Optimizing Energy Efficiency for Distributed Dense Matrix Factorizations via Utilizing Algorithmic Characteristics

Li Tan and Zizhong Chen
University of California, Riverside
$\{$*ltan003, chen*$\}$*@cs.ucr.edu*

***Abstract*—The pressing demands of improving energy efficiency for high performance scientific computing have motivated a large body of software-controlled hardware solutions that strategically switch hardware components to a low-power state, when the peak performance of the components is not necessary. Although OS level solutions can effectively save energy in a black-box fashion, for applications with random/variable execution patterns, slack prediction can be error-prone and thus the optimal energy efficiency can be blundered away. We propose to utilize algorithmic characteristics to predict slack accurately and thus maximize potential energy savings.**

***Keywords*-energy; critical path; algorithmic slack prediction.**

I. Introduction

With the growing prevalence of distributed-memory architectures, high performance scientific computing has been widely employed on supercomputers around the world ranked by the TOP500 list [1], Considering the crucial fact that the costs of powering a supercomputer is rapidly increasing nowadays due to expansion of its size and duration in use, improving energy efficiency of high performance scientific applications has been regarded as a pressing issue to solve. The Green500 list [2], ranks the top 500 supercomputers worldwide by performance-power ratio in six-month cycles. Consequently, root causes of high energy consumption while achieving performance efficiency in parallelism have been widely studied. With different focuses of studies, holistic hardware and software approaches for reducing energy costs of running high performance scientific applications have been extensively proposed. Software-controlled hardware solutions such as DVFS-directed (Dynamic Voltage and Frequency Scaling) energy efficient scheduling are deemed to be effective and lightweight.

DVFS is a runtime technique that is able to switch operating voltage and working frequency of a hardware component (CPU, GPU, memory, etc.) to different *scales* (also known as *gears*) per workload characteristics of applications to gain energy savings dynamically. CPU and GPU are the most widely applied hardware components for energy efficiency via DVFS. Energy saving opportunities can be exploited by reducing frequency and voltage of a specific component for operations not bound by the component. For instance, energy savings can be achieved if scaling down frequency/voltage of CPU during large-message communication on distributed-memory systems, since generally execution time of the communication barely increases at a low-power state of CPU, in contrast to that at a high-power state. According to the energy equation $E = \overline{P} \times T$ and the power relationship $P \propto fV^2$, lowering frequency and voltage without lengthening execution time can thus save energy effectively.

Running on distributed-memory architectures, high performance applications can be organized and scheduled in the unit of task, a set of operations that are functionally executed as a whole. As typical task-parallel algorithms for scientific computing, dense matrix factorizations in numerical linear algebra such as LU factorization have been widely adopted to solve systems of linear equations. Empirically, as standard functionality, routines of dense matrix factorizations are provided by various software libraries of numerical linear algebra for distributed-memory multicore architectures such as ScaLAPACK [3], DPLASMA [4], and MAGMA [5]. Therefore, saving energy for distributed dense matrix factorizations contributes significantly to the greenness of high performance scientific computing nowadays.

II. Overall Comparison: Energy Saving Methods

A. Critical Path and Slack Analysis

For task-parallel applications, slack generally refers to a time period when one hardware component waits for another due to imbalanced throughput and utilization. For instance, CPU usually waits for data from memory/disk in executions of memory/disk access intensive applications, since memory/disk access is performance bottleneck of the applications in accordance with the memory hierarchy. Typical examples of slack include load imbalance, network latency, communication delay, memory and disk access stalls, etc. Due to pervasive inequity of hardware throughput/utilization, slack becomes a significant source of achieving energy efficiency in high performance computing. During one execution of a task-parallel application, Critical Path (CP) is a particular sequence of tasks that starts from the beginning task of the execution until the ending task of it, where the total slack amounts to zero. Energy saving opportunities can thus be exploited for tasks not on the CP where slack can arise, in particular slack among non-communication tasks.

B. Energy Saving via Race-to-halt

As the name suggests, *race-to-halt* is a DVFS-directed energy saving strategy that enforces hardware components (e.g., CPU and GPU) to *race* when workloads are ready for processing, and to *halt* when no workloads are present and thus the components are idle. In other words, *race* refers to execute workloads at the highest frequency and voltage of the components until the finish of the workloads, and *halt* implies that frequency and voltage of the components are switched to the lowest scale from the end of the last workload to the start of the next one. This straightforward approach can effectively save energy without performance loss due to the following inferences: (a) The peak performance of the components is guaranteed as in the original execution without *race-to-halt*, and (b) the peak performance is not necessary when no workloads are being executed. Note that the *race-to-halt* strategy is *CP-free* such that no CP detection is required before any energy saving operations are performed. Thus it is lightweight and easy to implement.

C. Energy Saving via Slack Reclamation

Another critical strategy of saving energy is CP-based slack reclamation. Per the definition of CP, it is implied that any delay on tasks on the CP also delays the application as a whole, while appropriately dilating tasks off the CP into their slack individually without overflowing slack, does not increase the total execution time of the application. Energy savings can thus be achieved from scaling down frequency and voltage for dilating tasks off the CP into their slack without incurring performance loss. Since this approach is on top of detecting CP and energy efficient DVFS scheduling is determined among tasks on/off the CP, it is also referred to as the *CP-aware* approach for slack reclamation.

III. Optimizing Energy Efficiency via Utilizing Algorithmic Characteristics

Although measurable, slack needs to be known in advance of executing the task causing it for energy saving purposes, since the extent of scaling frequency/voltage must be calculated before applying DVFS to eliminate the slack. A dynamic accurate slack prediction algorithm is thus desirable.

A. OS-level Slack Prediction

At OS level, slack prediction essentially depends on a workload prediction mechanism: Execution characteristics in the upcoming interval can be predicted using prior execution information, e.g., execution traces in recent intervals. However, the workload prediction may not necessarily be reliable and lightweight: (a) For applications with variable execution patterns, such as dense matrix factorizations. Execution time of one iteration of the core loop shrinks as the remaining unfinished matrices become smaller. Dynamic prediction on execution characteristics such as task runtime and workload distribution can thus be inaccurate, leading to error-prone slack estimation; (b) for applications with random execution patterns, such as applications relying on random numbers that could lead to variation of control flow at runtime, which can be difficult to capture. Since the predictor needs to determine reproducible execution patterns at the beginning of one execution, it can be costly for obtaining an accurate prediction in both cases. Given the fact that no energy savings can be fulfilled until the prediction phase is finished, considerable potential energy savings may be wasted for a qualified but lengthy prediction as such at OS level.

B. Algorithmic Slack Prediction

We propose to utilize algorithmic characteristics of distributed dense matrix factorizations for predicting slack with high accuracy, where execution history of only the first core loop iteration is required. Consider LU factorization on a global matrix of size $N \times N$ with block size nb. Denote the measured execution time of the first instance of the task updating the *trailing matrix* $N' \times N'$ (computation time complexity is $O(N'^3)$) as T_0^U, slack of the rest instances of the task can be formalized as follows, where $0 \le i \le \frac{N}{nb}$:

$$\frac{T_{i+1}^U}{T_i^U} = \frac{O(N_{i+1}'^3)}{O(N_i'^3)} = \frac{(N - (i+2) \times N_{proc} \times nb)^3}{(N - (i+1) \times N_{proc} \times nb)^3}$$

$$slack_{i+1} = \mathtt{func}(T_i^U, T_{i+1}^U, param\_list)$$

We can solve slack of the rest instances of one type of tasks, given the execution time of the first iteration and computation time complexity of the tasks determined by algorithmic characteristics of LU factorization. Note that **`func()`** is a custom function that reflects relationship among tasks for calculating slack based on application-specific knowledge. The algorithmic slack prediction mechanism is reliable and lightweight compared to OS level slack prediction due to: (a) Using algorithmic information, we clearly know the type and amount of future workloads ahead of execution, regardless of random/variable execution patterns of the applications, and (b) prediction overhead is minimized since no training of the predictor based on prior execution information is necessary.

IV. Conclusions and Future Directions

DVFS-directed solutions have been widely adopted to improve energy efficiency for task-parallel applications. With high generality, OS level solutions are considered effective. We observe that for applications such as distributed dense matrix factorizations, the optimal energy efficiency cannot be achieved by OS level solutions due to inaccurate slack prediction. Giving up partial generality, we propose to utilize algorithmic characteristics for obtaining slack accurately and thus saving more energy, with negligible performance loss.

We plan to extend our work for typical distributed dense matrix factorizations including Cholesky/LU/QR factorization, on emerging large-scale power-aware architectures.

References

[1] *TOP500 Supercomputer Lists.* http://www.top500.org/.

[2] *Green500 Supercomputer Lists.* http://www.green500.org/.

[3] *ScaLAPACK - Scalable Linear Algebra PACKage.* http://www.netlib.org/scalapack/.

[4] *DPLASMA: Distributed Parallel Linear Algebra Software for Multicore Architectures.* http://icl.cs.utk.edu/dplasma/.

[5] *MAGMA (Matrix Algebra on GPU and Multicore Architectures).* http://icl.cs.utk.edu/magma/.