsche et al. provide a plausible solution demonstrating the upper-bound on how quickly and how fine-grained power monitoring can occur, the solution is not viable for a large HPC system due to cost and size. Verification of their solution showed that power measurements of the VRs provide the most fine-grained solution, but are limited in accuracy. Measurements from the power supply to the DC components may be more coarse-grained, but can provide samples at a rate of $100\mu s$.

The **High Definition Energy Efficiency Monitoring (HDEEM)** [20] is very similar to Penguin's PowerInsight, but differs in its hardware and software solution in order to reduce cost and complexity, and improve scalability. HDEEM makes use of an FPGA and baseboard management controller (BMC) already present in the hardware. By leveraging BMC, data can be collected using the standard Intelligent Platform Management Interface (IPMI). The data is sent over PCIe to the FPGA, which is faster than sending the data over ethernet or USB.