DOI: 10.14529/jsfi150205

# Energy-efficient Algorithms for Ultrascale Systems

*Jesus Carretero*[1], *Salvatore Distefano*[2], *Dana Petcu*[3], *Daniel Pop*[3], *Thomas Rauber*[4], *Gudula Rünger*[5], *David E. Singh*[1]



The chances to reach Exascale or Ultrascale Computing are strongly connected with the problem of the energy consumption for processing applications. For physical and economical reasons, the energy consumption has to be reduced significantly to make Ultrascale Computing possible. The research efforts towards energy-saving mechanisms of the hardware have already made energy-aware hardware systems available. However, to achieve a strong energy reduction, hardware mechanisms must be complemented with new energy-efficient software that can exploit them so that the foreseen energy savings actually result. In the software area, there also exist a multitude of research approaches towards energy saving, often concentrating either on the system software level or the application organization level, reflecting the expertise of the corresponding research group. The challenge of reducing the energy consumption dramatically to make Ultrascale Computing possible is so ambitious that a concerted action combining research efforts through all the software levels seems reasonable. In this article, we discuss the current research efforts and results related to energy efficiency in the diverse areas of software. We conclude with open problems and questions concerning energy-related techniques with an emphasis on the application or algorithmic side.

*Keywords: energy-awareness, energy-efficient algorithms, ultrascale computing.*

## Introduction

The performance of high-end HPC systems has been increased roughly by a factor of 1000 in each of the last two decades. With the world's most powerful systems already well past the Petaflop/s level in 2014, a projection of this trend leads to the prediction that by 2022, Exascale computing will be possible. However, progress towards this goal is threatened by energy issues because, based on the current technology, systems with Exascale performance would use excessive amounts of energy (e.g. Tianhe-2, a 33 PFlops system, needs about 18 MW). Moreover, due to physical constraints, the performance of processing elements can no longer be assumed to follow Moore's Law. Accordingly, because of physical constraints and environmental issues, power and energy consumption are considered to be one of the largest challenges for Exascale systems. The US DOE Exascale Initiative has set a target of 20 MW for the power consumption of an Exascale system. To achieve 1 ExaFLOP using 20 MW, the average energy cost per flop must be limited to 20 picojoules (20 pJs/FLOP), including all costs for memory accesses and communication [108]. However, the supercomputers on the current Top500 list need between 300 and 8000 pJs/flop[6].

Consequently, reducing the energy consumption for computing has become an increasingly important research topic in recent years, with the research community following two main research directions: The first direction is concerned with power-aware and thermal-aware hardware design, including low-power techniques on all levels, i.e. the circuit and logic level, the processor, the memory and the interconnects. The second research direction is based on the development

[1]University Carlos III of Madrid, Madrid, Spain
[2]Politecnico di Milano, Milano, Italy
[3]West University of Timisoara, Timisoara, Romania
[4]University Bayreut, Bayreut, Germany
[5]Technical University Chemnitz, Chemnitz, Germany
[6]`www.top500.org`

of power-aware software for the entire software stack, including operating systems, compilers, applications and algorithms. This second direction is the topic of this survey article, in which we summarize important contributions towards energy reduction that can be provided by the system software or the programming model and discuss how these contributions can be used for the construction of energy-efficient algorithms and applications. An important step towards a systematic development of energy-efficient algorithms is the energy-oriented investigation of benchmark programs. As an example, the energy characteristics of benchmark programs such as SPEC CPU and PARSEC are investigated and algorithmic techniques for energy saving are considered. The emphasis of our investigation is on large-scale complex computing systems, which will be referred to as Ultrascale or Exascale systems in the following.

The rest of the article is structured as follows: Section 1 gives a brief overview of the hardware mechanisms that can be used to reduce energy consumption. Section 2 deals with system support for energy efficiency and presents some energy metrics as well as novel energy measurement and power management techniques. Section 3 studies how the programming model and the software development process can support the construction of energy-efficient algorithms and applications. Section 4 considers the energy consumption of algorithms and discusses algorithmic techniques to enhance energy awareness at the programming level. The final section concludes the article with a discussion of important research directions that are crucial for reaching energy efficiency in algorithms.

## 1. Hardware mechanisms for energy saving

Nowadays, computers include different power management techniques which support the reduction of energy consumption. Examples are dynamic voltage frequency scaling (DVFS), clock gating, and power gating. Moreover, the usage of special instructions and specialized coprocessors can also help to reduce energy consumption.

DVFS [4] can reduce the clock frequency and voltage level of different components of the compute node (processors, DRAM memories, etc.) at the expense of some performance degradation. Currently, DVFS is broadly supported by low-power and high performance processors provided by different manufacturers under different names (e.g. *SpeedStep* in Intel processors and *PowerNow* or *Cool 'n' Quiet* in AMD processors). There are three factors that need to be considered when DVFS is applied: (a) the dynamic power, which has a quadratic relationship with frequency-voltage scaling; (b) the static power, which increases exponentially with the voltage; and (c) the performance, which has a linear relationship with the frequency. Because of its negative performance impact, DVFS may only be effective for non CPU-bounded applications, see Section 4.1 for more details.

*Clock Gating* [97] reduces the power consumption by disabling the clock in those parts of the circuit that are idle or, like in the case of flip-flops, maintain a steady state that does not need to be refreshed. The power used to drive the clock signal can represent more than a half of the overall power consumption. Therefore, clock gating can potentially achieve a significant energy reduction. This technique can be controlled both at hardware and software level. Hardware-level approaches typically provide a finer granularity, allowing also to disable components inside a functional block. Software-level approaches are usually applied at entire functional blocks, but they allow more elaborated energy-saving policies.

*Power gating* [96] is a more aggressive approach in which a functional block is disconnected from the power supply, powering off all its components. Nowadays, existing processors contain

clock gating logic managed by a power reduction policy for almost every functional block. For some components clock gating is used in combination with power gating features. Given that the entire functional unit is disconnected, power gating achieves a better power reduction than clock gating. However, given that the functional unit state is erased, it is necessary to provide mechanisms for saving and restoring the states of the functional units, which increases the complexity and complicates resource utilization when applying power gating to active components that need to preserve their state.

The use of special instructions can also help to reduce the energy consumption for compute-intensive applications. Examples are the SIMD vector instructions provided by the AVX (advanced vector extensions) instructions for the x86 architecture or the AES (advanced encryption standard) instructions to support encryption and decryption. Those instructions lead to an effective use of the corresponding transistors, thus reducing the energy consumption per operation [71].

Similarly, the use of specialized coprocessors or accelerators, such as GPU (Graphics Processing Unit), MIC (Many Integrated Cores) or FPGA (Field Programmable Gate Array), can also lead to a smaller energy consumption compared to general purpose CPUs. As an example, the NVIDIA "Fermi" generation of GPUs requires about 200 picojoules of energy to execute one instruction, which is 10x less than for the most efficient x86 CPU.

## 2. System support for energy efficiency

In order to obtain the benefits offered by an Ultrascale or Exascale system, it will be increasingly important to provide system services for an effective management of the system resources on behalf of the applications. Those services can be offered to the applications through the programming environment or through specialized libraries, but they should be as transparent to the user as possible to support application porting and sustainability. As energy is a cross-layer issue, several aspects of the system software and the operating system should be involved in energy efficiency resource management, but it is also paramount to provide metrics and facilities to monitor and express energy at the processor and system level.

### 2.1. Resource management

Currently, power requirements are driving the co-design of HPC systems, which in turn sets the course for a radical change in how to express the need for increasingly scarce resources, as well as how to manage them. Knowing that Ultrascale and Exascale systems will inevitably rely on a high-level heterogeneity of resources and new HPC usage challenges (such as providing performance hand-in-hand with energy efficiency), they need to become more and more self-aware with respect to performance, energy and resilience [36]. New usages, like many-task computing paradigms, will force the system to host, schedule, and load balance millions of heterogeneous tasks. Existing research provides analytical studies quantifying and comparing expected performance of new solutions proposed.

Another approach is to use layered solutions, such as the use of algorithm-specific checkpointing combined with system-level checkpointing [19], or to use imperfect fault predictors [10]. Following this trend, decentralized approaches for a multi-objective, energy-aware resource management will be a likely replacement for centralized approaches when these do not scale up. Gossip-based [65] and hierarchical approaches [124] are examples that have been proposed for

load balancing. However, the scale to which they have been evaluated and the complexity of their balancing requirements is far from what is expected for Exascale.

## 2.2. Energy metrics

In order to properly evaluate a specific system property, it is necessary to define corresponding metrics. With regard to energy, the main basic metric is usually the unit of work or amount of heat transferred, measured in *Joule* ($J$), while the power, i.e. the amount of transferred energy in time, is measured in $Watt$ ($W$).

In the computing system context, several initiatives related to energy measurement and management have been started, mostly grouped under the umbrella of Green IT. Some of them focus on distributed systems, aiming at identifying specific metrics for assessing energy efficiency in these systems. A good example is GreenGrid, which is "an association of IT professionals seeking to dramatically raise the energy efficiency of datacenters through a series of short-term and long-term proposals" [104]. They propose to use two main metrics for evaluating energy efficiency in datacenters: Power Usage Effectiveness (PUE), and Datacenter Infrastructure Efficiency (DCiE) [11, 16]. PUE is defined as follows:

$$PUE = \frac{TotalFacilityEnergy}{ITEquipmentEnergy}$$

while $DCiE$ is specified as its reciprocal:

$$DCiE = \frac{1}{PUE} = \frac{ITEquipmentEnergy}{TotalFacilityEnergy} \times 100\% \quad .$$

The energy for the total facility is the overall amount of energy consumed by the whole data center, including IT systems and facilities. The IT systems energy is the energy consumed by just the IT equipment such as processing, storage, and network components for data management and processing. The facilities include all other subsystems, such as UPS and power management systems, cooling systems, lighting systems, etc.

Other interesting initiatives in the direction towards widely used metrics and, possibly, standards, are Energy Star [110] and SPECpower [64]. Energy Star specifies specific rules, provides a rating for energy efficiency, called the Energy Star score, and is based on SPECpower. SPECpower is mainly a benchmark for evaluating the energy efficiency of server-class compute equipments. Several Performance-per-Power metrics have been proposed which report the ratio between a given performance metric (such as response time, throughput, utilization, delay, bandwidth, etc.) and the energy consumed for obtaining such a performance. An example is the metric transactions per second per Watt (TPS/Watt), using the throughput as performance metrics.

For the particular characteristics of Exascale platforms, specific energy efficiency metrics are not yet specified and a metric that is able to take performance, scalability, as well as energy efficiency into account still needs to be introduced.

## 2.3. Energy measurement techniques

A major challenge for energy measurement and monitoring is their use on heterogeneous platforms through a standard access monitoring interface. Standardized monitoring interfaces for energy and resource utilization are necessary to support local and global control decisions and

should be able to handle the diversity of hardware devices, such as GPUs, embedded CPUs, and nonvolatile low-power memory and storage. An example for a standardized access to performance counters is the PAPI interface, which currently can be used on a large number of platforms including the Intel Core i7 architecture, NVIDIA GPUs, the Intel Xeon Phi and IBM Blue Gene/Q systems [78].

For CPU power monitoring, one approach consists in finding the relationship between the power consumption and the utilization level. The utilization level is computed from different workloads that stress different components of the system (CPU, memory, I/O, etc.). In the literature [31, 81] it has been shown that the power consumption and the utilization level are related linearly, regardless of the type of workload and the configuration of the processor, e.g. in terms of operational frequency or the number of active cores.

As an alternative, the CPU performance can be indirectly modeled by means of hardware counters that capture different hardware events, such as the number of cache accesses or the number of instructions issued [98]. Performance monitoring counters do not require program modifications or an intrusion into the hardware structure and they can accurately reflect the activity levels of the processor or the memory subsystem. An example of this modeling technique is given in [66], where the event-based power prediction is enhanced by using the correlation of the power consumption with the change in core die temperature and the ambient temperature. Recent Intel CPU architectures include the Running Average Power Limit (RAPL) energy sensors to measure the power consumption of different components, including the CPU and the memory controller. The use of these counters is an efficient and low overhead alternative to measure the power of a system using specialized power meters [45]. Energy modeling approaches and a comparison with measured energy values are discussed in [88].

## 2.4. Power management techniques

The Advanced Configuration and Power Interface (ACPI) [26] is an open standard for device power management co-developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. It specifies different global and device energy states, which range from fully operational to completely powered off, and provides an interface to manage and monitor the power of the infrastructure components. ACPI can be accessed by the user with the aid of user-defined policies, such as specifying an application power level, or by the operating system, which applies power policies based on the platform load, such as switching the components to a low power state after a time of inactivity.

There are also advanced tools that provide support for a real-time power management of the infrastructure components, including servers, storage, network, and cooling equipment. Examples are the Intel Datacenter Manager [28], the IBM Systems Director Active Energy Manager [27], and the HP Power Advisor [50]. They provide a single cross-platform view, can be used at multiple hierarchy levels, and support different energy policies, such as power capping, power saving and generation, and the analysis of power history data logs. In addition, most of these tools are fully integrated in the infrastructure management software, allowing it to perform energy-aware tasks, such as workload scheduling.

Several approaches address the improvement of the system energy efficiency. An example is given in [44], where DVFS is used to control the CPU power based on different policies which are applied considering the number of executed instructions, the memory traffic, and the consumer power of the processor. Memscale [33] applies dynamic frequency scaling to the complete out-

of-chip memory subsystem (memory controller, memory channel, and DRAM device), as well as dynamic voltage scaling to the memory controller. It includes a control algorithm that minimizes the overall system energy based on performance counter monitoring. This work was extended [32] to multiple memory devices and controllers. [62] presents an energy model for the execution of a parallel conjugate gradient method split between the CPU and the GPU. The approach considers the CPU, GPU, and RAM energy consumption and uses the information to perform an energy-aware workload distribution minimizing the execution time. A more global approach is followed in [24], where a runtime optimization technique is presented for improving energy efficiency in processors, disks, and networks.

The effectiveness of DVFS is restricted by the range of the minimum and the maximum voltages at which the transistors can operate. Moreover, DVFS is difficult to apply when workloads of different characteristics are executed. To overcome these problems, the idea of complementing DVFS with power gating has been proposed. [75] introduces PGCapping, a system that integrates power gating with DVFS for chip multiprocessors. [1] presents a gating-aware scheduler and a power gating scheme for GPGPU execution units that achieve significant energy saving in simulations.

When considering large computing infrastructures, the power proportionality arises, besides the energy efficiency, as a crucial concept. Power-proportionality means that the system's energy usage is proportional to its workload. In this way, the machine would consume no power in the idle state and would gradually increase the power consumption as the workload increases. An Exascale architecture should be both energy efficient and power proportional. However, existing systems are far from fulfilling this requirement. Consequently, it is necessary to develop new hardware and software tools that help to achieve it [38]. Examples for such tools are described in [105] and [6]. The first one shows a power-proportional distributed storage system for data centers that powers down servers according to the load level and considering the performance degradation, availability and data consistency. The second one presents a distributed filesystem based on the Hadoop DFS. It provides power proportionality minimizing the number of active nodes, including power-proportional capabilities for failures such as minimizing the number of nodes that need to be restored when there is a failure of the filesystem. [47] describes a solution to provide energy proportionality for networks by dynamically adapting the energy consumption of a network through traffic patterns analysis and by finding minimum power network subsets. A survey of techniques that aim to improve the energy efficiency of computing and network resources is given in [80], covering techniques that operate both on parallel and distributed system levels.

## 2.5. Monitoring and Benchmarking

With specific regard to Exascale platforms, there are three main challenges for energy efficiency metrics and monitoring: (1) scalability, (2) standard access monitoring methods, and (3) its application on heterogeneous platforms [53]. Monitoring everything produces extremely large trace files making their analysis prohibitive. Alternatives are statistical models [83], time series approaches [67], and data filtering with a distributed analysis that produces small trace files with a small runtime overhead [60, 84].

At node level, it is crucial to find the relationship between the power consumption and the utilization level computed, which seems to be linear [31, 81]. As discussed above, one possibility is to use hardware counters to model the CPU performance [98] and Intel RAPL to measure the

CPU and memory controller power consumption [45, 66]. At the whole compute infrastructure level, power proportionality arises as a crucial concept [39, 70]. Even if the current hardware components are not power-proportional, we can see in the literature examples of system wide [47, 105] and system specific models to achieve power-proportionality. In any case, standardized monitoring interfaces for energy and resource utilization are needed to handle the diversity of hardware and support local and global control decisions based on well-known and accepted metrics, see Section 2.3.

The energy metrics collected at node and system level must be provided to the operating system and the system software to optimize important energy-consuming operations in extreme-scale systems. One of these operations is data movement, as it is recognized that today data movement and storage uses more power than computation in many HPC usages. As an example, [37] indicates explicitly that managing data movement may be an energy-efficiency technique.

Coupled to monitoring frameworks, benchmarking provides useful and complete tools for the proper evaluation of distributed systems. Many stable benchmarking suites are available for HPC systems, such as the NAS Parallel Benchmarks (NPB) [13] and LINPACK [35], which for example is used for the performance evaluation and comparison of the Top500 list entries, see www.top500.org. There are also some interesting attempts towards standards in benchmarking. The most authoritative ones are the Standard Performance Evaluation Corp (SPEC) [101] and TPC [109]. The Standard Performance Evaluation Corp (SPEC) has developed solutions that can be adopted in distributed and cloud environments, such as SPECvirt, SPEC SOA, and SPECweb. With specific regard to energy, SPEC define the SPECpower_ssj2008 benchmark [64], considering performance and energy efficiency altogether. TPC is a non-profit corporation defining transaction processing and database benchmarks through verifiable TPC performance data to the industry. The TPC benchmarks can be considered as application-level benchmarks in distributed environments and they are a basis for the evaluation of the actual performance offered by standard transactional software on the top of (physical or virtual) machines.

# 3. Programming models and software development

An important aspect for the development of energy-aware applications is the use of suitable programming models. This is the main topic of this section, along with a coverage of energy-aware scheduling algorithms and software development approaches.

## 3.1. Hierarchical programming models

Applications for Exascale computing are expected to incorporate multiple programming models. For example, a single application might incorporate components that are based on MPI and other components that are based on other paradigms. The particular combination of programming models may differ over time (different execution phases of the application) or space (e.g. some of the nodes run MPI, and others run shared-memory libraries). It is widely believed that to cope with these models, Exascale systems will require support for hierarchical programming models, which include more than two levels of today's models (such as MPI + OpenMP) [42]. The particular combination of programming models may differ over time (e.g. different execution phases of the application) or space (e.g. some of the nodes run MPI, and others run shared-memory libraries). It is widely believed that to cope with these models, Exascale systems will require support for hierarchical programming models, which may include

more than two levels of today's models (such as MPI + OpenMP) [42]. In Exascale systems, hierarchies with a higher number of levels and a larger degree of parallelism will coexist with more heterogeneous hardware, making load balancing and communication reduction a critical task. Those features can be addressed through functional portability and performance portability. Even through functional portability can be achieved due to standardized environments such as MPI or OpenCL, performance portability, however, is often a crucial issue, as the required abstractions are still not present in the current HPC code generation tools. Performance portability for future systems might require a durable abstraction expressed in programming models that do not exist for HPC code generation so far [58].

Examples for existing hierarchical programming models are the TwoL [85] and the Tlib [86] approaches, which are both defined on top of MPI and allow a flexible and hierarchical grouping of processes into groups each of which can execute multi-processor tasks (M-tasks). The M-tasks are the basic execution units and each M-task can be executed by an arbitrary number of processing cores. In the TwoL approach, the M-tasks can be combined using a coordination language, which allows the specification of input-output and control dependences between M-tasks. M-tasks without a dependence between them can be executed in parallel on disjoint groups of processors. The runtime system can select a suitable number of processing cores for each M-task and can decide which of the M-tasks are executed in parallel. If the internal M-task communication is based on collective MPI operations, it is often advantageous to execute M-tasks in parallel as this reduces the communication overhead. This approach can also be used to enable an energy-efficient execution of M-task programs [87], since the runtime system can perform the mapping of M-tasks to cores based on an energy minimization instead of a performance maximization goal. It is also possible to provide different implementations for M-tasks, such as a standard MPI implementation, a GPU implementation and a specialized implementation for MIC processors, and select the most energy-efficient implementation at runtime, depending on the hardware resources available. To support such an energy-efficient mapping, it is important that the runtime system has access to suitable monitoring facilities (see Section 2.5) or can use suitable energy metrics (see Section 2.2).

The M-task model can also be used to support performance portability, since the same M-task program can be executed on different hardware platforms and the runtime system is responsible for the appropriate mapping to the hardware resources. For different hardware platforms, the runtime system can select different mappings and different M-tasks could be executed in parallel, if this results in a faster or more energy-efficient execution.

## 3.2. Many task approaches

The ever-increasing performance of supercomputer systems is enabling the emergence of new problem-solving methods that require an efficient execution of many concurrent and interacting tasks, usually integrating data analysis and visualization, to maximise the productivity on Exascale systems [37]. Hence, Exascale systems will need new problem-solving approaches beyond hierarchical models.

One of the most promising candidate approaches is the many-task programming model, with the workflow model currently being the most widely used many task-like technique. An example of these tools is Swift/T, a description language and runtime system that supports the dynamic creation and execution of workflows with varying granularity on high-component-count platforms. The Swift/T system [117] provides an asynchronous dynamic load balancer (ADLB),

which dynamically distributes the tasks among the nodes [119]. The problem is that communication and synchronization for shared global resources (as files) could degrade performance in case of the absence of data locality. Current research has shown that emerging high-speed networks outperform physical disk solutions, which reduces the relevance of disk locality [7]. Thus, most solutions provided for ultrascale will be based on the intensive usage of RAM and NVRAM memory near the processors. However, existing software engineering methods and models do not provide a mechanism to express energy aspects in applications and they still rely on system services that are not energy-aware.

### 3.3. Energy-aware scheduling algorithms

In order to cope with energy saving while considering the particularities of Exascale systems, i.e. various levels of heterogeneity, fault tolerance, strong energy consumption constraints, it is mandatory to move towards an energy-aware resource management [22], including scheduling algorithms that are able to handle various levels of heterogeneity and the diversity of available resources [73].

Power-aware scheduling algorithms for homogeneous systems are already available for more than one decade [46, 51, 72]. Popular approaches commonly use DVFS to reduce the power consumption of processing elements during idle times and during slack times of non-critical jobs [115]. Other approaches even power off the entire computing node with only a small impact on the resulting makespan [76].

In many HPC usage scenarios, data movements consume more power than computations do, so that reducing data movement can be considered an energy-efficiency technique [37]. Therefore, energy-aware scheduling algorithms should guide the system to schedule computation jobs to the nodes containing the required data, thus avoiding costly data movement and considering the trade-offs between data locality and load balance. While traditional task clustering algorithms reduce the makespan by zeroing edges of high communication costs, a Power Aware Task Clustering (PATC) algorithm has recently been proposed [115] that guides the edge zeroing process with the objective of reducing the power consumption. The initial experiments were performed on homogeneous small clusters (100 PEs), where promising results have been obtained, specifically yielding up to 39% energy saving, which is more than double compared to 16% obtained on EADUS and TEBUS algorithms [122] that do not use DVFS. Energy-aware algorithms have also been developed and tested against heterogeneous clusters. The EETCS (Efficient-Energy based Task Clustering Scheduling) algorithm [69] significantly reduces the power consumption by shrinking the communication energy consumption when allocating parallel tasks to heterogeneous computing nodes. Another example is RADS (Resource-Aware Scheduling Algorithm with Duplication) [79], which saves up to 15% resource power consumption compared to similar algorithms.

Current scheduling and load balancing mechanisms are using meta-heuristics to solve the multi-criteria optimization problem taking into account the overload of the system and the incoming task requirements. Traditional multi-objective optimization algorithms, including population based metaheuristics aiming to estimate Pareto optimal sets, require an adaptation in order to be effective in the case of ultrascale dynamic optimization. In [22] a two-stage approach is proposed: First, a list of preliminary schedules resulting from a static multi-criteria optimization method is computed at design time. Then the schedules are adapted, using low cost operations, according to the particular requirements of the running applications and the

characteristics of the available resources. However, the approach has not been tested in the context of large scale dynamic scheduling. Another aspect to be considered is the exploration of the relationship between tasks and computing resources and the proper usage of data location [14]. Existing scheduling techniques for Exascale rely on various combinatorial optimization algorithms. For example, in [103] a new approach is proposed for simultaneously reducing the energy consumption while maximizing system performance. The method consists in computing the Pareto front of optimal solutions to the bi-objective problem of minimizing energy and makespan for a bag of tasks allocated to a set of heterogeneous compute resources.

The ultrascale dimension, the heterogeneous architecture of current parallel systems, and the need to re-schedule due to system faults have not been taken into consideration yet, especially not together with energy awareness. The task scheduler needs to support locality-awareness and be capable of supporting function shipping and data shipping as interchangeable alternatives. For this purpose, all data movement operations need to be abstracted as asynchronous tasks whose completion can trigger additional computation tasks and data movements. Moreover, the current slow meta-heuristic based mechanism should be redesigned to ensure a real-time reaction especially in the case of re-scheduling. A set of strategies, such as minimal energy consumption with deadline matching in scheduling mechanism assuming no faults, or energy aware rescheduling in the case of faults without time limits, should be defined as working conditions for the resource management system.

### 3.4. Energy-aware software development process

In a complex and highly distributed context, energy awareness should be applied at any level, both hardware and software, and within them. It needs to be addressed at different layers and services adopting a holistic approach. With regard to software, energy efficiency and optimization could be implemented and enforced at several levels: (a) at low level, through specific scheduling algorithms; (b) at code level, by optimizing programs and compilers and also by adopting specific, e.g. hierarchical, programming models and design patterns; and (c) at higher levels, in the software development process. In the latter case, the goal is to design the overall software architecture taking into account energy aspects and metrics, thus also considering a possible deployment in an Ultrascale infrastructure for the overall software. This approach comes from software performance engineering [99, 100], which is a systematic, quantitative technique to construct software systems that meet performance objectives. It includes performance requirements and goals into a software development process, a technique also known as performance-driven development [68, 74, 77]. As in the test-driven development [15], the performance-driven development is an iterative process composed of development and performance evaluation phases at each cycle.

The idea of an energy-aware software development process, which aims at enabling and taking into account energy efficiency and other important deployment properties and requirements at the early stages of the software lifecycle, is not new in literature but quite unexplored, especially in large scale parallel and distributed contexts. The first attempt in such a direction is green software engineering [21, 61] and development [2, 95]. All of those approaches mainly suggest adopting a green, sustainable software development process taking into account energy properties, but so far just provide some suggestions and guidelines for this purpose, mainly at lower levels, e.g. code, programming models, or design patterns. A slightly more concrete solution is discussed in [106] where a reference model for sustainable software development,

called GreenRM, is defined according to the ISO/IEC 14001 environmental requirements. But also in this case a model mainly containing only some guidelines is defined. Therefore, addressing energy, green and sustainability issues in the software development process is still an open problem.

# 4. Energy-efficient algorithms

As stated in the introduction, a huge reduction in the average energy cost per flop is required for Exascale systems [108]. There have been large efforts on the hardware side which aim at a reduction of the energy consumption, including new memory systems and new processor technologies with power management, see Section 1. However, while these techniques can help to significantly reduce the energy consumption of unloaded systems, their contribution to the energy consumption of loaded systems is quite limited. Most of the efforts for reducing the energy consumption of loaded systems are directed towards an efficient control of the power management techniques according to the system load, but the contribution of these techniques may not be sufficient to reach the 20 MW target for Exascale systems.

A major problem in current approaches is that the algorithms or the applications being executed have no direct interaction with the hardware system to express or control energy needs. Such an interaction is needed to bring energy-awareness to the application level and to support a goal-directed use of algorithmic changes or transformations of the application code. In this section, we give an overview of the most important aspects for the energy awareness of algorithms, including the energy characteristics of algorithms, the effect of algorithmic changes and transformations on the resulting energy consumption, as well as adaptivity approaches used to cope with the increasing heterogeneity of HPC systems resulting from the integration of accelerators such as GPU, MIC or FPGAs. Finally, we show some specific examples for energy-efficient algorithms from different areas.

## 4.1. Energy characteristics of algorithms

Hardware mechanisms introduced during the last years to reduce the overall energy consumption of processors (see Section 1) will also play an important role for future Ultrascale systems. Thus, it is important to study the influence of these techniques on algorithms and applications. In particular, it has to be investigated whether these techniques can be employed to reduce the energy consumption of algorithms and which specific characteristics of algorithms have an effect on the resulting energy consumption. If the influencing factors are known and can be captured quantitatively, this information can be used to tune applications towards a smaller energy consumption by applying suitable algorithmic transformation techniques.

The energy consumption $E$ of an algorithm can be described by the power consumption $P$ of the execution resources employed and by integrating $P$ over the execution time of the algorithm: $E = \int_{t=t_0}^{t_{max}} P(t)dt$. Typically, the power consumption varies during the execution time of the application, depending on the specific execution situation of the application and the resulting usage of the different execution resources. The variations of the power consumption during the execution time can be measured in detail with specialized power meters and power acquisition systems [90] (see Section 2.2), but hardware counters can be used as well (e.g. Intel RAPL interface). However, the specific interaction of computation and power consumption is

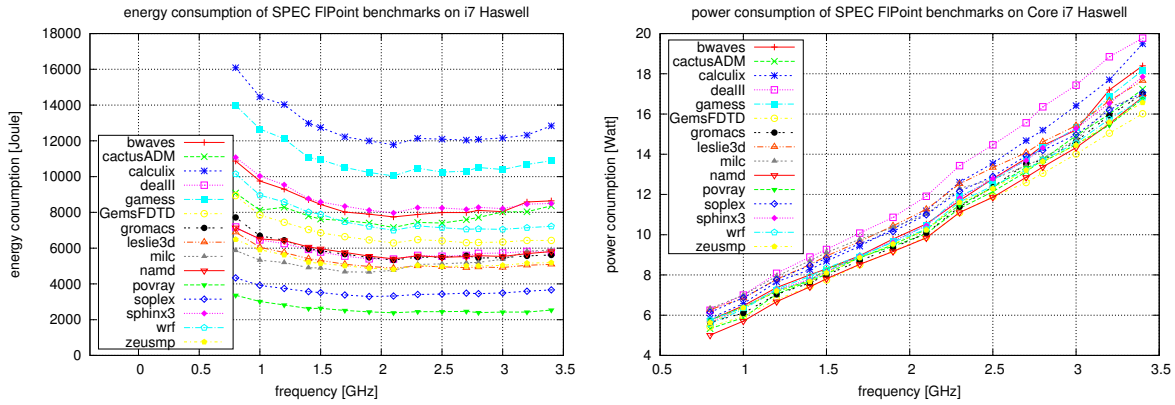


**Figure 1.** SPEC CPU2006 floating-point benchmarks on an Intel Core i7 Haswell processor: energy consumption (left), and power consumption (right) for varying frequencies [90]

complex and it is challenging to predict which algorithmic properties lead to which amount of power consumption at a specific point in the execution time.

The power consumption of processors comprises a dynamic and a static power consumption part [56]. The dynamic power consumption $P_{dyn}$ is related to the switching activity of the processor during execution and it can be expected that it is smaller during processor idle periods. The static power consumption $P_{stat}$ captures the leakage power, which becomes more important for processors with smaller transistor size, and it is present even if there is no switching activity of the transistors. It has been stated that in 2014 25%–40% of the total power consumption in server chips was caused by leakage power [48]. For DVFS processors, the dynamic power consumption increases significantly with the operational frequency $f$, and often, a dependence $P_{dyn}(f) = \gamma \cdot f^{\alpha}$ with $2.5 \leq \alpha \leq 3$ is assumed, where $\gamma$ is a suitable parameter. The dependence of the static power consumption $P_{stat}$ on $f$ is typically quite small and is often neglected and assumed to be constant [56].

The average power consumption of algorithms increases with the operational frequency. Fig. 1 shows the dependence of the energy and the power consumption on the frequency for the SPEC CPU2006 floating-point benchmarks, which consist of real (sequential) programs from different application areas, (see [48] and [90] for more details). It can be observed that for most of the programs a frequency between 2.0 and 2.5 GHz leads to the smallest energy consumption. It can also be observed that different SPEC programs lead to different amounts of power consumption, which shows that there is a dependence of the power consumption on the features of the application. This effect is even larger for parallel applications, as those included in the PARSEC benchmarks that contain parallel programs from different application areas, see [17]. Fig. 2 shows the average energy and power consumption of the PARSEC benchmarks for different frequencies. As shown, the variation of the power consumption is much larger than for the SPEC benchmarks. Fig. 2 also shows that the difference between the largest and the smallest average power consumption for the different applications is more than 100% (see [89] for details). It can be concluded that parallel execution adds significant variations to the power consumptions observed.

The observation that the power consumption may be quite different for different algorithms and applications leads to the question which algorithmic properties have an influence on the resulting power consumption. For parallel applications, the speedup obtained plays a role and

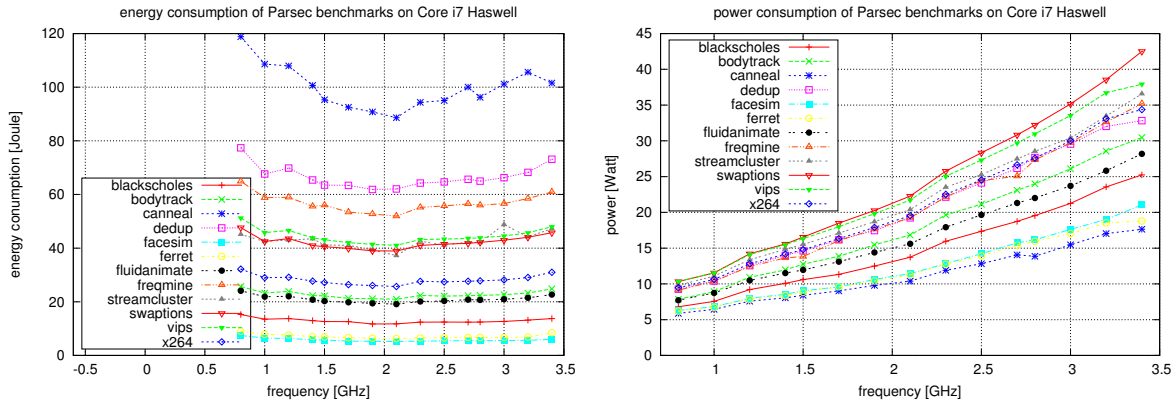


**Figure 2.** PARSEC benchmarks executed with eight threads on an Intel Core i7 Haswell processor: energy consumption (left), and power consumption (right) for varying frequencies [89]

it can be observed that applications with a larger speedup tend to have a larger power consumption than applications with a smaller speedup [90]. This can be explained by the fact that applications with a smaller speedup typically include more idle times during which some parts of the processing cores can be powered down, thus reducing the average power consumption. However, there are other influences that will be discussed in more detail in the next subsection.

## 4.2. Algorithmic techniques towards energy awareness

There are some efforts to explore the energy effects of specific programming techniques for selected algorithms, mainly from the area of linear algebra [5], with the goal of advancing towards an energy optimization of algorithms. Seminal articles in the literature demonstrate that a huge number of technical applications can be decomposed into up to 7 or 13 "Dwarfs" [9], which are a small set of common kernels with a tremendous impact on a huge number of computing-intensive applications and libraries. Thus, it seems advisable to concentrate on those kernels.

Systematic approaches that investigate the energy effects of algorithmic changes and transformations are very rare. Some recent results show that standard techniques used for performance optimization, such as tiling, have only a minor effect on the energy consumption [41], since loading and storing data to the on-chip caches constitute the largest contribution to the dynamic energy consumption. Therefore, alternative techniques, such as register tiling [91], seem to be more promising for the energy optimization of algorithms than standard techniques used for performance optimization. Currently, it is not feasible to think of a single solution for the energy optimization of algorithms, as the energy behavior of the algorithms is closely related to specific architectures.

Several approaches model the energy consumption of application programs on CPUs or GPUs [23]. These models usually distinguish between the dynamic and the static power consumption, but they do not take algorithmic properties of the application into consideration. There are also some approaches that model the energy consumption of individual algorithms by considering the operations performed [59], however these approaches are difficult to transfer to other algorithms and they require a significant effort for the analysis at the algorithmic level. Another attempt in finding a relation between properties of the algorithms and the resulting energy consumption and execution time is described in [25], but the results are only presented

at the level of micro-benchmarks. So far, there is no broad investigation that determines which algorithmic properties have which effect on the energy consumption for a specific architecture. Thus, there is a need to develop algorithm-specific energy models and mechanisms to express the energy behavior of the algorithms on the underlying system. A survey of power and efficiency issues for numerical linear algebra methods [102] identifies several major techniques for energy savings, e.g. profiling, trading off performance, static and dynamic saving, and concludes that the current techniques are application-specific and difficult to generalize. The impact of different CPU workloads on power consumption and energy efficiency is studied in [111], showing that different workloads can lead to significant differences in energy efficiency.

In addition, the architecture of different HPC and Exascale systems is expected to be quite heterogeneous and rapidly developing [37], as they might include specialized niche market devices, such as GPUs, MIC and FPGA accelerators. This perspective constitutes a major challenge for the system software, comprising the operating system, runtime system, I/O system, and interfaces to the external environment, since the system software is responsible for an effective use of the hardware resources. However, algorithmic properties of an application also play an increasingly important role and it is required that the programmer uses the right programming techniques for the specific architecture of a given HPC system. This places a large burden on the programmer to tune her or his applications towards a better performance. Since this is often quite time-consuming, autotuning approaches [114] and efforts towards Self-Adapting Numerical Software (SANS) [34] have been proposed. Those aspects will be considered in more detail in the next subsection.

### 4.3. Autotuning approaches towards energy efficiency

Autotuning software is able to optimize its own execution parameters with respect to a specific objective function, which was usually the execution time, but might as well be the energy consumption. The methods for autotuning are diverse, including model-based parameter optimization, or an optimization based on candidate sets generated by the autotuning software. Autotuning based on a set of equivalent candidate implementations for an algorithm considers different candidate implementations using different programming techniques for the formulations of the algorithm, which, for example, may differ in their loop structure by applying loop transformations such as loop fusion, loop interchange, loop tiling, or loop unrolling. Moreover, different parameters for the loop transformation, such as block sizes for tiling or unrolling factors, can be used. The idea of the autotuning approaches is to automatically select one of the candidate implementations for a specific HPC architecture to reach a given optimization goal, such as minimal execution time or minimal energy consumption. The selection can be made both offline or online.

*Offline* autotuning performs the autotuning procedure at software installation time. In this scenario, the installation of the autotuning software or library can take a significant amount of time due to an extensive evaluation of the different candidate implementations using runtime tests or energy measurements. However, at runtime, the best implementation variant selected during the installation is directly used, with little or no overhead. Offline autotuning can be applied if there is no significant dependence of the runtime of the implementation variants on characteristics of the specific input. A number of offline autotuning libraries aiming at performance optimization already exist for decades: ATLAS [116] and PHiPAC [18] for dense matrix computations; OSKI [113] and SPARSITY [52] for sparse matrix computations; or FFTW [40]

for fast Fourier transformations. Offline frameworks, such as PERI [118], SPIRAL [82] and Green [12], allow the programmer to setup an application to be autotuned for a given micro-architecture. If supported by a model-based approach [121], the installation time overhead can be reduced. Model-based approaches use an analytical model of the execution platform and the algorithm to be executed, and select a set of implementation variants and parameter values which are then tested at installation time, which may reduce the number of variants to be tested significantly.

Besides the overall execution time of a specific algorithm, additional optimization goals, such as energy consumption or computing costs, need to be considered by auto-tuners. Therefore, more sophisticated methods capable of exploiting and identifying the trade-offs among these goals are required, like those presented in [43] where the authors present and discuss results of applying a multi-objective search-based auto-tuner to optimize for three conflicting criteria: execution time, energy consumption, and resource usage. Offline autotuning approaches for energy usage vs. performance degradation in scientific applications are discussed in [107], where the authors conduct several experiments in which the tuning is performed with respect to software level performance-related tunables, such as cache tiling factors and loop un-rolling factors, as well as for the processor clock frequency. [63] presents an energy-oriented autotuning for the ATLAS library.

If the execution time of the implementation variants depends on characteristics of the specific input, offline autotuning has to be replaced by *online* autotuning, where applications are able to monitor and automatically tune themselves to optimize a particular objective (execution time, energy consumption, etc.), as in the case shown for ordinary differential equations in [55]. Online autotuning can especially be used successfully for time-stepping methods. In this case, the time steps can be performed with different implementation variants and parameter values until the best implementation variant is found. Then this implementation variant is used for the remaining time steps, as shown in [62]. A model-based pre-selection phase can be used to reduce the number of implementation variants that need to be tested at runtime. For ordinary differential equations, this approach has been applied successfully [55], and it has been shown that the autotuning overhead at runtime is not too large. An automated online performance tuning approach for general applications is provided by the Active Harmony automated runtime system [29], which allows runtime switching of algorithms and tuning of libraries and application parameters to improve the resulting performance on a given hardware platform. The system uses a server which uses a Nelder-Mead method to search through a potentially large parameter space. The server sends a parameter selection to a client, which then measures the resulting performance and sends the corresponding information back to the server. This procedure is repeated until a good parameter selection has been found.

Another example for online autotuning is PowerDial [49], which converts static configuration parameters that already exist in a program into dynamic knobs that can be tuned at runtime, with the goal of trading QoS guarantees for meeting performance and power usage goals. The system uses an online learning stage to construct a linear model of the choice configuration space which can be subsequently tuned using a linear control system. In the SiblingRivalry [8] model, requests are processed by dividing the available cores in half, and processing two identical requests in parallel on each half. Half of the cores are devoted to a known program configuration, while the other half of the cores are used for an experimental program configuration chosen using a self-adapting evolutionary algorithm. The faster configuration (either the known or

the experimental one) is always kept and the other one is terminated. The authors show that over time, this model allows programs to adapt to changing dynamic environments and often outperform the original algorithm that uses the entire system.

As mentioned before, most of existing autotuning models consider the execution time as main objective function. However, the resulting energy consumption can also be directly used as an optimization goal of an autotuning approach. This can be based on energy measurements using hardware counters as they are, for example, provided by the Intel RAPL interface (see Section 2.5) or on a model for the energy consumption of the algorithm (see [62] for more information).

## 4.4. Examples of energy-efficient algorithms

Examples of energy-efficient algorithms can be found in the graph theory area. In [20], the authors propose a new algorithm which solves the min cut/max flow problem on a graph. It is based on augmenting paths and building two search trees, one from the source and the other from the sink, which are reused to avoid rebuilding them from scratch. Experimental comparisons show that the algorithm is faster and minimizes the energy usage for functions in vision. Another example is [94], in which a large-scale energy-efficient graph traversal is proposed. More recently, the initiative "EDGAR: Energy-efficient Data and Graph Algorithms Research" of the Berkeley Labs has been started to design new parallel algorithms to reduce communication costs of data and graph analysis algorithms in Exascale, aiming at a reduction of the execution time and the energy consumption. An important observation in this context is that the power required to transmit data in a network also depends on the length of the wire in traditional cooper networks, i.e., data exchanges between neighboring nodes in a network require less energy than exchanges between non-neighboring nodes. The energy consumption of different MPI collective communication operations has been investigated in [112], showing that the size of the execution platform plays an important role. A quantitative analysis of the energy costs of data movements between different levels of a memory hierarchy (main memory, L3, L2 and L1 cache) has been reported in [57]. The analysis is based on a set of micro-benchmarks that continuously access data stored in a given level of the memory hierarchy and measure the resulting energy consumption. An experimental evaluation captures several benchmarks, including the NAS parallel benchmarks suite and applications from the Exascale Co-Design centers. The results show that, in current systems, scientific applications spend between 18% and 40% of their total dynamic energy in moving data and between 19% and 36% in stalled cycles. The energy consumption of different data access patterns in PGAS (Partitioned Global Address Space) models has been investigated in [54].

Sorting algorithms are among the most important fundamental algorithms in computer science and many applications depend on efficient sorting techniques. Energy efficiency also plays an important role here and using an energy-efficient sorting could help in reducing the overall energy consumption significantly. The energy consumption of different basic sorting algorithms such as odd-even sort, shellsort, or quicksort has been investigated in [123], showing that quicksort leads to the smallest energy consumption and that the choice of a suitable recursion depth for quicksort may have a large influence on the energy consumption. An external sort benchmark JouleSort for evaluating the energy efficiency of a wide range of computer systems from clusters to handhelds is described in [92]. The energy consumption of vector and matrix operations as well as sorting and graph algorithms is investigated in [93], showing that the energy

consumption depends on the memory parallelism that the algorithms exhibit for a given data layout.

Other examples of energy-efficient algorithms can be found in thread scheduling [30], financial applications [3], and big data applications [120]. All these research efforts use memorization as a techniques to avoid repeating computation by caching previous results, thus achieving a better energy efficiency in application execution.

## 5. Discussion

The summarizing state-of-the-art analysis of energy-aware programming has shown that there already exists a multitude of research directions and results in many areas of computing. From this current research situation, we can derive a number of open problems to be solved for a successful energy-aware programming. As energy is a cross-layer issue, we argue that a holistic energy-aware approach is needed, which requires the development of interacting interfaces between the different software and hardware layers. Such an approach will allow researchers to investigate different directions of the ETP4HPC agenda. Three of these directions are addressed below: new energy-aware algorithms for Exascale, software engineering for extreme parallelism and energy-aware systems support for managing extreme scale systems.

New energy-aware algorithms for Exascale: Advancing the state-of-the-art at an algorithmic level needs to include energy-awareness into the algorithm/application level. One way of achieving this is the introduction of interacting interfaces between the different hardware and software layers, combined with algorithm-specific mathematical energy models. We argue that this will enable a dynamic adjustment of the computation and communication characteristics of algorithms/applications with the goal to achieve a perceivable reduction of the overall energy consumption. Such a new layered approach with its interacting interfaces will also allow a direct interaction between the control of the power management and the algorithm or application being executed. With the aid of annotations, applications may provide a parameterized energy model which can be exploited to articulate a policy for managing trade-offs on different system architectures. A general goal is that future energy-aware algorithms should not only be evaluated based on FLOPs but also based on energy cost of operations.

Software engineering for extreme parallelism: To hide the complexity of the development process of algorithms and applications for Exascale systems, we propose to develop a high level language environment supporting an energy-aware software development. This language environment should be intuitive and easy to handle for application programmers from diverse application areas. This can be achieved by using a human-like language or a descriptive or graphic annotation approach. For an increase of the acceptance and usability, it is important that such a language environment allows a seamless integration of different programming models, accompanied by support for a hierarchical development of all necessary Exascale system coordination, control and monitoring functions in a reasonably human-understandable way. It necessarily should provide energy consumption indicators which system designers and developers can rely on during software development so that they can achieve a reduction of the energy footprint of the resulting program code. Considering the heterogeneity of Exascale systems, a high-level software development process is needed in order to allow a seamless integration of multiple energy-aware programming models beyond the state-of-the-art. We propose a research agenda in this field targeted towards abstract hierarchical programming models and optimized many-task programming models. The first direction will allow the annotation of power and en-

ergy consumption information by defining energy patterns and constraints in the hierarchical programming model. Based on this abstract model, one can build a general hierarchical optimization technique for collective communication algorithms, such as MPI operations, which will not be platform specific but will address the scale of the HPC platform. The second direction should evolve existing programming models to enable locality-based optimizations through the intensive usage of RAM and NVRAM memory near the processors, thus avoiding data movements, along with an energy aware scheduling that will guide the system to schedule computation jobs in the nodes containing the required data taking also into account the trade-offs between data locality and load balance.

Energy-aware system support for managing extreme scale systems: Expressing the cross-layered nature of energy can be achieved by providing system mechanisms that support energy efficiency in extreme scale systems. The first research topic is the design of metrics and tools for exporting energy features, at node and system level, to the applications through (approximate) energy monitoring and management services. These services will be provided to the upper levels of the hierarchy to allow optimizations in runtime resources, libraries and applications. The second topic should investigate the elaboration of energy-efficient data access and communication models relaying on a better exploitation of data locality and layout, and supporting the development of cross-layer locality-aware I/O software. Equally promising and complementary to the previous topics, researchers should look into energy profiling at component and application level in order to dynamically redirect the workload to those components that can yield the maximum amount of throughput. Ultimately, it should be possible to predict the energy consumption of particular code segments. This information can be used to enable a dynamic provisioning of resources, to provide the ability to manage new important resources, such as power and data motion, through an energy aware scheduler and dispatcher, and an energy-aware load balancer that is conscious of the system energy, node energy, and data-locality needs. Last but not least, we need to elaborate novel energy-aware models, APIs and tools to automatically map applications onto heterogeneous architectures trying to optimize performance over energy ratio.

*The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)".*



## References

1. M. Abdel-Majeed, D. Wong, and M. Annavaram. Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs. In *Proc. of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 111–122, New York, NY, USA, 2013. ACM. DOI: 10.1145/2540708.2540719.

2. S. Afzal, M.F. Saleem, F. Jan, and M. Ahmad. A Review on Green Software Development in a Cloud Environment Regarding Software Development Life Cycle (SDLC) Perspective. *International Journal of Computer Trends and Technology (IJCTT)*, 4(9), 2013.

3. G. Agosta, M. Bessi, E. Capra, and C. Francalanci. Dynamic memorization for energy efficiency in financial applications. In *Proc of the 2011 Int. Green Computing Conference*

*and Workshops (IGCC)*, pages 1–8, July 2011.

4. S. Albers. Energy-efficient Algorithms. *Commun. ACM*, 53(5):86–96, May 2010. DOI: 10.1145/1735223.1735245.

5. J.I. Aliaga1, H. Anzt, M. Castillo, J. C. Fernandez, G. Leon, J. Perez, and E.S. Quintana-Orti. Unveiling the performance-energy trade-off in iterative linear system solvers for multithreaded processors. *Concurrency and Computation: Practice and Experience*, 26(17), 2014.

6. H Amur, J Cipar, V Gupta, and GR Ganger. Robust and Flexible Power-Proportional Storage. *ACM Symposium on Cloud Computing (SOCC)*, 2010.

7. G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Disk-locality in Datacenter Computing Considered Irrelevant. In *Proc. of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 12–12, Berkeley, CA, USA, 2011. USENIX Association.

8. J. Ansel, M. Pacula, Y. Wong, C. Chan, M. Olszewski, U. O'Reilly, and S. Amarasinghe. SiblingRivalry: online autotuning through local competition. In *Proc. of the 2012 Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, pages 91–100. ACM, 2012.

9. K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

10. G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni. Checkpointing Strategies with Prediction Windows. In *Proc. of the 19th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 1–10. IEEE, Dec 2013.

11. V. Avelar, D. Azevedo, and A. French. PUE$^{TM}$: A Comprehensive examination of the metric, http://www.thegreengrid.org/~/media/WhitePapers/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf?lang=en , 2012.

12. W. Baek and T. Chilimbi. Green: A Framework for Supporting Energy-conscious Programming Using Controlled Approximation. In *Proc. of the 2010 ACM SIGPLAN Conf. on Programming Language Design and Implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM. DOI: 10.1145/1806596.1806620.

13. D.H. Bailey, E. Barszcz, L. Dagum, and H.D. Simon. NAS parallel benchmark results. *Parallel Distributed Technology: Systems Applications, IEEE*, 1(1):43–51, 1993.

14. O. Beaumont and L. Marchal. What Makes Affinity-Based Schedulers So Efficient ?, https://hal.inria.fr/hal-00875487. October 2013.

15. Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

16. C. Belady, A. Rawson, J.Pfleuger, and T. Cader. Green Grid Data Center Power Efficiency Metrics: PUE and DCIE, 2008.

17. C. Bienia, S. Kumar, J.P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the 17th Int. Conf. on Parallel Architectures and Compilation Techniques*, October 2008.

18. J. Bilmes, K. Asanovic, C. Chin, and J. Demmel. Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology. In *Proc. of the 11th Int. Conf. on Supercomputing*, ICS '97, pages 340–347, New York, NY, USA, 1997. ACM. DOI: 10.1145/263580.263662.

19. W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra. Extending the scope of the Checkpoint-on-Failure protocol for forward recovery in standard MPI. *Concurrency and computation: Practice and experience*, 25(17):2381–2393, 2013.

20. Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

21. C. Calero and M. Piattini, editors. *Green in Software Engineering*. Springer, 2015. ISBN 978-3-319-08580-7.

22. H. Casanova, Y. Robert, and U. Schwiegelshohn. Algorithms and Scheduling Techniques for Exascale Systems (Dagstuhl Seminar 13381). *Dagstuhl Reports*, 3(9):106–129, 2014. DOI: 10.4230/DagRep.3.9.106.

23. H. Chen and W. Shi. Power Measurement and Profiling. In I. Ahmad and S. Ranka, editors, *Handbook of Energy-Aware and Green Computing*, pages 649–674. CRC Press, 2012.

24. G.L.T. Chetsa, L. Lefevre, J. Pierson, P. Stolf, and G. Da Costa. Beyond CPU Frequency Scaling for a Fine-grained Energy Control of HPC Systems. In *Proc. of the 24th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 132–138, Oct 2012. DOI: 10.1109/SBAC-PAD.2012.32.

25. J. Choi, M. Dukhan, X. Liu, and R.W. Vuduc. Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks. In *Proc. of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 447–457, 2014. DOI: 10.1109/IPDPS.2014.54.

26. ACPI Promoters Corporation. Advanced configuration and power interface specification. Technical report, ACPI Promoters Corporation, 11 2013.

27. IBM Corporation. IBM Systems Director Active Energy Manager. http://lwww.ibm.com /systems/director/aem. Accessed January 16, 2015.

28. Intel Corporation. Intel Datacenter Manager Energy Director. http://www.intel.com/content /www/us/en/software/intel-energy-director-product-detail.html. Accessed January 16, 2015.

29. C. Ţăpuş, I-H. Chung, and J. Hollingsworth. Active Harmony: Towards Automated Performance Tuning. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, SC '02, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

30. H. Cui, J. Wu, C. Tsai, and J. Yang. Stable Deterministic Multithreading Through Schedule Memorization. In *Proc. of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–13, Berkeley, CA, USA, 2010. USENIX Association.

31. W. Dargie. A Stochastic Model for Estimating the Power Consumption of a Processor. *IEEE Transactions on Computers*, PP(99):1–1, 2014. DOI: 10.1109/TC.2014.2315629.

32. Q. Deng, D. Meisner, A. Bhattacharjee, T.F. Wenisch, and R. Bianchini. MultiScale: Memory System DVFS with Multiple Memory Controllers. In *Proc. of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12, pages 297–302, New York, NY, USA, 2012. ACM. DOI: 10.1145/2333660.2333727.

33. Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, and R. Bianchini. MemScale: Active Low-power Modes for Main Memory. In *Proc. of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 225–238, New York, NY, USA, 2011. ACM. DOI: 10.1145/1950365.1950392.

34. J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. E. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. S. Vadhiyar. Self-adapting Numerical Software (SANS) Effort. *IBM J. Res. Dev.*, 50(2/3):223–238, March 2006. DOI: 10.1147/rd.502.0223.

35. J. Dongarra, P. Luszczek, and A. Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.

36. G.C. Durelli, M. Pogliani, A. Miele, C. Plessl, H. Riebler, M.D. Santambrogio, G. Vaz, and C. Bolchini. Runtime Resource Management in Heterogeneous System Architectures: The SAVE Approach. In *Proc. of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 142–149. IEEE, Aug 2014.

37. J. Dongarra et al. The International Exascale Software Project Roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, February 2011. DOI: 10.1177/1094342010391989.

38. P. Kogge et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, 2008.

39. EU. *European technological Platform for High Performance Computing, Vision White paper*, 2012.

40. M. Frigo and S. G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

41. E. Garcia, J. Arteaga, R. Pavel, and G. Gao. Optimizing the LU Factorization for Energy Efficiency on a Many-Core Architecture. In *Proc. of the 26th Int. Workshop on Languages and Compilers for Parallel Computing (LCPC 2013)*, pages 237–251. Springer LNCS 8664, 2013.

42. W. Gropp and M. Snir. Programming for Exascale Computers. *Computing in Science and Engineering*, 15(6):27–35, 2013. DOI: 10.1109/MCSE.2013.96.

43. P. Gschwandtner, J. Durillo, and T. Fahringer. Multi-Objective Auto-Tuning with Insieme: Optimization and Trade-Off Analysis for Time, Energy and Resource Usage. In Fernando Silva, Inês Dutra, and Vítor Santos Costa, editors, *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 87–98. Springer International Publishing, 2014. DOI: 10.1007/978-3-319-09873-9_8.

44. Shin gyu K., Chanho C., Hyeonsang E., H.Y. Yeom, and Huichung B. Energy-Centric DVFS Controling Method for Multi-core Platforms. In *2012 SC Companion: High Per-*

*formance Computing, Networking, Storage and Analysis (SCC)*, pages 685–690, Nov 2012. DOI: 10.1109/SC.Companion.2012.94.

45. M. Hähnel, B. Döbel, M. Völp, and H. Härtig. Measuring Energy Consumption for Short Code Paths Using RAPL. *SIGMETRICS Perform. Eval. Rev.*, 40(3):13–17, January 2012. DOI: 10.1145/2425248.2425252.

46. Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira, Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '05, pages 186–195, New York, NY, USA, 2005. ACM. DOI: 10.1145/1065944.1065969.

47. B Heller, S Seetharaman, P Mahadevan, Y Yiakoumis, P Sharma, S Banerjee, and N McKeown. ElasticTree: Saving energy in data center networks. *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 17–17, 2010.

48. J.L. Hennessy and D.A. Patterson. *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

49. H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic Knobs for Responsive Power-aware Computing. *SIGPLAN Not.*, 46(3):199–212, March 2011. DOI: 10.1145/1961296.1950390.

50. HP. HP Power Advisor A tool for estimating power requirements of HP enterprise solutions. Technical report, Hewlett-Packard Development Company, 01 2013.

51. C.-H. Hsu and Wu chun Feng. A power-aware run-time system for high-performance computing. In *Proc. of the ACM/IEEE Conference on Supercomputing*, pages 1–1, Nov 2005. DOI: 10.1109/SC.2005.3.

52. E. Im and K. Yelick. Optimizing Sparse Matrix Computations for Register Reuse in SPARSITY. In V.. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, editors, *Computational Science — ICCS 2001*, volume 2073 of *Lecture Notes in Computer Science*, pages 127–136. Springer Berlin Heidelberg, 2001. DOI: 10.1007/3-540-45545-0_22.

53. K. Iskra, K. Yoshii, R. Gupta, and P. Beckman. Power Management for Exascale, 2012.

54. S. Jana, J. Schuchart, and B. Chapman. Analysis of Energy and Performance of PGAS-based Data Access Patterns. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, PGAS '14, pages 15:1–15:10, New York, NY, USA, 2014. ACM. DOI: 10.1145/2676870.2676882.

55. N. Kalinnik, M. Korch, and T. Rauber. Online auto-tuning for the time-step-based parallel solution of ODEs on shared-memory systems. *Journal of Parallel and Distributed Computing*, 74(8):2722–2744, 2014.

56. S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan & Claypool Publishers, 2008.

57. G. Kestor, R. Gioiosa, D. Kerbyson, and Hoisie A. Quantifying the energy cost of data movement in scientific applications. In *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC*, pages 56–65, 2013. DOI: 10.1109/IISWC.2013.6704670.

58. P. Kogge and J. Shalf. Exascale Computing Trends: Adjusting to the New Normal for Computer Architecture. *Computing in Science and Engineering*, 15(6):16–26, November 2013. DOI: 10.1109/MCSE.2013.95.

59. V.A. Korthikanti and G. Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In *SPAA '10: Proc. of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 157–165, New York, NY, USA, 2010. ACM. DOI: 10.1145/1810479.1810510.

60. I. Koutsopoulos and M. Halkidi. Measurement aggregation and routing techniques for energy-efficient estimation in wireless sensor networks. In *Proc. of the 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pages 1–10, May 2010.

61. P. Lago, R. Kazman, N. Meyer, M. Morisio, H.A. Müller, and F. Paulisch. Exploring Initial Challenges for Green Software Engineering: Summary of the First GREENS Workshop, at ICSE 2012. *SIGSOFT Softw. Eng. Notes*, 38(1):31–33, January 2013. DOI: 10.1145/2413038.2413062.

62. J. Lang and G. Rünger. An Execution Time and Energy Model for an Energy-aware Execution of a Conjugate Gradient Method with CPU/GPU Collaboration. *J. Parallel Distrib. Comput.*, 74(9):2884–2897, September 2014. DOI: 10.1016/j.jpdc.2014.06.001.

63. J. Lang, G. Rünger, and P. Stöcker. Towards energy-efficient linear algebra with an ATLAS library tuned for energy consumption. In *The 2015 International Conference on High Performance Computing and Simulation (HPCS 2015)*, 2015.

64. K.-D. Lange, M.G. Tricker, J.A. Arnold, H. Block, and S. Sharma. SPECpower_Ssj2008: Driving Server Energy Efficiency. In *Proc. of the 3rd ACM/SPEC International Conference on Performance Engineering*, ICPE '12, pages 253–254, New York, NY, USA, 2012. ACM. DOI: 10.1145/2188286.2188329.

65. E. Levy, A. Barak, A. Shiloh, M. Lieber, C. Weinhold, and H. Härtig. Overhead of a Decentralized Gossip Algorithm on the Performance of HPC Applications. In *Proc. of the 4th Int. Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '14, pages 10:1–10:7, New York, NY, USA, 2014. ACM. DOI: 10.1145/2612262.2612271.

66. A. Lewis, S. Ghosh, and N.-F. Tzeng. Run-time Energy Consumption Estimation Based on Workload in Server Systems. In *Proc. of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

67. G. Li and Y. Wang. Automatic ARIMA modeling-based data aggregation scheme in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2013(1):1–13, 2013.

68. J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in Clouds. In *CLOUD '09: Proc. of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 15–22, Washington, DC, USA, 2009. IEEE Computer Society. DOI: 10.1109/CLOUD.2009.5071528.

69. W. Liu, H. Li, Wei Du, and F. Shi. Energy-Aware Task Clustering Scheduling Algorithm for Heterogeneous Clusters. In *Proc. of the 2011 IEEE/ACM Int. Conf. on Green Computing and Communications*, GREENCOM '11, pages 34–37, Washington, DC, USA, 2011. IEEE Computer Society. DOI: 10.1109/GreenCom.2011.14.

70. P. Llopis, J.G. Blas, F. Isaila, and J. Carretero. Survey of Energy-Efficient and Power-Proportional Storage Systems. *The Computer Journal*, 2013. DOI: 10.1093/comjnl/bxt058.

71. M. Lorenz, P. Marwedel, T. Dräger, G. Fettweis, and R. Leupers. Compiler based exploration of DSP energy savings by SIMD operations. In *Proc. of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair*, pages 838–841, 2004. DOI: 10.1145/1015090.1015314.

72. Jiong Luo, Li-Shiuan Peh, and Niraj Jha. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. In *Proc. of the Conf. on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 11150–, Washington, DC, USA, 2003. IEEE Computer Society.

73. T. M. Lynar, R. D. Herbert, S. Chivers, and W. J. Chivers. Resource allocation to conserve energy in distributed computing. *Int. J. Grid Util. Comput.*, 2(1):1–10, 2011.

74. H. Ma. *QoS-driven composition analysis for component-based system development.* PhD thesis, Computer Science Department, Richardson, TX, USA, 2007. Adviser-Yen, I-Ling.

75. K. Ma and X. Wang. PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 13–22, New York, NY, USA, 2012. ACM. DOI: 10.1145/2370816.2370821.

76. O. Mämmelä, M. Majanen, R. Basmadjian, H. De Meer, A. Giesler, and W. Homberg. Energy-aware job scheduler for high-performance computing. *Computer Science - Research and Development*, 27(4):265–275, 2012. DOI: 10.1007/s00450-011-0189-6.

77. E. Mancini, U. Villano, N. Mazzocca, M. Rak, and R. Torella. Performance-Driven Development of a Web Services Application using MetaPL/HeSSE. In *PDP '05: Proceedings of the 13th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pages 12–19, Washington, DC, USA, 2005. IEEE Computer Society. DOI: 10.1109/EMPDP.2005.31.

78. H. McCraw, J. Ralph, A. Danalis, and J. Dongarra. Power Monitoring with PAPI for Extreme Scale Architectures and Dataflow-based Programming Models. In *Workshop on Monitoring and Analysis for High Performance Computing Systems Plus Applications (HPCMASPA 2014), IEEE Cluster 2014*, pages 385–391, Sept 2014.

79. J. Mei, K. Li, and K. Li. A Resource-aware Scheduling Algorithm with Reduced Task Duplication on Heterogeneous Computing Systems. *J. Supercomput.*, 68(3):1347–1377, June 2014. DOI: 10.1007/s11227-014-1090-4.

80. A. Orgerie, M. Dias de Assuncao, and L. Lefevre. A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014. DOI: 10.1145/2532637.

81. M. Pedram and Inkwon Hwang. Power and Performance Modeling in a Virtualized Server System. In *Proc. of the 39th International Conference on Parallel Processing Workshops (ICPPW)*, pages 520–526. IEEE, Sept 2010. DOI: 10.1109/ICPPW.2010.76.

82. M. Püschel, J.M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R.W. Johnson, and N. Rizzolo. SPIRAL: Code Generation for DSP Transforms. *Proceedings of the IEEE*, 93(2):211–215, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

83. R. Rajagopalan and P.K. Varshney. Data-aggregation techniques in sensor networks: A survey. *IEEE Communications Surveys Tutorials*, 8(4):48–63, 2006. DOI: 10.1109/COMST.2006.283821.

84. V. Rapp and K. Graffi. Continuous Gossip-Based Aggregation through Dynamic Information Aging. In *Proc. of the 22nd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–7, July 2013.

85. T. Rauber and G. Rünger. A Transformation Approach to Derive Efficient Parallel Implementations. *IEEE Transactions on Software Engineering*, 26(4):315–339, 2000.

86. T. Rauber and G. Rünger. Tlib - A Library to Support Programming with Hierarchical Multi-Processor Tasks. *Journal of Parallel and Distributed Computing*, 65(3):347–360, 2005.

87. T. Rauber and G. Rünger. Towards an Energy Model for Modular Parallel Scientific Applications. In *IEEE International Conference on Green Computing and Communications (GreenCom 2012)*, pages 523–532. IEEE, 2012. DOI: 10.1109/GreenCom.2012.79.

88. T. Rauber and G. Rünger. Modeling and Analyzing the Energy Consumption of Fork-Join-based Task Parallel Programs. *Concurrency and Computation: Practice and Experience*, 27(1):211–236, 2015. DOI: 10.1002/cpe.3219.

89. T. Rauber, G. Rünger, and M. Schwind. Energy Measurement and Prediction for Multithreaded Programs. In *Proc. of the High Performance Computing Symposium*, HPC '14, pages 20:1–20:9, San Diego, CA, USA, 2014. Society for Computer Simulation International.

90. T. Rauber, G. Rünger, M. Schwind, H. Xu, and S. Melzner. Energy Measurement, Modeling, and Prediction for Processors with Frequency Scaling. *The Journal of Supercomputing*, 70(3):1451–1476, 2014. DOI: 10.1007/s11227-014-1236-4.

91. L. Renganarayana, U. Bondhugula, S. Derisavi, A. E. Eichenberger, and K. O'Brien. Compact multi-dimensional kernel extraction for register tiling. In *Proc. of the Conf. on High Performance Computing Networking, Storage and Analysis*, page 45. ACM, 2009.

92. S. Rivoire, M. Shah, P. Ranganathan, and C. Kozyrakis. JouleSort: A Balanced Energy-efficiency Benchmark. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 365–376. ACM, 2007. DOI: 10.1145/1247480.1247522.

93. S. Roy, A. Rudra, and A. Verma. Energy Aware Algorithmic Engineering. In *Proceedings of the 22nd Int. Symp. on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, MASCOTS '14, pages 321–330. IEEE Computer Society, 2014. DOI: 10.1109/MASCOTS.2014.47.

94. N. Satish, C. Kim, J. Chhugani, and P. Dubey. Large-scale Energy-efficient Graph Traversal: A Path to Efficient Data-intensive Supercomputing. In *Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 14:1–14:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

95. S.S. Shenoy and R. Eeratta. Green software development model: An approach towards sustainable software development. In *2011 Annual IEEE India Conference (INDICON)*, pages 1–6, Dec 2011. DOI: 10.1109/INDCON.2011.6139638.

96. Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai. Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-cell VLSI Designs. *ACM Trans. Des. Autom. Electron. Syst.*, 15(4):28:1–28:37, October 2010. DOI: 10.1145/1835420.1835421.

97. J. Shinde and S.S. Salankar. Clock gating A power optimizing technique for VLSI circuits. In *Proc. of the 2011 Annual IEEE India Conference (INDICON)*, pages 1–4, Dec 2011. DOI: 10.1109/INDCON.2011.6139440.

98. K. Singh, M. Bhadauria, and S. McKee. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News*, 37(2):46–55, July 2009. DOI: 10.1145/1577129.1577137.

99. C. U. Smith. *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

100. C.U. Smith and L.G. Williams. *Performance Solutions: a Practical Guide to Creating Responsive, Scalable Software*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2002.

101. SPEC Consortium. Standard Performance Evaluation Corp (SPEC), www.spec.org, 2015.

102. L. Tan, S. Kothapalli, L. Chen, O. Hussaini, R. Bissiri, and Z. Chen. A survey of power and energy efficient techniques for high performance numerical linear algebra operations. *Parallel Computing*, 40:559–573, 2014.

103. K.M. Tarplee, R. Friese, A.A. Maciejewski, and H.J. Siegel. Efficient and scalable computation of the energy and makespan Pareto front for heterogeneous computing systems. In *Proc. of the Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 401–408, Sept 2013.

104. The Green Grid Consortium. The Green Grid Website: http://www.thegreengrid.org/, 2014.

105. E. Thereska, A. Donnelly, and D. Narayanan. Sierra: practical power-proportionality for data center storage. In *EuroSys '11: Proc. of the 6th Conference on Computer systems*. ACM Request Permissions, April 2011.

106. M. Thiry, L. Frez, and A. Zoucas. GreenRM: Reference Model for Sustainable Software Development. In *Proc. of the 26th International Conference on Software Engineering and Knowledge Engineering*, pages 39–42, 2014.

107. A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely. Auto-tuning for Energy Usage in Scientific Applications. In *Proc. of the 2011 Int. Conf. on Parallel Processing - Volume 2*, Euro-Par'11, pages 178–187, Berlin, Heidelberg, 2012. Springer-Verlag. DOI: 10.1007/978-3-642-29740-3_21.

108. M.E. Tolentino and K.W. Cameron. The Optimist, the Pessimist, and the Global Race to Exascale in 20 Megawatts. *IEEE Computer*, 26(4):95–97, 2012.

109. TPC Consortium. Transaction Processing Performance Council (TPC), www.tpc.org, 2015.

110. U.S. Environmental Protection Agency. The ENERGYSTAR Website: http://www.energystar.gov/, 2015.

111. J. v. Kistowski, H. Block, J. Beckett, K. Lange, J. Arnold, and S. Kounev. Analysis of the Influences on Server Power Consumption and Energy Efficiency for CPU-Intensive

Workloads. In *Proc. of the 6th ACM/SPEC Int. Conf. on Performance Engineering*, ICPE '15, pages 223–234, New York, NY, USA, 2015. ACM. DOI: 10.1145/2668930.2688057.

112. A. Venkatesh, K. Kandalla, and D. Panda. Evaluation of Energy Characteristics of MPI Communication Primitives with RAPL. In *Proc. of the International Workshop on High Performance Power-Aware Computing at IPDPS*. IEEE, 2013.

113. R. Vuduc, J. Demmel, and K. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. In *Proc. SciDAC, J. Physics: Conf. Ser.*, volume 16, pages 521–530, 2005. DOI: 10.1088/1742-6596/16/1/071.

114. R.W. Vuduc. Autotuning. In *Encyclopedia of Parallel Computing*, pages 102–105. 2011.

115. Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kołodziej, Rajiv Ranjan, Cheng zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Generation Computer Systems*, 29(7):1661 – 1670, 2013.

116. R.C. Whaley and J.J. Dongarra. Automatically Tuned Linear Algebra Software. In *Proc. of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–27, Washington, DC, USA, 1998. IEEE Computer Society.

117. M. Wilde, M. Hategan, J.M. Wozniak, B. Clifford, D.S. Katz, and I. Foster. Swift: A Language for Distributed Parallel Scripting. *Parallel Computing*, 37(9):633–652, September 2011. DOI: 10.1016/j.parco.2011.05.005.

118. S. Williams, K. Datta, J. Carter, L. Oliker, J. Shalf, K. Yelick, and D Bailey. PERI - auto-tuning memory-intensive kernels for multicore. *Journal of Physics Conference Series*, 125(1), July 2008.

119. J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, and I.T. Foster. Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. In *Proc. of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 95–102, May 2013. DOI: 10.1109/CCGrid.2013.99.

120. D. Yamada, Sonobe T., H. Tezuka, and M. Inaba. Grid Spider: A framework for Data-Intensive research with Data Process Memorization Cache. In *Proc. of the 4th Int. Confererence on Resource Intensive Applications and Services. INTENSIVE 2012*, pages 5–8, 2012.

121. K. Yotov, X. Li, G. Ren, M.J. Garzarán, D. Padua, K. Pingali, and P. Stodghill. Is search really necessary to generate high-performance BLAS? *Proceedings of the IEEE*, 93(2):358–386, 2005.

122. Ziliang Z., A. Manzanares, B. Stinar, and Xiao Q. Energy-Aware Duplication Strategies for Scheduling Precedence-Constrained Parallel Tasks on Clusters. In *Proc. of the 2006 IEEE Int. Conf. on Cluster Computing*, pages 1–8, Sept 2006. DOI: 10.1109/CLUSTR.2006.311860.

123. I. Zecena, Ziliang Zong, Rong Ge, Tongdan Jin, Zizhong Chen, and Meikang Qiu. Energy consumption analysis of parallel sorting algorithms running on multicore systems. In *Proc. of the 2012 International Green Computing Conference (IGCC)*, pages 1–6. IEEE, June 2012. DOI: 10.1109/IGCC.2012.6322290.

124. S. Zheng, P. Zhang, and Zhang Q. A Routing Protocol Based on Energy Aware in Ad Hoc Networks. *Information Technology Journal*, 9(4):797–803, 2010. DOI: 10.3923/itj.2010.797.803.

*Received February 27, 2015.*