Parallel Computing 40 (2014) 559–573

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Review

A survey of power and energy efficient techniques for high performance numerical linear algebra operations

Li Tan, Shashank Kothapalli, Longxiang Chen, Omar Hussaini, Ryan Bissiri, Zizhong Chen [*]

Department of Computer Science and Engineering, University of California, Riverside, United States

A R T I C L E I N F O

Article history:
Received 12 December 2013
Received in revised form 2 August 2014
Accepted 4 September 2014
Available online 16 September 2014

Keywords:
Power and energy
Performance
Power management
Supercomputers
Numerical linear algebra
DVFS

A B S T R A C T

Extreme scale supercomputers available before the end of this decade are expected to have 100 million to 1 billion computing cores. The power and energy efficiency issue has become one of the primary concerns of extreme scale high performance scientific computing. This paper surveys the research on saving power and energy for numerical linear algebra algorithms in high performance scientific computing on supercomputers around the world. We first stress the significance of numerical linear algebra algorithms in high performance scientific computing nowadays, followed by a background introduction on widely used numerical linear algebra algorithms and software libraries and benchmarks. We summarize commonly deployed power management techniques for reducing power and energy consumption in high performance computing systems by presenting power and energy models and two fundamental types of power management techniques: static and dynamic. Further, we review the research on saving power and energy for high performance numerical linear algebra algorithms from four aspects: profiling, trading off performance, static saving, and dynamic saving, and summarize state-of-the-art techniques for achieving power and energy efficiency in each category individually. Finally, we discuss potential directions of future work and summarize the paper.

Contents

[*] Corresponding author.
E-mail address: chen@cs.ucr.edu (Z. Chen).

http://dx.doi.org/10.1016/j.parco.2014.09.001
0167-8191/© 2014 Elsevier B.V. All rights reserved.

1. Introduction

In this era of pervasive high performance computing, supercomputers are of ever-growing computation capability and network bandwidth. Given the rapidly climbing power bills, achieving power and energy efficiency of supercomputers has become a prime concern and also been considered as a challenging issue. As a fundamental integrant of high performance computing, numerical linear algebra algorithms including Cholesky, LU, and QR factorizations, serve as backbone for most scientific algorithms. Generally numerical linear algebra algorithms have been widely employed for solving a system of linear equations in high performance scientific applications running on supercomputers around the world, ranked by the TOP500 list [1]. To name a few, for the purpose of benchmarking, HPL [2] is a portable high performance software package that solves a dense linear system via LU factorization on distributed-memory architectures; NPB [3] is a set of benchmarks for evaluating performance of supercomputers, where highly parallel implementations of numerical linear algebra algorithms such as conjugate gradient method and LU factorization are included. For the purpose of scientific computing, ScaLAPACK [4] and DPLASMA [5] are two extensively used high performance and scalable numerical linear algebra software libraries for distributed-memory multicore systems, where routines of Cholesky, LU, and QR factorizations are provided as standard functionality. Moreover, with regard to software products, MATLAB [6] is a commercialized computing software developed by MathWorks for performing numerical calculations, where matrix factorizations are implemented in terms of easy-to-use user commands. MKL [7] is a commercialized software library developed by Intel for optimized linear algebra routines for scientific computing, including highly tuned routines of matrix factorizations, sparse solvers, and fast Fourier transforms. The open source libraries include ATLAS [8], HPL [2], etc, which are extensively used for solving and benchmarking. Numerical linear algebra algorithms are also generally adopted in many other areas of high performance scientific computing, including computer graphics, quantum mechanics, game theory, and economics.

With the growing severity of power and energy consumption on high performance computing systems nowadays in terms of operating costs and system reliability [9,10], in particular given the fact that power supply for large-scale data centers and clusters are usually limited within a power cap, reducing power and energy costs has been deemed as a critical issue in high performance computing. The Green500 list [11], ranks the top 500 supercomputers around the world by energy efficiency in a six-month cycle, which indicates the trend that supercomputers with a high ratio of performance-power (FLOPS per Watt) are favored nowadays. Motivated by the pressing and ever-growing demands of power and energy issues nowadays, people have proposed numerous solutions to maximize the use of the deployed power capacity of data centers [12,13]. As general purpose fundamental operations required for solving a system of linear equations, numerical linear algebra algorithms are extensively adopted in a large body of high performance scientific applications for different purposes as presented above. Consequently attempts of decreasing power and energy costs of running numerical linear algebra algorithms is beneficial to lowering energy consumption of executions of the applications that employ these algorithms ultimately.

Various holistic hardware and software solutions have been proposed for mitigating power and energy costs of running such applications on supercomputers around the world, and substantial power and energy savings have been achieved for different types of architectures. In this paper, we survey common power management strategies for high performance computing systems, with a focus on state-of-the-art techniques for achieving power and energy efficiency of high performance scientific applications where numerical linear algebra algorithms are widely employed.

The remainder of the paper is organized as follows. Section 2 introduces high performance numerical linear algebra and related software libraries and benchmarks. We present commonly used power management strategies for saving power and energy for high performance computing systems in general in Section 3, and provide details of state-of-the-art techniques for power and energy efficient numerical linear algebra in Section 4. Section 5 summarizes the paper.

2. Background: high performance numerical linear algebra

Next we provide a brief overview on widely employed numerical linear algebra algorithms, and present relevant high performance open source software libraries and benchmarks that provide routines of these algorithms.

2.1. Major dense numerical linear algebra algorithms

In dense numerical linear algebra, when solving a system of linear equations $Ax = b$, we generally first factorize the matrix A into a product of matrices of certain patterns, and then x can be easily solved from the factorized matrices via forward substitution and back substitution (sometimes the inverse of a matrix is computed). Different matrix factorizations are suitable for specific problem classes. Next we introduce three major matrix factorization algorithms Cholesky, LU, and QR factorizations [14] that are widely employed in extensive areas of high performance scientific computing, individually.

2.1.1. Cholesky factorization

$$\begin{pmatrix} A_{11} & A_{21}^T & A_{31}^T \\ A_{21} & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \times \begin{pmatrix} L_{11} & L_{21}^T & L_{31}^T \\ 0 & L_{22}^T & L_{32}^T \\ 0 & 0 & L_{33}^T \end{pmatrix} = \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T & L_{11}L_{31}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T & L_{21}L_{31}^T + L_{22}L_{32}^T \\ L_{31}L_{11}^T & L_{31}L_{21}^T + L_{32}L_{22}^T & L_{31}L_{31}^T + L_{32}L_{32}^T + L_{33}L_{33}^T \end{pmatrix} \quad (1)$$

Given the system of linear equations $Ax = b$, Cholesky factorization can be applied if A is a symmetric positive definite matrix (consequently A is a square $N \times N$ matrix). The goal of Cholesky factorization is to factorize A into the form LL^T where L is lower triangular and L^T is the transpose of L, and thus solve x from $LL^Tx = b$ via forward substitution and back substitution. Eq. (1) illustrates a 3×3 blocked Cholesky factorization in matrix representation, where a block denotes a submatrix partitioned from the global matrix A using a specific block size. The computation time complexity of Cholesky factorization on an $N \times N$ global matrix A is $O\left(\frac{1}{3}N^3\right)$.

2.1.2. LU factorization

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = P\begin{pmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{pmatrix} \times \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} \quad (2)$$

Instead of decomposing the global matrix A into the product of a lower triangular matrix L and its transpose L^T, LU factorization factorizes $M \times N$ matrix A into a product of an $M \times N$ unit lower triangular matrix L (diagonal elements are all 1s, i.e., $L_{ii} = 1$, where $0 \leqslant i \leqslant \min(M, N)$ and an $N \times N$ upper triangular matrix U, and thus the system of linear equations is transformed into $LUx = b$, where x can be easily solved similarly as in the case of Cholesky factorization. The differences between LU factorization and Cholesky factorization are as follows: (a) LU factorization applies to the case that A is any general $M \times N$ matrix; (b) besides the change of system form from LL^T to LU, LU factorization sometimes introduces a permutation matrix P to ensure numerical stability and leads to the ultimate form of the system $PLUx = b$. Eq. (2) shows the matrix representation of a 3×3 blocked LU factorization with the product matrix omitted due to the similarity as Cholesky factorization. The computation time complexity of LU factorization on an $N \times N$ global matrix A is $O(\frac{2}{3}N^3)$.

For performance purposes, matrix factorizations can be implemented for parallel execution on distributed-memory architectures as follows: (a) Partition a global matrix into a cluster of computing nodes as a process grid using load balancing techniques; (b) perform local diagonal matrix factorizations in each computing node individually and communicate factorized local matrices to the other computing nodes for panel matrix solving and trailing matrix updating, as shown in Fig. 1, a stepwise LU factorization without pivoting. A well-designed partitioning and highly-efficient parallel algorithms of computation and communication substantially determine performance and energy efficiency of distributed matrix factorization algorithms.

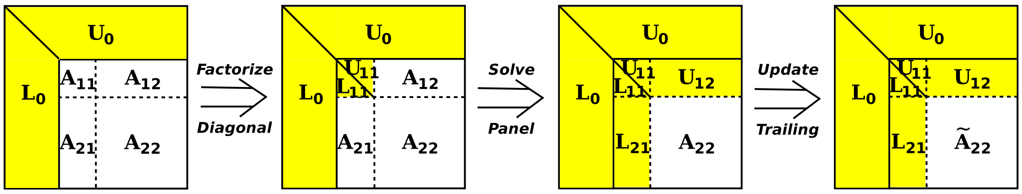

Fig. 1. Stepwise illustration of LU factorization without pivoting.

2.1.3. QR factorization

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{21} & Q_{22} & Q_{23} \\ Q_{31} & Q_{32} & Q_{33} \end{pmatrix} \times \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{pmatrix}, \text{ where } \begin{pmatrix} Q_{11} & Q_{12} & Q_{13} \\ Q_{21} & Q_{22} & Q_{23} \\ Q_{31} & Q_{32} & Q_{33} \end{pmatrix} \times \begin{pmatrix} Q_{11}^T & Q_{12}^T & Q_{13}^T \\ Q_{21}^T & Q_{22}^T & Q_{23}^T \\ Q_{31}^T & Q_{32}^T & Q_{33}^T \end{pmatrix} = I \tag{3}$$

As LU factorization, QR factorization also applies to a general $M \times N$ global matrix A, factorizing it into a product of an $M \times M$ orthogonal matrix Q and an $M \times N$ upper triangular matrix R, and thus the system of linear equations becomes $QRx = b$, where x can be easily solved similarly. Compared to LU factorization without pivoting, QR factorization without any pivoting (e.g., the modified Gram-Schmidt algorithm) is numerically more stable. Eq. (3) shows the matrix representation of a 3×3 blocked QR factorization with the product matrix omitted due to the space limitation, where $Q \times Q^T = I$ and I is an $M \times M$ unit matrix with ones on the diagonal and zeros elsewhere. The computation time complexity of QR factorization on an $N \times N$ global matrix A is $O\left(\frac{4}{3}N^3\right)$.

2.2. Algorithmic characteristics of Cholesky, LU, and QR factorizations

Although targeting different problem classes of linear algebra equation systems, Cholesky, LU, and QR factorizations hold similar algorithmic characteristics that can be utilized by various power and energy efficient techniques. As shown in Fig. 1, all algorithms can be characterized into three primary steps: Factorizing diagonal matrices, solving panel matrices, and updating trailing matrices. In the blocked versions of all algorithms, solving panel matrices and updating trailing matrices amount to the most of the execution time, where in general solving is memory-bound while updating is CPU-bound [14]. The boundness feature can thus be leveraged for devising specific power and energy saving strategies to save energy accordingly, without significant performance impact.

2.3. Major sparse numerical linear algebra algorithms

Different from dense linear algebra, sparse linear algebra operates on sparse matrices, i.e., in the linear equation system $Ax = b$, matrix elements of A are primarily zeros. The fraction of zero elements in a sparse matrix is referred to as the sparsity. It can thus be taken advantage of for achieving computation and memory efficiency using compressed data structures, where only the non-zero values and their indices are stored. The sparsity of the coefficient matrix A determines one fundamental difference between the two types of linear algebra that dense linear algebra is in general compute-bound, while sparse linear algebra is usually memory-bound. With regarding to power and energy efficiency, this key difference in workload characteristics can be leveraged when designing power and energy saving techniques for the two types of linear algebra, with different focuses individually.

2.3.1. Direct methods

We can still use matrix factorization algorithms above to solve a sparse linear equation system, within a finite and fixed number of steps. However, factorizing sparse matrices can generate enormous fill-in, i.e., matrix elements that change from an initial zero to a non-zero value during the execution of the factorization algorithm. Consequently, the introduction of such fill-in results in high computational costs and memory footprint, which degrades the performance of direct methods greatly. In some matrix factorization algorithm such as Cholesky factorization, fill-in can be minimized using ordering methods like permutation $B = PAP^T$.

2.3.2. Iterative methods

In contrast to direct methods, iterative methods are usually preferred in solving sparse linear equation systems to reduce the fill-in. Typical iterative methods include conjugate gradient method, generalized minimal residual method, and biconjugate gradient method, which utilize fast computation of sparse matrix–vector multiplication dominating the distinct iteration steps.

2.4. High performance open source software libraries and benchmarks of numerical linear algebra algorithms

The presence of various high performance open source software libraries and benchmarks of numerical linear algebra algorithms enables the use of hardware and software based power and energy efficient techniques in high performance scientific computing nowadays. We next introduce several major numerical linear algebra software libraries and benchmarks of our concern.

2.4.1. ATLAS

ATLAS [8] is a software library of linear algebra kernels that automatically tunes performance according to configuration of the hardware where it is deployed. Specifically, ATLAS produces a BLAS [15] library that has been highly optimized for the platform where ATLAS is installed. The goal of ATLAS is to provide portable and optimal performance of linear algebra routines across arbitrary cache-based architectures. Supported operations of ATLAS include a complete BLAS APIs and a subset of LAPACK [16] APIs.

2.4.2. HPL and NPB

Both HPL [2] and NPB [3] are highly parallel benchmarks targeting performance evaluation of supercomputers where numerical linear algebra algorithms are extensively applied. As introduced in Section 1, HPL is a portable and scalable implementation of dense linear system solver on distributed-memory computing systems. The algorithm employed by HPL is essentially a highly optimized right-looking variant of the LU factorization algorithm with row partial pivoting featuring multiple look-ahead depths. The global matrix elements are distributed into a two-dimensional $P \times Q$ process grid via the block cyclic data partitioning scheme (also known as, 2-D block cyclic data distribution) to guarantee good load balance and scalability.

As a collection of parallel and distributed algorithmic scientific methods, numerical linear algebra algorithms such as calculating eigenvalues and eigenvectors, fast Fourier transforming, solving systems of linear equations and partial differential equations are involved in 8 out of 11 benchmarks included in the latest NPB. NPB benchmarks are parallelized using MPI, OpenMP, and High Performance Fortran with additional optimizations, and feature genericness, architecture neutrality, and scalability. Specifically, performance efficiency of these benchmarks is highly dependent on optimizations based on hardware, OS, and compilers. Therefore, NPB has been deemed a viable solution of benchmarking performance of modern computing architectures.

2.4.3. ScaLAPACK and DPLASMA

ScaLAPACK [4] and DPLASMA [5] are two leading implementations of high performance numerical linear algebra algorithms for distributed-memory multicore systems. Both libraries feature 2-D block cyclic data distribution, block-partitioned algorithms, and high scalability/portability. DPLASMA is also devised for distributed heterogeneous systems with multiple sockets of multicore processors and accelerators such as NVIDIA Tesla GPU, AMD FirePro GPU, and Intel Xeon Phi coprocessor. The compatible interface of numerical linear algebra routines is provided by both implementations. Empirically compared to ScaLAPACK, DPLASMA is able to achieve higher performance (GFLOPS and weak scalability) and energy efficiency on large-scale distributed-memory architectures.

2.4.4. MUMPS and PETSc

MUMPS [17] is a massively parallel sparse direct solver for large linear systems with symmetric positive definite matrices and general matrices. It features parallel factorizing/solving phases, and is implemented atop ScaLAPACK. Moreover, it employs dynamic distributed scheduling to accommodate numerical fill-in and multi-user environment. PETSc [18] is a scalable and parallel sparse iterative solver for scientific applications modeled by partial differential equations. It focuses around Krylov subspace methods and allow easy and lightweight switching between several linear system solvers, with supports of distributed-memory, shared-memory, and heterogeneous CPU-GPU architectures, as well as any hybrid architectures. Moreover, it allows advanced users to have detailed control over the solution process.

3. Power management for high performance computing systems

The goal of power management during executions of high performance applications is to reduce power and energy consumption of the computing system limited within a power cap, with negligible performance loss [9]. In general, saving energy can be achieved by either reducing the average power consumed, or the execution time of the applications. In this paper, we primarily present power and energy saving techniques with constrained performance impact that are beneficial to decreasing both power and energy costs. A great number of hardware and software based power and energy saving solutions have been proposed for different components of the system. First of all, we model the power and energy costs of a high performance computing system, and next introduce commonly deployed power management techniques for reducing power and energy consumption of specific components in such systems.

3.1. Power and energy models

In general, power and energy consumption of a computing system where high performance applications are running can be formally modeled using the notation listed in Table 1. Within a given time interval (t_1, t_2) when a high performance application runs, the total energy consumption E_{sys} of a distributed-memory computing system consisting of multiple computing nodes can be formulated as below, where we denote the execution time as $T = t_2 - t_1$ and the nodal average power consumption as $\overline{P}_{node}$:

$$E_{sys} = \sum_{1}^{\#nodes} E_{node} = \sum_{1}^{\#nodes} \int_{t_1}^{t_2} P_{node} dt = \sum_{1}^{\#nodes} \overline{P}_{node} \times T \tag{4}$$

Table 1
Notation in power and energy consumption formalization.

E_{sys}	The total energy consumption of the whole cluster
E_{node}	The total energy consumption of all components in a node
P_{node}	The total power consumption of all components in a node
P_{CPU}	The total power consumption of CPU in any states
P_{CPU_d}	CPU dynamic power consumption in the busy state
P_{CPU_s}	CPU static/leakage power consumption in any states
P_{GPU}	The total power consumption of GPU in any states
P_{mem}	The total power consumption of memory in any states
P_{other}	Power consumption of components other than CPU/GPU/memory
P_{cap}	Actual available maximal value of system power consumption
A	Percentage of active gates in the CMOS-based chip
C	The total capacitive load in the CMOS-based chip
f	Current CPU working frequency
V	Current CPU supply voltage

Assuming each node in the computing system has the same hardware configuration and local energy efficiency results in global energy efficiency according to Eq. (4), we only consider nodal energy consumption (disregarding network devices connecting computing nodes and public network) and generally further break down nodal power consumption as (assume that GPU is present in the system):

$$P_{node} = P_{CPU} + P_{GPU} + P_{mem} + P_{other} \tag{5}$$

$$P_{CPU} = P_{CPU\_d} + P_{CPU\_s}, \text{ where } P_{CPU\_d} = ACfV^2 \tag{6}$$

In Eq. (5), we categorize the nodal power consumption by power consumption of CPU, GPU, memory, and other components, where power consumption of other components such as network, motherboard, disk, and fans amount to a small proportion of the total system power [19], and employing DVFS on these components is limited at the time of writing. In Eq. (6), by substituting P_{CPU_d}, we obtain the ultimate CPU power consumption formula with scalable parameters f and V (we discuss techniques that scale f and V in Sections 4 and 5) as:

$$P_{CPU} = ACfV^2 + P_{CPU\_s} \tag{7}$$

GPU power model [20] and memory power model [21] are similar to CPU power model: All of CPU, GPU, and memory power models consist of static power and dynamic power of the component, so GPU and memory power models are not elaborated for simplicity. Moreover, despite the difference on difficulty of implementation, employing DVFS on CPU, GPU, and memory essentially conforms to the same mechanism. Thus we only focus on CPU DVFS to show how to save power and energy using DVFS in this paper.

Regardless of types of workloads in the application, P_{CPU_s} and P_{other} are not affected by CPU frequency and voltage and empirically do not vary much during executions of the application, and thus can be regarded as constants for simplicity ($P_{CPU\_s} + P_{other}$ can be denoted as P_c). Consider a GPU compute intensive or memory intensive application. In this case, the peak CPU performance is not necessary due to the GPU-/memory-boundness. In other words, the nodal power consumption greatly depends on f and V, since P_{CPU} can be scaled down by reducing CPU frequency and voltage without incurring significant performance loss per the application characteristics. Following the constraints of Eqs. (4)–(7), we can optimize power costs of the computing system during executions of a high performance application, and eventually save energy by leveraging different power management techniques, including the ones that scale down f and V when necessary per application characteristics. With the constraint that performance of the application should be barely compromised, the ultimate goal of achieving power and energy efficiency in high performance computing can be formalized in general as follows:

$$\begin{aligned} \text{minimize } E_{sys} &= \sum_{1}^{\#nodes} \overline{P}_{node} \times T \\ \text{subject to } P_{sys} &\leqslant P_{cap} \\ \text{where } P_{sys} &= \sum_{1}^{\#nodes} P_{node} \end{aligned} \tag{8}$$

Empirically, due to limitations of power budget (often for large-scale data centers), available power supply for a computing system is normally restricted by a cap smaller than the theoretical peak value restricted by the hardware. Specifically, a system power cap is a definitive limit of power consumption the system can actually obtain and usually does not exceed, and the cap has no effect on performance until the system reaches its power consumption limit. The relationship between actual system power consumption and system power cap is reflected in Eq. (8). Generally, the goal is to minimize $\overline{P}_{node}$ and thus $\overline{P}_{sys}$ under the limitation of P_{cap} with as little performance degradation as possible. In other words, we need to minimize the trade-off between execution time and average power in terms of energy-performance efficiency quantitative metrics such as Energy-Delay Product (EDP), and consequently the optimal system energy efficiency can be fulfilled. We next present details of commonly employed power management techniques for achieving power and energy efficiency onsupercomputers nowadays.

3.2. Classic power management techniques

Generally, existing power management techniques for achieving power and energy efficiency in high performance computing systems can be categorized into two types, determined by the stage where the techniques are performed: Static and dynamic. Both types of techniques aim to reduce power and energy consumption of scientific applications running on high performance computing systems with negligible performance degradation. We present the details of the two types of power management techniques as below.

3.2.1. Static power management

Static power management techniques are launched prior to running an application. Typical static power management techniques include employing low power hardware components and enabling hyper-threading for logical cores ahead of the application runs.

(1) *Low Power Components* (LPC)
Recent research indicates for different workload intensive high performance applications (e.g., CPU-bound, network-bound, memory-bound, and disk-bound), power and energy consumption on CPU dominate system power and energy costs (35%~48%), and the second most power-/energy-consuming hardware component is memory (16%~27%) [9,22,19], more specifically, DRAM. Consequently, leveraging LPC such as low power CPU and memory can improve power and energy efficiency of the systems effectively.

(2) *Hyper-Threading* (HT)
As an implementation of simultaneous multithreading [23], the HT technique launched by Intel [24] helps to utilize more hardware resources by allowing two threads to run on each physical core. Empirically, for communication intensive applications, the HT technology is able to save energy effectively, since for such applications, the HT technique enables that two threads on one physical core can achieve similar performance as one thread each on two physical cores. Consequently, performance efficiency from using the HT technique results in energy efficiency.

3.2.2. Dynamic power management

In contrast to the static techniques, dynamic power management techniques are performed on the fly when an application runs. In general, dynamic techniques include all strategies that perform runtime monitoring and scheduling of system status and resources for power and energy efficiency. There exist two common dynamic power management mechanisms: Dynamic speed scaling and dynamic resource sleeping.

(1) *Dynamic Speed Scaling* (DSS)
As is implied by the name, the DSS technique exploits software and hardware based power-scalable components to dynamically adjust power consumption of computing systems. Dynamic Voltage and Frequency Scaling (DVFS), a popular example of the DSS technique, is able to dynamically modify the performance of a power-scalable component by altering its operating voltage and working frequency, which also adjusts power and energy costs of the component, given the fact that energy consumption equals product of average power consumption and execution time, and the assumption that dynamic power consumption P by a CMOS-based processor is proportional to product of frequency f and square of supply voltage V, i.e., $P \propto fV^2$ [25,26]. Typical components where the DVFS techniques have been applied include CPU, GPU, and memory. As discussed above, CPU and memory are the two most power and energy consuming components among all hardware components of a computing system. Moreover, power and energy costs on GPU also take a great proportion of the total power and energy costs of a heterogeneous computing system. Nowadays, various easy-to-use DVFS APIs/tools have been industrialized and even incorporated into the Linux kernel, such as CPUFreq kernel infrastructure [27] for CPU, and NVIDIA Management Library (NVML) [28] and NVIDIA System Management Interface (nvidia-smi) [29] for NVIDIA GPU. Therefore, employing DVFS is deemed to be an effective dynamic approach to achieve power and energy efficiency for high performance computing nowadays.
Generally, DVFS can be employed to reduce frequency and voltage of power-scalable components such as CPU, GPU, and memory, when current running operations are not CPU-bound, GPU-bound, or memory-bound respectively. Specifically, frequency and voltage of the components are lowered down when the peak performance of the components is not necessary for achieving the optimal performance of an application, while are kept at the highest scale otherwise. Power and energy savings can thus be achieved due to lower average frequency and voltage during executions of the application with negligible performance loss. Note that on many architectures such as GPU, the core frequency and the memory frequency are coupled and have to be switched simultaneously as a combination [28].
Due to the difference between memory accesses and CPU/GPU computation, memory DVFS is rather difficult empirically compared to CPU/GPU DVFS [21]. Cho et al. [30] observed that memory is generally non-supply-voltage-scalable, but its energy consumption is variable to its clock frequency. They derived the energy-optimal memory frequency as a

function of the number of CPU clock cycles, the number of memory clock cycles, and the number of memory accesses, and energy efficiency was achieved by different frequency assignment schemes. Two nearly concurrent efforts fulfilled memory DVFS from different aspects [31,21]. MemScale [31] applied two low-power modes: voltage and frequency scaling to the memory controller and only frequency scaling to the memory channels and DRAM devices, and guided by an OS policy that determines which mode to choose. The experimental evaluation is based on simulations due to hardware limitations. The approach proposed in [21] was however evaluated on a real hardware platform, where memory voltage and frequency were scaled via observing memory bandwidth utilization, with minimized performance loss. Specifically, frequency scaling was fulfilled by emulating memory frequency with altered timing settings, and voltage was scaled together with frequency proportionally.

Adaptive Voltage Scaling (AVS) is another runtime technique that monitors performance variability of the chip together with system characterization to influence voltage and frequency on the fly in a closed-loop configuration. Different from DVFS that is utilized in open-loop systems where frequency-voltage pairs are stored in a look-up table with built-in margin to cover temperature and process variations, AVS is employed to automatically adjust supply voltage to the minimum necessary level to meet performance requirements, usually for FPGA and VLSI systems. Different parameters are involved when selecting between the closed-loop and open-loop configurations for voltage scaling.

(2) *Dynamic Resource Sleeping* (DRS)

DRS dynamically hibernates components from the active running state C_0 into power-saving states for power and energy efficiency, and wakes them up on demand. Possible power-saving states include the active idle state I, different levels of sleep states denoted as $C_1, C_2, \ldots, C_n$, and the power-off state O. Generally, all sleep states consume less power than the active idle state. Power consumption of different sleep states depends on the level of sleeping, i.e., the deeper the component sleeps, the less power is dissipated, but the more time is needed for waking it up [32]. The power-off state consumes zero power. Note that different components may have different power states, and state transition also incurs time and energy costs inevitably.

4. Power and energy efficient high performance numerical linear algebra

Numerical linear algebra algorithms are widely applied in high performance scientific computing. Next we summarize different types of power management techniques for saving power and energy, with a focus on numerical linear algebra algorithms. We first present some efforts on power and energy profiling for such applications, which serves as the cornerstone of achieving power and energy efficiency for them, followed by a detailed review of the state-of-the-art techniques for saving power and energy of high performance numerical linear algebra algorithms from three different perspectives: trading off performance, static saving, and dynamic saving.

4.1. Power and energy profiling for high performance numerical linear algebra

Power and energy measurement are of primary concerns for high performance computing systems nowadays and have been widely studied [33,19,34], since it guides hardware and software based techniques for mitigating the operating costs and system reliability issues of modern high performance computing architectures. There exists a large body of work conducted on power and energy measurement using power sensors and meters equipped on different components of a system [19,35–43], and these efforts are essentially towards power and energy measurement in general, not for high performance numerical linear algebra in particular.

Several efforts have leveraged a handful of hardware counters supported by new microprocessor designs and directly used for power and energy measurement. Bui et al. [33] proposed to use on-chip counters to model power consumption of parallel scientific applications modeled by partial differential equations running on multicore and multiprocessor systems. Specifically, the power of different components of a processor is computed by exploiting the published thermal design power and the idle power, and being weighted via the access rates for different cache levels and core logic, where the access rates can be determined using the on-chip counters. The authors also presented a component-based framework for automated performance and power measurement and analysis. The work however suffers from two disadvantages: (a) The LU factorization performed by the multigrid solution method is sequential at coarse grid level, resulting in a severe scalability issue, and (b) the power consumption reported is only for processors of a node. Other significant power consuming components such as memory, disk, and motherboard are not studied. Hardware counter-based solutions were also explored by Demmel et al. [34], where on-chip energy counters provided by Intel's Running Average Power Limit (RAPL) interfaces [44] were leveraged for profiling energy costs of dense and sparse linear algebra routines for various problem sizes and core frequencies. Since 2011, Intel Sandy Bridge and later microarchitecture for CPU have emerged with on-chip counters allowing software based measurement for CPU and memory energy (specifically, DRAM) through the RAPL interfaces. The easy-to-use capability of collecting energy data via RAPL is however limited by two factors: (a) The energy data is of socket scope, and individual core energy profiling is current not supported, and (b) the ability of setting power limits for individual core is not available for use.

Dongarra et al. [45] studied energy consumption measurement using two component-based energy monitoring frameworks PowerPack and Intel RAPL for high performance dense linear algebra libraries LAPACK [16] and PLASMA [46], respectively. Power profiling of block and tile algorithms of LU factorization were reported using the two power and energy measurement frameworks for comparison purposes. The close match between physical power measurement using PowerPack and RAPL observed in the experimental results indicates that RAPL offers a viable alternative to physical power meters for the tested algorithms. In addition to employing on-chip energy counters as an alternative of physical power sensors and meters, given the fact that there exist no clear OS interfaces for accurately exposing resource power consumption to user-level runtimes, Kestor et al. [47] devised an interface between OS and user runtime to measure accurate per-core power costs via a proxy power sensor based on a regression analysis of core activities. Instead of obtaining power information from a real power sensor, the proposed interface allows user-level software based power measurement. A power profiling runtime library was implemented for analyzing NPB benchmarks and the analysis shows the feasibility of the proxy power sensor technique for comprehensively understanding the power characteristics of high performance scientific applications.

A considerable amount of work has been conducted to analytically predict power and energy consumption via prior knowledge on power usage data. Cabrera et al. [48] presented an analytical model for predicting power consumption for HPL, where architectural and algorithmic parameters associated for power and energy models were obtained from real power consumption data, and these parameters can be utilized to estimate HPL power and energy costs across a range of architectures effectively with reasonable errors. This work however suffers from two drawbacks: (a) At larger problem sizes, disk swapping becomes a factor to affect power and energy costs a lot, but the current approach does not take this into account; (b) at smaller problem sizes, execution time of HPL can be even smaller than the minimal sample time interval of the power meter used for obtaining necessary power data, which causes considerable errors between the measured and predicted data. Subramaniam et al. [49] proposed to utilize multi-variable regression analysis to model performance and energy efficiency of HPL. The coefficients of the multi-variable regression techniques have a close relationship with the 18 parameters used for HPL performance tuning, and can be determined by the least square method. The statistical regression model was able to predict the HPL configuration (i.e., the most efficient HPL parameters) for achieving the maximal energy efficiency with high accuracy. The disadvantage of this work is that the power values were extrapolated from a single node of a cluster and the overhead between nodes were not taken into consideration. Tiwari et al. [50] demonstrated that artificial neural networks can be employed to predict the component level power draw and energy usage of HPC kernels such as matrix multiplication and LU factorization, where a subset of measured power data were used as a training input to an artificial neural network, and predictions were made for power draw rate and energy consumption of different system components with acceptable absolute error rate. One problem of this approach is that the training data was gathered from a multicore architecture with specific hardware configurations, which means the model needs to be rebuilt for a cluster of nodes with different architectures, and the model does not necessarily scale well on a distributed-memory architecture with more processors.

Two typical research efforts have been contributed to comparing power and energy efficiency of state-of-the-art implementations of high performance numerical linear algebra algorithms on multicore and distributed-memory systems. For shared-memory multicore architectures, Ltaief et al. [51] conducted experiments on power profiling the commonly used routines of numerical linear algebra algorithms in LAPACK and PLASMA for identifying different phases of computation and thus isolating bottlenecks for energy efficiency. Experimental results show that PLASMA using fine-grained task-parallel tile algorithms is superior to LAPACK using fork-join block algorithms, in terms of both performance and energy efficiency. For distributed-memory multicore architectures, Bosilca et al. [52] entended the power profiling to high performance numerical linear algebra libraries such as ScaLAPACK and DPLASMA, and in particular presented performance and power data for Cholesky and QR factorizations. Reported data shows that DPLASMA using fine-grained task parallelism with dynamic distributed scheduling and tiled data layout is more performance and power efficient compared to ScaLAPACK using multithreaded block algorithms and 2-D block cyclic data distribution, on a power-aware cluster with 2592 cores.

4.2. Optimizing power and performance efficiency trade-off for high performance numerical linear algebra

Performance and power/energy are generally two critical, correlated, while mutually constrained factors in high performance computing, indicated by the formula $E = \overline{P} \times T$ where $\overline{P}$ is the average power consumption during runs of high performance applications. The fulfillment of energy efficiency in some cases is at the cost of performance degradation within a reasonable range. For instance, as presented in Section 3.2.1, employing low power components may achieve better performance-energy ratio in terms of minor performance loss and major energy efficiency, due to constraints that hardware is reaching its physical limits empirically [53,54].

There also has been extensive software based work conducted on trading increased execution time for energy savings. Tan et al. [55] proposed a DVFS scheduling strategy to achieve energy efficiency for data intensive applications. The approach aggressively applies DVFS to mixed types of workloads holistically, and adaptively sets an appropriate CPU frequency to the hybrid workloads according to the percentage of CPU-bound computation within the total execution time. Additional energy savings is fulfilled via speculation to mitigate DVFS overhead for imbalanced branches. Significant energy savings were achieved at the cost of minor performance loss experimentally towards several memory and disk access intensive benchmarks with imbalanced branches compared to another two energy saving solutions on a power-aware 64-core cluster. Subsequent work [56] studied the effect of performance gain in distributed matrix multiplication on energy efficiency via a high performance pipeline broadcast as the communication scheme. Benner et al. [57,58] investigated the performance and power/energy trade-off for two low power architectures, compared to a general purpose multicore processor. Experimental results indicate that for dense linear algebra operations where matrix–matrix multiplication is widely employed, the race-to-halt strategy (i.e., execute the application at the highest available CPU frequency during computation and switch to the lowest available CPU frequency during other tasks) tends to attain both high throughput and high performance-per-watt ratio on general purpose architectures, while a hybrid architecture that combines low power architectures and GPU can offer competitive performance with higher energy efficiency.

From a system point of view, performance and power/energy trade-off has been thoroughly studied. Song et al. [59] built a system-level iso-energy-efficiency model to analyze, evaluate, and predict energy-performance of data intensive applications running on large scale power-aware clusters where various execution patterns were exploited. Effects of machine and application dependent characteristics (e.g., processor count, CPU power/frequency, workload size, and network bandwidth) to balance energy use and performance were studied extensively. The model was experimentally evaluated on two power-aware clusters with different scales, and the total system energy costs can be predicted within negligible errors for applications with different execution patterns. Subsequent work [60] extended the iso-energy-efficiency model to determine appropriate parameters for performance and power/energy scaling of numerical linear algebra applications running on scalable systems, and proposed correlation functions to quantitatively explain effects of parameters involved in maintaining performance and power/energy efficiency as system size scales. Barreda et al. [61] and Aliaga et al. [62] provided a detailed experimental analysis on the balance between power consumption and computational performance for LU factorization, Cholesky factorization, reduction to tridiagonal form, and sparse linear system solvers. Ge et al. [9] studied three types of distributed performance-directed DVFS scheduling strategies for achieving performance-constrained energy efficiency in scalable power-aware HPC clusters, and showed empirically that the proposed techniques consistently achieve energy efficiency with little impact on execution time. They also discussed automatically selection of distributed DVFS scheduling meeting users' requests via the metric of energy-delay products.

Numerous work has been conducted from the perspective of hardware. Pedram et al. [63,54] introduced a custom microarchitecture design for a Linear Algebra Core (LAC), in particular for matrix computations, and extended the LAC design with a fine-tuning general memory hierarchy model to evaluate trade-offs in performance and energy efficiency in terms of energy per operation. The prototype linear algebra processor empirically shows the implementation of matrix multiplication can achieve high performance while consuming less energy compared to the cutting-edge commercial architectures. The proposed design is however not generalized to handle more matrix operations other than matrix multiplication due to lack of demonstrated applicability experimentally. Jang et al. [64] profiled state-of-the-art FPGA designs to identify energy hot spots where most of energy dissipation is attributed, and developed new algorithms and architectures concerning trade-offs among available hardware parameters. The impact of system-wide energy dissipation, area, and latency is represented by optimization functions. Extensive low-level simulations were conducted and the proposed designs was demonstrated to improve energy efficiency of FPGA without any increase in the area-latency product. In terms of comprehensive metrics such as energy-area-time, the shown designs also outperform the state-of-the-art techniques considerably. Prasanna et al. [65–67] presented two block based parallel designs for LU factorization on FPGA using a linear array architecture, which can minimize the usage of long interconnects for higher throughput and thus lead to lower energy dissipation. Choi et al. [68,69] evaluated both performance and energy efficiency of FPGAs, embedded processors, and DSPs in matrix multiplication, and the experimental results show that FPGAs can perform matrix multiplication with much lower latency while also consuming much less power than the other two devices. The disadvantage of this work is that it only considers single signal processing application, i.e., matrix multiplication, instead of various types of applications to show the genericness of FPGAs in high performance energy efficient scientific computing. Garcia et al. [70] proposed instruction-/task-level optimizations and an energy-aware tiling to reduce static and dynamic energy for LU factorizations on many-core architectures, without sacrificing scalability or performance.

4.3. Static Power and Energy Saving for High Performance Numerical Linear Algebra

One straightforward strategy of saving power and energy for high performance scientific applications is to deploy Low Power Components (LPC) in the computing systems. Essentially as a hardware solution, this strategy mitigates power and energy consumption regardless of application characteristics, and thus should save power and energy for any applications running on the computing systems where LPC modules are deployed.

Various LPC modules are employed in high performance architectures for power and energy efficiency nowadays. IBM Blue Gene/L, Blue Gene/P, and Blue Gene/Q [53] are the most well-known and used to be the most power efficient supercomputers around the world according to the June 2012 Green500 list [11]. IBM Blue Gene systems are characteristic by two primary features of power and energy efficiency: (a) Trading performance for low power and energy consumption, i.e., by exploiting relatively low power CPU and other LPCs, the systems are able to achieve better performance-energy ratio, and (b) leveraging the system-on-a-chip design, which saves power and energy due to low power supply to fewer number of chips since the design aggregates more components to be embedded on one chip.

The latest three versions of the Green500 list unveil the top spots of the most power efficient supercomputers have been taken over by heterogeneous high performance computing systems that combine classic computational processors with graphics processors or coprocessors. In other words, the architectures consisting of a combination of CPU with GPU such as NVIDIA Tesla K20 or coprocessors such as Intel Xeon Phi prevail in power and energy efficiency [11]. Eurora and Aurora [71], the currently top two most power and energy efficient supercomputers all over the world, are heterogeneous systems based on NVIDIA Kepler K20 GPU accelerators, which improves performance-power ratio (MFLOPS/W) by 40%, compared to the previous greenest supercomputer around the world, IBM Blue Gene/Q. Experimental results show that the combination of CPU, GPU, and other types of accelerators can balance the trade-offs between performance and energy and yield the best performance-power ratio, since strengths of different processors are exploited in executing high performance applications nowadays with different types of workloads.

There exists a large body of work on leveraging LPC for performance and power efficient data-driven operations. Caulfield et al. [72] presented a low power architecture for data intensive applications that combines lower power processors, flash memory, and data-centric programming systems to achieve both performance and power efficiency. Andersen et al. [73] introduced a new low power architecture for data-intensive computing that couples low power embedded CPUs to local flash storage, and balances computation and I/O to provide power efficient parallel data accesses. Szalay et al. [74] proposed a novel architecture consisting of a large number of energy efficient CPUs with solid state disks to increase sequential read I/O throughput while keeping power consumption constant. Beckmann et al. [75] exploited a low power processor, solid state disks together with energy efficient sorting algorithms to improve data access performance per energy unit using well-established sorting benchmarks. A general conclusion is drawn from the above work that a balance between low power costs and performance efficiency should be carefully maintained to optimize the performance-power ratio for data intensive applications. Potential efforts could be made to extend the proposed solutions to energy efficient computing in general in addition to data-centric applications via utilizing more low power components.

With regard to multicore architectures, employing simultaneous multithreading is able to improve both performance and energy efficiency for high performance applications due to thorough utilization of additional instruction-level parallelism from multiple threads [23]. Intel's implementation of simultaneous multithreading, namely Hyper-Threading (HT) Technology [24], allows two threads to run on each physical core as two virtual/logical cores to improve parallelism of computation, taking advantage of superscalar architectures. Demmel et al. [34] experimentally observed the HT Technology can save energy for communication intensive applications (e.g., the dgemv() routine from numerical linear algebra libraries), while can incur extra energy costs for applications where computation dominates (e.g., the dgemm() routine from numerical linear algebra libraries), since applying the HT Technology in compute intensive applications results in threads on the same core competing for computation power [76] and thus performance degradation, while in the case of communication intensive applications, the shared core can be utilized thoroughly via the HT technology. Fig. 2 shows performance and energy efficiency of the dgemv() routine from Intel MKL 10.3.

4.4. Dynamic power and energy saving for high performance numerical linear algebra

At the OS, library, or application level, power and energy savings can be achieved with negligible performance loss for high performance numerical linear algebra algorithms. Next we summarize the state-of-the-art dynamic techniques for power/energy efficient numerical linear algebra. Ge et al. [9] categorized three types of distributed DVFS scheduling strategies that are viable for conserving energy while maintaining performance: a) Using a daemon process as system-driven external control, (b) scheduling from the command-line as user-driven external control, and (c) scheduling within the application via power-aware component APIs as user-driven internal control, where strategy (b) is essentially a static approach since the user sets CPU frequency for each node prior to executing the application. We focus on strategies (a) and (c) in the later review.

The primary difference between strategy (a) and strategy (c) lies in the genericness. Since strategy (a) utilizes a standby process that is essentially a daemon independent on the processes manipulated by the application itself, similar to the black-box testing in software engineering, strategy (a) applies to a general application regardless of prior knowledge of internal structures of the application. Strategy (c) exploits the algorithmic characteristics of the application and typically requires modification of the source code of libraries the application extensively invokes, and thus understanding of algorithmic details of the application is necessary. Previous review involves several efforts on strategy (a) [9,77]. Zhu et al. [78] proposed a consolidation scheme of scientific workflow tasks for energy and resource cost optimization, and developed a power-aware consolidation framework named pSciMapper consisting of online consolidation and offline analysis components. The primary drawback of this approach is the considerable slowdown around 15% for the application, which is unacceptable for

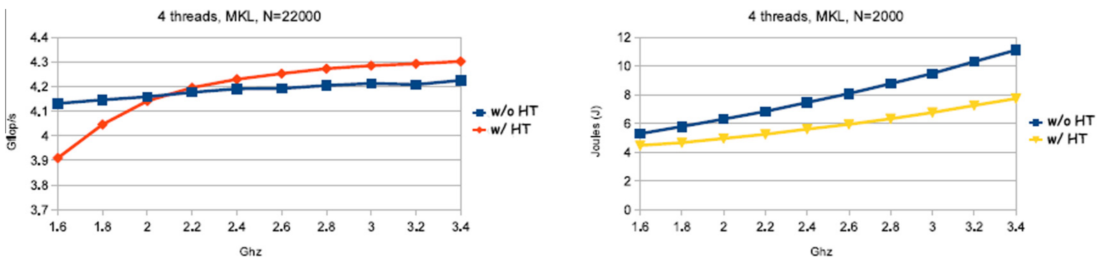

Fig. 2. Effectiveness of hyper-threading on performance and energy efficiency [34].

Table 2
Summary of power/energy efficient techniques for high performance numerical linear algebra algorithms.

Ref.	Category	Technique	Power/Energy savings	Applicable targets
[55]	Trading-off	Aggressive and adaptive DVFS scheduling on mixed types of workloads with speculation	32.6% avg. energy ↓ w/6.2% avg. perf. ↓	Data intensive scientific applications on clusters
[56]	Trading-off	Improving communication perf. for energy efficiency as a whole	7.5% avg. perf. ↑ w/6.5% avg. energy ↓	Matrix multiplication on distributed-mem. sys.
[57]/[58]	Trading-off	Analyzing perf./energy balance on heterogeneous systems w/low power architectures	Heterogeneous systems tend to offer higher perf./energy ratio	Matrix inversion via Gauss–Jordan elim. on multicore architectures
[59]/[60]	Trading-off	Analyzing execution patterns to predict energy costs and balance perf./energy ratio	Within 5% avg. error of total system energy consumption prediction	Data intensive scientific app. on scalable clusters
[63]/[54]	Trading-off	Devising a microarchitecture for linear algebra routines w/fine-tuned memory hierarchy	600 GFLOPS w/25 W and 50× energy efficient than cutting-edge CPU	Matrix multiplication on the prototype linear algebra processor
[64]	Trading-off	Identifying energy hot spots for optimizing perf./energy	40% avg. perf./energy ↑ w/o area-time product ↑	Matrix multiplication in VHDL on FPGA
[65]/[66]/[67]	Trading-off	Block based designing via a linear array architecture for high throughput/low energy	35% avg. energy ↓ w/50% avg. perf. ↑	LU factorization in VHDL on FPGA
[68]/[69]	Trading-off	Evaluating perf. and energy efficiency of FPGA, embedded processors, and DSP	FPGA multiplies matrices w/lower latency and lower energy than others	Matrix multiplication on FPGA, embedded processors, and DSP
[70]	Trading-off	Designing a power-aware tiling to ↓ static and dynamic energy	3.28× avg. power/perf. ratio ↑ using 156 threads	LU factorization on many-core architectures
[53]	Static	Exploiting low power components and aggregating more components on one chip	BlueGene/L scales up to 65536 dual-core nodes & 360 TFLOPS perk perf.	Generic purposes
[72]	Static	Integrating low power processors, flash memory, and data-centric programming systems	1.5× perf. ↑ and 2.5× perf. per Watt ↑ than disk-based clusters	Data intensive scientific applications on clusters
[73]	Static	Coupling lower power embedded CPU to flash storage, and balancing computation & I/O	350 queries per Joule & two orders of magnitude ↑ than disk-based sys.	Data intensive scientific applications on clusters
[74]	Static	Combining low power CPU w/solid state disk to increase I/O throughput and reduce power	I/O throughput ↑ one order of magnitude/5× w/power/cost constant	Data intensive scientific applications on clusters
[75]	Static	Employing low power processor solid state disk & efficient algo.	5.1× perf./energy ratio (records per Joule) ↑	Data intensive scientific app. on multicore sys.
[34]	Static	Utilizing simultaneous multithreading (Hyper-Threading)	up to 29.5% energy ↓ w/up to 1.9% perf. ↑	Commun. intensive sci. app. on multicore sys.
[77]	Dynamic	Utilizing power-aware mapping to meet deadline while reducing power by DVFS on CPU + GPU	w/Deadline met, more than 20% energy ↓	Time sensitive scientific app. on heterogeneous systems w/CPU + GPU
[78]	Dynamic	Use power-aware consolidation via a distance metric and resource correlation analysis	up to 56% energy ↓ w/less than 15% perf. ↓	Scientific applications on distributed virtualized architectures
[79]/[80]	Dynamic	Leveraging Monte-Carlo simulation to adaptively balance workload based on hardware status	44× perf. ↑ and 19.6× energy ↓ w/27% perf./energy ratio ↑ overall	Financial applications on clusters w/FPGA, CPU, and GPU
[81]/[82]/[83]	Dynamic	Reducing frequency for tasks off the critical path to lower power w/o performance loss	up to 2.5% perf. ↓ w/up to 5% energy ↓ on shared-memory systems	Cholesky, LU, and QR factorizations on simulated multicore processors
[84]	Dynamic	Analyzing Linux DVFS governors on energy saving modes	Different governors impact perf. & energy balance	Three matrix routines on distributed-mem. sys.
[85]	Dynamic	Utilize algorithmic characteristics of linear algebra operations to predict slack accurately	33.8% avg. energy ↓ w/3.9% avg. perf. ↓	Maximize potential energy savings via algorithmic slack prediction
[86]	Dynamic	Adaptively trading more comp. & comm. w/mem. for DVFS ↓	7.5% extra avg. energy ↓ than a classic strategy	Matrix multiplication on distributed-mem. sys.
[87]/[88]	Dynamic	Scaling frequency for tasks not located on the critical path	15% avg. energy ↓ w/o observable perf. loss	Sparse matrix app. on distributed-mem. sys.
[89]	Dynamic	Comparing energy saving impact of DVFS on GPU + CPU	GPU frequency ↑ leads to perf. ↑ w/small energy ↑	Matrix multiplication on hetero. CPU + GPU
[90]	Dynamic	Eliminating polling when core is idle or running	up to 9% energy ↓ w/up to 4% perf. ↓	Cholesky and LU factorizations on CPU + GPU
[91]/[92]	Dynamic	Setting system to a low consuming state when GPU runs	up to 25.8% energy ↓ w/noticeable comp. perf. ↓	Iterative linear solvers on hetero. CPU + GPU
[20]	Dynamic	Predicting at runtime the opt. num. of active cores for higher perf./energy ratio	up to 22.09% energy ↓ on GPU w/9.18% avg. error	Matrix kernel operations on GPU architectures
[93]/[94]/[95]	Dynamic	Monitoring runtime performance variability to scale power using adaptive voltage scaling	up to 85% energy ↓ w/nominal voltage operation at the same frequency	Fast Fourier transform on FPGA architectures

high performance computing nowadays. Luk et al. [79,80] proposed a novel Monte-Carlo simulation framework that supports multiple types of hardware accelerators (FPGA and GPU) and provides scheduling interfaces to adaptively perform load balancing for performance and energy efficiency. Energy savings achieved is from performance gain obtained from the collaborative simulation framework, but not from an energy saving strategy itself.

One effective and widely studied approach that can be implemented in terms of both strategies (a) and (c) is the Critical Path (CP) approach, where energy saving opportunities can be exploited by appropriately reducing operating frequency of computation components to dilate tasks off the CP into their slack and thus achieving lower average power consumption without incurring performance loss. Based on the CP approach, several algorithms such as Slack Reduction algorithm and Race-to-Idle algorithm [81–83] have been proposed to save energy for dense linear algebra algorithms on shared-memory multicore processors. One great disadvantage is their work is either based on simulation (including the DVFS techniques) or only work for single multicore machine. Subsequent work [84] analyzed the impact of different energy saving modes via Linux governors on dense linear algebra kernels in a distributed-memory architecture, but no algorithmic strategies via energy efficient DVFS scheduling within the application were proposed. Tan et al. [85] proposed to utilize algorithmic characteristics of distributed dense matrix factorizations for obtaining slack accurately and thus saving more energy, with negligible performance loss and trading off partial generality. Subsequent work [86] presented an adaptive memory-aware strategy for improving energy efficiency of distributed matrix multiplication by trading more computation and communication at a time with memory cost for less DVFS switches. Chen et al. [87,88] proposed a DVFS energy reduction scheme for tree based parallel sparse applications that exploits load imbalance across parallel processors and applies DVFS to processors off the critical path without sacrificing performance of the application. However, their approach was based on simulation atop theoretical assumption and no empirical studies were conducted for real-world applications running on real machines.

Considerable amount of dynamic power/energy efficient approaches have been devised for heterogeneous systems that comprised of multiple types of computing components. Liu et al. [77] proposed several power-aware mapping techniques for a CPU-GPU heterogeneous system that reduced power and energy consumption by applying DVFS on both CPU and GPU, and timing requirements of applications were met. More than 20% power and energy savings were achieved towards several matrix workloads. Ge et al. [89] experimentally studied the impacts of DVFS on performance and energy efficiency for GPU computing. Compared to CPU computing, they observed that for compute intensive high performance matrix workloads running on GPU, performance and power consumption are approximately proportional to GPU frequency, and thus increasing GPU frequency may lead to better performance and energy efficiency. Alonso et al. [90] investigated the trade-off between execution time and energy costs of task-parallel Cholesky and LU factorizations on a hybrid CPU-GPU platform. Anzt et al. [91,92] performed analysis of energy consumption on GPU-accelerated iterative linear solvers to demonstrate energy savings can be fulfilled without harming performance on CPU-GPU architectures by setting the host system to a low power state for the time that GPU is executing via DVFS. Hong et al. [20] discussed the fact that due to high number of parallel processors, power consumption of GPU architectures has increased significantly. They proposed an integrated power and performance model to predict the optimal number of processors for a given application running on GPU, based on the intuition that using more cores is not necessary for applications that reach the peak memory bandwidth. Unlike other prediction models that utilize various static information, the proposed runtime prediction model takes advantage of the outcomes of prediction on performance and power costs and determine the optimal number of cores for the highest performance-power ratio. Potential future work may target other bottlenecks such as network bandwidth and disk access stalls.

Other efforts of saving power/energy dynamically have been conducted on FPGA [93–95] and VLSI [96,97] via Adaptive Voltage Scaling (AVS). Although results on scientific kernels such as Fast Fourier transform, fast motion estimation and convolution demonstrate that AVS serves as a great alternative of DVFS, where the device uses a number of pre-calculated valid working points like FPGA, there exist few studies dedicated on algorithmic power/energy efficient numerical linear algebra kernels on FPGA. The common algorithmic characteristics of numerical linear algebra routines may provide opportunities to utilize AVS for energy saving purposes on embedded systems like FPGA as a potential research direction.

5. Summary

High performance scientific computing nowadays requires extensive use of numerical linear algebra algorithms for solving systems of linear equations that models target problems formally and precisely. Decreasing power and energy consumption for numerical linear algebra algorithms is greatly beneficial to the overall power and energy efficiency of high performance scientific applications running on supercomputers around the world. Given the empirical trend that energy costs on computing components such as CPU and GPU dominate system energy consumption, employing techniques that appropriately assign voltage and frequency such as DVFS from the OS, library, or application level, according to system utilization status and algorithmic details of applications, can be a promising approach for performance and energy optimization to mitigate the power constraints of supercomputers nowadays. Table 2 summarizes state-of-the-art techniques for above purposes.

In this paper, we survey recent work conducted on power and energy efficient techniques for numerical linear algebra algorithms that serves as the cornerstone of achieving low power and energy consumption for high performance scientific computing nowadays. Power management for high performance computing systems is generic to all types of applications and thus need to be adopted as a cornerstone in fulfilling power and energy efficiency for high performance scientific applications. There exist a large body of research efforts on power and energy optimization via profiling, trading off performance, static saving, and dynamic saving respectively.

Regarding reducing power and energy costs for numerical linear algebra algorithms in particular, leveraging algorithmic details of these algorithms to devise dedicated power and energy saving strategies is potential to achieve extra optimization at the cost of partial loss of generality, since the effectiveness of algorithmic power and energy efficient approaches relies heavily on prior knowledge on internal structures of the application (e.g., algorithms and data structures). Applying such algorithm-based solutions at the library level would mitigate the disadvantage of partial loss of generality, due to extensive employment of these numerical linear algebra algorithms as software libraries. Moreover, runtime overhead of OS level solutions would also be eliminated by applying static analysis based on application-specific knowledge of these algorithms.

References

[1] TOP500 Supercomputer Lists. <http://www.top500.org/>.
[2] HPL – High-Performance Linpack Benchmark. <http://www.netlib.org/benchmark/hpl/>.
[3] NAS Parallel Benchmarks. <http://www.nas.nasa.gov/publications/npb.html>.
[4] ScaLAPACK – Scalable Linear Algebra PACKage. <http://www.netlib.org/scalapack/>.
[5] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemariner, H. Ltaief, P. Luszczek, A. YarKhan, J. Dongarra, Distributed dense numerical linear algebra algorithms on massively parallel architectures: DPLASMA, Tech. Rep. UT-CS-10-660, University of Tennessee Computer Science (Sep. 2010).
[6] MATLAB – The Language of Technical Computing <http://www.mathworks.com/products/matlab/>.
[7] Intel Math Kernel Library. <http://software.intel.com/en-us/intel-mkl/>.
[8] Automatically Tuned Linear Algebra Software (ATLAS). <http://math-atlas.sourceforge.net/>.
[9] R. Ge, X. Feng, K.W. Cameron, Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters, in: Proc. SC, 2005, pp. 34.
[10] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters, in: Proc. CCGRID, 2007, pp. 541–548.
[11] Green500 Supercomputer Lists. <http://www.green500.org/>.
[12] A. Faraj, X. Yuan, Automatic generation and tuning of MPI collective communication routines, in: Proc. ICS, 2005, pp. 393–402.
[13] X. Fan, W.-D. Weber, L.A. Barroso, Power provisioning for a warehouse-sized computer, in: Proc. ISCA, 2007, pp. 13–23.
[14] J. Choi, J.J. Dongarra, L.S. Ostrouchov, A.P. Petitet, D.W. Walker, R.C. Whaley, The design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines, Scientific Programming 5 (3) (1996) 173–184.
[15] BLAS (Basic Linear Algebra Subprograms). <http://www.netlib.org/blas/>.
[16] LAPACKG – Linear Algebra PACKage. <http://www.netlib.org/lapack/>.
[17] MUMPS: a MUltifrontal Massively Parallel sparse direct Solver. <http://mumps.enseeiht.fr/>.
[18] PETSc: a Portable, Extensible Toolkit for Scientific Computation. <http://www.mcs.anl.gov/petsc/>.
[19] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, K.W. Cameron, PowerPack: energy profiling and analysis of high-performance systems and applications, IEEE Trans. Parallel Distrib. Syst. 21 (5) (2010) 658–671.
[20] S. Hong, H. Kim, An integrated GPU power and performance model, in: Proc. ISCA, 2010, pp. 280–289.
[21] H. David, C. Fallin, E. Gorbatov, U.R. Hanebutte, O. Mutlu, Memory power mamagement via dynamic voltage/frequency scaling, in: Proc. ICAC, 2011, pp. 31–40.
[22] X. Feng, R. Ge, K.W. Cameron, Power and energy profiling of scientific applications on distributed systems, in: Proc. IPDPS, 2005, pp. 34.
[23] Y. Li, D. Brooks, Z. Hu, K. Skadron, P. Bose, Understanding the energy efficiency of simultaneous multithreading, in: Proc. ISLPED, 2004, pp. 44–49.
[24] Intel Hyper-Threading Technology. <http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-t>
[25] A. Miyoshi, C. Lefurgy, E.V. Hensbergen, R. Rajamony, R. Rajkumar, Critical power slope: Understanding the runtime effects of frequency scaling, in: Proc. ICS, 2002, pp. 35–44.
[26] C.-H. Hsu, W.-C. Feng, A power-aware run-time system for high-performance computing, in: Proc. SC, 2005, pp. 1.
[27] CPUFreq – CPU Frequency Scaling. <https://wiki.archlinux.org/index.php/CPUFrequencyScaling>.
[28] NVIDIA Management Library (NVML). <https://developer.nvidia.com/nvidia-management-library-nvml/>.
[29] NVIDIA System Management Interface (nvidia-smi). <https://developer.nvidia.com/nvidia-system-management-interface/>.
[30] Y. Cho, N. Chang, Memory-aware energy-optimal frequency assignment for dynamic supply voltage scaling, in: Proc. ISLPED, 2004, pp. 387–392.
[31] Q. Deng, D. Meisner, L. Ramos, T.F. Wenisch, R. Bianchini, Memscale: Active low-power modes for main memory, in: Proc. ASPLOS, 2011, pp. 225–238.
[32] Processors - What is difference between deep and deeper sleep states?, <http://www.intel.com/support/processors/sb/CS-028739.htm>.
[33] V. Bui, B. Norris, K. Huck, L.C. McInnes, L. Li, O. Hernandez, B. Chapman, A component infrastructure for performance and power modeling of parallel scientific applications, in: Proc. CBHPC, 2008, p. 6.
[34] J. Demmel, A. Gearhart, Instrumenting linear algebra energy consumption via on-chip energy counters, Tech. Rep. UCB/EECS-2012-168, University of California, Berkeley, Electrical Engineering and Computer Sciences, 2012.
[35] V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, S. Moore, Measuring energy and power with PAPI, in: Proc. PASA, 2012, pp. 262–268.
[36] Intel, Intel energy checker: Software developer kit user guide, 2010.
[37] P. Popa, Managing server energy consumption using IBM PowerExecutive, IBM Systems and Technology Group, Tech. Rep., 2006.
[38] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, N. Chang, Energy-monitoring tool for low-power embedded programs, IEEE Des. Test Comput. 19 (4) (2002) 7–17.
[39] S. Ryffel, LEA^{2}P: The linux energy attribution and accounting platform, Master's Thesis, Swiss Federal Institute of Technology, 2009.
[40] J. Flinn, M. Satyanarayanan, PowerScope: a tool for profiling the energy usage of mobile applications, in: Proc. MCSA, 1999, pp. 2–10.
[41] T. Stathopoulos, D. McIntire, W. Kaiser, The energy endoscope: real-time detailed energy accounting for wireless sensor nodes, in: Proc. IPSN, 2008, pp. 383–394.
[42] C. Isci, M. Martonosi, Runtime power monitoring in high-end processors: Methodology and empirical data, in: Proc. MICRO, 2003, pp. 93.
[43] F. Bellosa, The benefits of event: Driven energy accounting in power-sensitive systems, in: Proc. ACM SIGOPS European Workshop, 2000, pp. 37–42.
[44] H. David, E. Gorbatov, U.R. Hanebutte, R. Khanna, C. Le, Rapl: memory power estimation and capping, in: Proc. ISLPED, 2010, pp. 189–194.
[45] J. Dongarra, H. Ltaief, P. Luszczek, V.M. Weaver, Energy footprint of advanced dense numerical linear algebra using tile algorithms on multicore architecture, in: Proc. CGC, 2012, pp. 274–281.
[46] Parallel Linear Algebra for Scalable Multi-core Architectures (PLASMA). <http://icl.cs.utk.edu/plasma/>.
[47] G. Kestor, R. Gioiosa, D.J. Kerbyson, A. Hoisie, Enabling accurate power profiling of HPC applications on exascale systems, in: Proc. ROSS, 2013, pp. 4.
[48] A. Cabrera, F. Almeida, V. Blanco, D. Giménez, Analytical modeling of the power consumption for the high performance LINPACK, in: Proc. PDP, 2013, pp. 343–350.
[49] B. Subramaniam, W. Feng, Statistical power and performance modeling for optimizing the energy efficiency of scientific computing, in: Proc. GreenCom, 2010, pp. 139–146.
[50] A. Tiwari, M.A. Laurenzano, L. Carrington, A. Snavely, Modeling power and energy usage of HPC kernels, in: Proc. HPPAC, 2012, pp. 990–998.

[51] H. Ltaief, P. Luszczek, J. Dongarra, Profiling high performance dense linear algebra algorithms on multicore architectures for power and energy efficiency, in: Proc. EnA-HPC, 2011.
[52] G. Bosilca, H. Ltaief, J. Dongarra, Power profiling of Cholesky and QR factorizations on distributed memory systems, in: Proc. EnA-HPC, 2012.
[53] The BlueGene/L Team, An overview of the BlueGene/L supercomputer, in: Proc. SC, 2002, p. 60.
[54] A. Pedram, R.A. van de Geijn, A. Gerstlauer, Codesign tradeoffs for high-performance, low-power linear algebra architectures, IEEE Trans. Comput. 61 (12) (2012) 1724–1736.
[55] L. Tan, Z. Chen, Z. Zong, R. Ge, D. Li, A2E: Adaptively aggressive energy efficient DVFS scheduling for data intensive applications, in: Proc. IPCCC, 2013, pp. 1–10.
[56] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, D. Li, Improving performance and energy efficiency of matrix multiplication via pipeline broadcast, in: Proc. CLUSTER, 2013, pp. 1–5.
[57] P. Benner, P. Ezzatti, E.S. Quintana-Ortí, A. Remón, Trading off performance for power-energy in dense linear algebra operations, in: Proc. HPCLatAm, 2013, pp. 158–166.
[58] P. Benner, P. Ezzatti, E.S. Quintana-Ortí, A. Remón, On the impact of optimization on the time-power-energy balance of dense linear algebra factorizations, in: Proc. ADPC, 2013, pp. 3–10.
[59] S. Song, C.-Y. Su, R. Ge, A. Vishnu, K.W. Cameron, Iso-energy-efficiency: an approach to power-constrained parallel computation, in: Proc. IPDPS, 2011, pp. 138–139.
[60] S. Song, M. Grove, K.W. Cameron, An iso-energy-efficiency approach to scalable system power-performance optimization, in: Proc. CLUSTER, 2011, pp. 262–271.
[61] M. Barreda, M.F. Dolz, R. Mayo, E.S. Quintana-Ortí, R. Reyes, Binding performance and power of dense linear algebra operations, in: Proc. ISPA, 2012, pp. 63–70.
[62] J. Aliaga, M.F. Dolz, A.F. Martín, R. Mayo, E.S. Quintana-Ortí, Leveraging task-parallelism in energy-efficient ILU preconditioners, in: Proc. ICT-GLOW, 2012, pp. 55–63.
[63] A. Pedram, A. Gerstlauer, R.A. van de Geijn, A high-performance, low-power linear algebra core, in: Proc. ASAP, 2011, pp. 35–42.
[64] J.-W. Jang, S.B. Choi, V.K. Prasanna, Energy- and time-efficient matrix multiplication on FPGAs, IEEE Trans. Very Large Scale Integr. Syst. 13 (11) (2005) 1305–1319.
[65] S. Choi, V. Prasanna, Time and energy efficient matrix factorization using FPGA, in: Proc. FPL, 2003, pp. 507–519.
[66] G. Govindu, S. Choi, V. Prasanna, V. Daga, S. Gangadharpalli, V. Sridhar, A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs, in: Proc. IPDPS, 2004, pp. 149.
[67] V. Daga, G. Govindu, V. Prasanna, S. Gangadharpalli, V. Sridhar, Floating-point based block LU decomposition on FPGAs, in: Proc. ERSA, 2004, pp. 276–279.
[68] R. Scrofano, S. Choi, V.K. Prasanna, Energy efficiency of FPGAs and programmable processors for matrix multiplication, in: Proc. FPT, 2002, pp. 422–425.
[69] J.-W. Jang, S. Choi, V.K. Prasanna, Energy-efficient matrix multiplication on FPGAs, in: Proc. FPL, 2002, pp. 534–544.
[70] E. Garcia, J. Arteaga, R. Pavel, G.R. Gao, Optimizing the LU factorization for energy efficiency on a many-core architecture, in: Proc. LCPC, 2013, pp. 1–15.
[71] High Performance Computing at Eurotech. <http://www.eurotech.com/en/hpc/>.
[72] A.M. Caulfield, L.M. Grupp, S. Swanson, Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications, in: Proc. ASPLOS, 2009, pp. 217–228.
[73] D.G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, V. Vasudevan, FAWN: a fast array of wimpy nodes, in: Proc. SOSP, 2009, pp. 1–14.
[74] A.S. Szalay, G. Bell, H.H. Huang, A. Terzis, A. White, Low-power Amdahl-balanced blades for data intensive computing, in: Proc. HotPower, 2009, pp. 71–75.
[75] A. Beckmann, U. Meyer, P. Sanders, J. Singler, Energy-efficient sorting using solid state disks, in: Proc. IGCC, 2010, pp. 191–202.
[76] R. Schöne, D. Hackenberg, D. Molka, Simultaneous multithreading on x86_64 systems: an energy efficiency evaluation, in: Proc. HotPower, 2011, p. 10.
[77] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, X. Lin, Power-efficient time-sensitive mapping in heterogeneous systems, in: Proc. PACT, 2012, pp. 23–32.
[78] Q. Zhu, J. Zhu, G. Agrawal, Power-aware consolidation of scientific workflows in virtualized environments, in: Proc. SC, 2010, pp. 1–12.
[79] A.H.T. Tse, D.B. Thomas, K.H. Tsoi, W. Luk, Dynamic scheduling Monte-Carlo framework for multi-accelerator heterogeneous clusters, in: Proc. FPT, 2010, pp. 233–240.
[80] Q. Liu, W. Luk, Heterogeneous systems for energy efficient scientific computing, in: Proc. ARC, 2012, pp. 64–75.
[81] P. Alonso, M.F. Dolz, R. Mayo, E.S. Quintana-Ortí, Improving power efficiency of dense linear algebra algorithms on multi-core processors via slack control, in: Proc. HPCS, 2011, pp. 463–470.
[82] P. Alonso, M.F. Dolz, F.D. Igual, R. Mayo, E.S. Quintana-Ortí, DVFS-control techniques for dense linear algebra operations on multi-core processors, Comput. Sci. – Res. Dev. 27 (4) (2012) 289–298.
[83] P. Alonso, M.F. Dolz, F.D. Igual, R. Mayo, E.S. Quintana-Ortí, Saving energy in the LU factorization with partial pivoting on multi-core processors, in: Proc. PDP, 2012, pp. 353–358.
[84] M. Castillo, J.C. Fernández, R. Mayo, E.S. Quintana-Ortí, V. Roca, Analysis of strategies to save energy for message-passing dense linear algebra kernels, in: Proc. PDP, 2012, pp. 346–352.
[85] L. Tan, Z. Chen, Optimizing Energy Efficiency for Distributed Dense Matrix Factorizations via Utilizing Algorithmic Characteristics, in: Proc. IPDPS PhD Forum, 2014.
[86] L. Tan, L. Chen, Z. Chen, Z. Zong, R. Ge, D. Li, HP-DAEMON: high performance distributed adaptive energy-efficient matrix-multiplication, in: Proc. ICCS, 2014, pp. 599–613.
[87] G. Chen, K. Malkowski, M. Kandemir, P. Raghavan, Reducing power with performance constraints for parallel sparse applications, in: Proc. IPDPS, 2005, pp. 1–8.
[88] S.W. Son, K. Malkowski, G. Chen, M. Kandemir, P. Raghavan, Reducing energy consumption of parallel sparse matrix applications through integrated link/CPU voltage scaling, J. Supercomput. 41 (3) (2007) 179–213.
[89] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, Z. Zong, Effects of dynamic voltage and frequency scaling on a K20 GPU, in: Proc. PASA, 2013, pp. 826–833.
[90] P. Alonso, M.F. Dolz, F.D. Igual, R. Mayo, E.S. Quintana-Ortí, Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms, in: Proc. ISPA, 2012, pp. 56–62.
[91] H. Anzt, V. Heuveline, J. Aliaga, M. Castillo, J.C. Fernández, R. Mayo, E.S. Quintana-Ortí, Analysis and optimization of power consumption in the iterative solution of sparse linear systems on multi-core and many-core platforms, in: Proc. IGCC, 2011, pp. 1–6.
[92] H. Anzt, M. Castillo, J.C. Fernández, V. Heuveline, F.D. Igual, R. Mayo, E.S. Quintana-Ortí, Optimization of power consumption in the iterative solution of sparse linear systems on graphics processors, Comput. Sci. – Res. Dev. 27 (4) (2012) 299–307.
[93] S. Ishihara, Z. Xia, M. Hariyama, M. Kameyama, Architecture of a low-power FPGA based on self-adaptive voltage control, in: Proc. ISOCC, 2009, pp. 274–277.
[94] J. Nunez-Yanez, Energy proportional computing in commercial FPGAs with adaptive voltage scaling, in: Proc. FPGAworld, 2013, p. 6.
[95] J. Nunez-Yanez, Adaptive voltage scaling with in-situ detectors in commercial FPGAs, IEEE Trans. Comput., PrePrint (99), 2013, 1.
[96] S. Dhar, D. Maksimović, B. Kranzen, Closed-loop adaptive voltage scaling controller for standard-cell ASICs, in: Proc. ISLPED, 2002, pp. 103–107.
[97] M. Elgebaly, M. Sachdev, Variation-aware adaptive voltage scaling system, IEEE Trans. Very Large Scale Integr. Syst. 15 (5) (2007) 560–571.