Slow Down or Halt: Saving the Optimal Energy for Scalable HPC Systems

Li Tan and Zizhong Chen
University of California, Riverside

Abstract

Slack is pervasive in runs of high performance applications, in the presence of various performance boosting solutions. The presence of slack provides ample opportunities for achieving energy efficiency for high performance computing nowadays. Regardless of communication slack, classic energy saving approaches for saving energy during the slack otherwise include *race-to-halt* and *CP-aware slack reclamation*, which reply on power scaling techniques to adjust processor power states judiciously during the slack. Existing efforts demonstrate *CP-aware slack reclamation* is superior to *race-to-halt* in energy saving capability. In this paper, we formally model our observation that the energy saving capability gap between the two approaches is significantly narrowed down on today's processors, given the fact that state-of-the-art CMOS technologies allow insignificant variation of supply voltage as operating frequency of a processor scales. We also provide experimental evaluation for validation on a large-scale power-aware cluster.

1 Introduction

Power and energy efficiency are now of great concern when the launching date of exascale computers is approaching. Power and energy consumption of a supercomputer nowadays have been rapidly increasing due to expansion of its size and duration in use. The US Department of Energy has set up a goal of 20 MW for the exascale computers targeted in the year around 2020 [2]. The advancement of hardware and software solutions have greatly improved power and energy efficiency of high performance computing, where the pervasive slack during runs of task-parallel applications is regarded as an important source for achieving power and energy savings, regardless of various performance boosting techniques (e.g., load balancing [4] and work stealing [5]) for decreasing the slack as much as possible.

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 to be ready from memory for memory intensive applications, in accordance with the fundamental memory hierarchy. Typical examples of slack include load imbalance, network latency, communication delay, memory and disk access stalls, etc. Energy saving opportunities can be exploited during the slack of runs of task-parallel applications, since the peak performance of hardware components that are not fully utilized during the slack is not necessary. Software-controlled hardware solutions such as Dynamic Voltage and Frequency Scaling (DVFS) techniques have been extensively leveraged to mitigate energy costs by appropriately scaling power states of the hardware without incurring performance loss for the applications [6] [11] [7] [10] [12].

Critical Path (CP) is one particular task trace from the beginning task of one execution of a task-parallel application to the ending one with the total slack of zero. Any delay on tasks on the CP increases the total execution time of the application, while *appropriately slowing down* the processors where the application is running by dilating tasks off the CP into their slack, or *halting* tasks off the CP during their slack individually without further delay, does not cause performance loss as a whole. Energy savings can be achieved effectively by both approaches with negligible performance loss.

In this paper, we discuss the energy saving capability of two classic energy saving approaches, and formally calculate and compare the energy savings from both approaches. Previous formal proof indicates that *CP-aware slack reclamation* always beat *race-to-halt* in terms of energy efficiency [8] [9]. We demonstrate that for DVFS on state-of-the-art architectures, supply voltage of a processor scales much less than its operating frequency, the energy saving gap between the two approaches is narrowed down significantly. We also provide preliminary experimental evaluation to validate our observations.

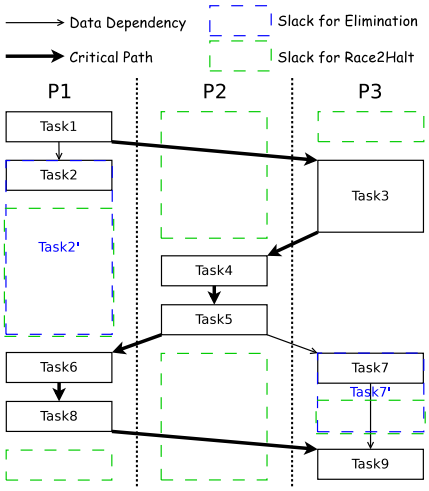

Figure 1: DAG Representation of Slack Handling of Two Energy Saving Approaches for a 3-Process Application.

Table 1: Notation in Energy Saving Capability Analysis.

E	Total nodal energy consumption of all components
P	Total nodal power consumption of all components
$P_{dynamic}$	Dynamic power consumption in the running state
$P_{leakage}$	Static/leakage power consumption in any states
T	Execution time of a task at CPU peak performance
T'	Slack of executing a task at CPU peak performance
A	Percentage of active gates in a CMOS-based chip
C	The total capacitive load in a CMOS-based chip
f	Current CPU working frequency
V	Current CPU supply voltage
V'	Supply voltage of components other than CPU
I_{sub}	CPU subthreshold leakage current
I'_{sub}	non-CPU component subthreshold leakage current
f_m	Available frequency assumed to eliminate T' without using frequency approximation
V_h	The highest supply voltage set by DVFS
V_l	The lowest supply voltage set by DVFS
V_m	Supply voltage corresponding to f_m set by DVFS
n	Ratio between execution time and slack of a task

2 Classic Energy Saving Strategies

Existing energy efficient approaches that save energy strategically during slack of running high performance applications can essentially be categorized into two types: Race-to-halt and CP-aware slack reclamation. Next we illustrate how they work in different ways.

2.1 Energy Saving for Communication

Slack from communication is an important source of energy savings. Consider a high performance application running on a distributed-memory system based on message passing, reduction of energy consumption can be achieved by reducing frequency and voltage of computing components such as CPU and GPU for large-message MPI communication, since generally execution time of such operations barely increases at a low-power state of the computing hardware during the communication slack. We adopt the low-power strategy for communication slack (also known as *scheduled communication* approach [11]). The next two classic energy saving approaches are intended in particular for slack arising from non-communication, i.e., mostly, computation.

2.2 Race-to-halt

As the name suggests, *race-to-halt* is a DVFS scheduling 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 available, as the area covered by green dashed boxes shown in Figure 1. Specifically, *race* refers to execute workloads with the maximum performance at the highest working frequency and voltage of the hardware, until the finish of the workloads, while *halt* means to slow down the hardware to the minimum working frequency and voltage, i.e., the lowest power state for energy saving purposes, from the end of the precedent workload to the start of the subsequent workload. This straightforward approach can effectively save energy without incurring performance loss.

2.3 CP-aware Slack Reclamation

Another critical strategy of saving energy during the slack is to reclaim slack by appropriately slowing down tasks that are not on the Critical Path (CP) of an execution trace of a HPC application. 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, as prolonged tasks in blue dashed boxes shown in Figure 1. Energy savings can thus be achieved from scaling down frequency/voltage for dilating tasks off the CP into their slack without performance degradation. This solution is based on CP detection. Energy efficient DVFS scheduling decisions for slack reclamation are determined among tasks on/off the CP.

3 Energy Saving Capability Analysis

Existing work demonstrates that under a time constraint, slowing down a processor can reduce energy consumption the most, compared to completing a task as fast as possible and completing a task using combination of discrete frequencies [8] [9]. However, the gap between energy saving capability of *race-to-halt* and *CP-aware*

Table 2: Frequency-Voltage Pairs for Different Processors (Unit: Frequency (GHz); Voltage (V)).

Gear	AMD Opteron 2380		AMD Opteron 846 and AMD Athlon64 3200+		AMD Opteron 2218		Intel Pentium M		Intel Pentium 4 HT 530		Intel Xeon E5 2687W		Intel Core i7-2760QM	
	Freq.	Volt.	Freq.	Volt.	Freq.	Volt.	Freq.	Volt.	Freq.	Volt.	Freq.	Volt.	Freq.	Volt.
0	2.5	1.300	2.0	1.500	2.4	1.250	1.4	1.484	3.0	1.430	3.1	1.200	2.4	1.060
1	1.8	1.200	1.8	1.400	2.2	1.200	1.2	1.436	N/A	N/A	N/A	N/A	2.0	0.970
2	1.3	1.100	1.6	1.300	1.8	1.150	1.0	1.308	N/A	N/A	N/A	N/A	1.6	0.890
3	0.8	1.025	0.8	0.900	1.0	1.100	0.8	1.180	2.1	1.250	1.2	0.840	0.8	0.760

slack reclamation shrinks, since state-of-the-art CMOS technologies allow insignificant variation of supply voltage as operating frequency of a processor scales. Next we formally prove that the two approaches can be comparable in energy saving capability.

Given the following two energy saving strategies, towards a task t with an execution time T and slack T' at the peak CPU performance, we calculate the total nodal system energy consumption for both strategies, i.e., $E(\mathsf{S}_1)$ and $E(\mathsf{S}_2)$ formally below:

- Strategy I (Race-to-halt): Execute t at the highest CPU frequency f_h until the finish of t and then switch to the lowest CPU frequency f_l, i.e., run in T at f_h and in T' at f_l;
- Strategy II (CP-aware Slack Reclamation): Execute t at the optimal CPU frequency f_m with which T' is eliminated, i.e., run in $T+T'$ at f_m (For simplicity in the later discussion, assume T' can be eliminated using available frequency f_m without frequency approximation).

For simplicity, let us assume the tasks for the use of DVFS are compute-intensive; i.e, $T+T'=nT$, when $f_m=\frac{1}{n}f_h$, where $1\le n\le\frac{f_h}{f_l}$. Consider the nodal power consumption P, we model it formally as follows:

$$P = P^{CPU}_{dynamic} + P^{CPU}_{leakage} + P^{other}_{leakage} \quad (1)$$

$$P_{dynamic} = ACfV^2 \quad (2)$$

$$P_{leakage} = I_{sub}V \quad (3)$$

Then, substituting Equations 2 and 3 into Equation 1 yields:

$$P = ACfV^2 + I_{sub}V + I'_{sub}V' \quad (4)$$

In our scenario, $P^{other}_{leakage} = I'_{sub}V'$ is independent of CPU voltage and frequency scaling, and thus can be regarded as a constant in Equation 4, so we denote $P^{other}_{leakage}$ as P_c for simplicity. Further, although subthreshold leakage current I_{sub} has an exponential relationship with threshold voltage, results presented in [13] indicate that I_{sub} converges to a constant after a certain threshold voltage value. Without loss of generality, we treat $P^{CPU}_{leakage} = I_{sub}V$ as a function of supply voltage V only. Thus, we model the nodal energy consumption E_{node} for both strategies individually below:

$$\begin{aligned} E(\mathsf{S}_1) &= \overline{P(\mathsf{S}_1)}\times T + \overline{P'(\mathsf{S}_1)}\times T' \\ &= (ACf_hV_h^2 + I_{sub}V_h + P_c)T + (ACf_lV_l^2 + I_{sub}V_l + P_c)T' \\ &= AC(f_hV_h^2T + f_lV_l^2T') + I_{sub}(V_hT + V_lT') + P_c(T+T') \end{aligned} \quad (5)$$

$$\begin{aligned} E(\mathsf{S}_2) &= \overline{P(\mathsf{S}_2)}\times(T+T') \\ &= (ACf_mV_m^2 + I_{sub}V_m + P_c)(T+T') \\ &= ACf_mV_m^2(T+T') + I_{sub}V_m(T+T') + P_c(T+T') \end{aligned} \quad (6)$$

We obtain the difference between energy costs of both strategies by subtracting Equation 5 from Equation 6:

$$\begin{aligned} E(\mathsf{S}_2) - E(\mathsf{S}_1) = {} & AC\left((f_mV_m^2 - f_hV_h^2)T + (f_mV_m^2 - f_lV_l^2)T'\right) \\ & + I_{sub}\left((V_m - V_h)T + (V_m - V_l)T'\right) \end{aligned} \quad (7)$$

Denote the first term as ΔE_d and the second term as ΔE_l in the above addition of products. Substituting the assumption that $T' = (n-1)T$ and $f_m = \frac{1}{n}f_h$ into both terms yields simplified formulae:

$$\begin{aligned} \Delta E_d &= AC\left(\left(\frac{1}{n}f_hV_m^2 - f_hV_h^2\right)T + \left(\frac{1}{n}f_hV_m^2 - f_lV_l^2\right)(n-1)T\right) \\ &= AC\left(\left(\frac{1}{n}f_hV_m^2 - f_hV_h^2\right)T + \left(\frac{n-1}{n}f_hV_m^2 - (n-1)f_lV_l^2\right)T\right) \\ &= ACT\left(f_h\left(V_m^2 - V_h^2\right) - (n-1)f_lV_l^2\right) \end{aligned} \quad (8)$$

$$\begin{aligned} \Delta E_l &= I_{sub}\left((V_m - V_h)T + (V_m - V_l)(n-1)T\right) \\ &= I_{sub}T\left(nV_m - V_h - (n-1)V_l\right) \end{aligned} \quad (9)$$

Given the fact that voltage has a positive correlation with (i.e., not strictly proportional/linear to) frequency (scaling up/down frequency results in voltage up/down accordingly as shown in Table 2), from Equation 8 we conclude that ΔE_d is a monotonically decreasing function for n, where the maximum 0 is attained when $n=1$, i.e., when slack T' equals 0. Although generally $\Delta E_d \le 0$, state-of-the-art CMOS technologies allow insignificant variation of voltage as frequency scales (see Table 2). Consequently the term $V_m^2 - V_h^2$ within ΔE_d is not a large value. Moreover, the ratio between the highest frequency and the lowest one determines the upper bound of n, so the term $(n-1)f_lV_l^2$ is not significant either. Equation 9 indicates that ΔE_l is a non-monotonic function for n, since V_m decreases as n increases.

EXAMPLE. From the operating points of different processors shown in Table 2, we can calculate numerical energy savings for different n values for a specific processor configuration, and thereby quantify energy efficiency of both energy saving strategies. For instance, for AMD Opteron 2218 processor, given a task with the execution time T and slack $0.25T$, i.e., $n = 1.25$, for eliminating the slack, 1.8 GHz is adopted as the working frequency for running the task, and thus $\Delta E_d = ACT \times (2.4 \times (1.15^2 - 1.25^2) - (1.25 - 1) \times 1.0 \times 1.1^2) = -0.8785 \times ACT$; $\Delta E_l = I_{sub}T \times (1.25 \times 1.15 - 1.25 - (1.25 - 1) \times 1.1) = -0.0875 \times I_{sub}T$; $E(\mathsf{S}_2) - E(\mathsf{S}_1) = \Delta E_d + \Delta E_l = -0.8785 \times ACT - 0.0875 \times I_{sub}T < 0$. We can see that with slightly higher energy costs, Strategy I is comparable to Strategy II in energy efficiency.

4 Scalability Analysis

Regardless of energy saving capability, a scalable energy saving solution can prevail for today's ever-growing supercomputers. Due to the nature of slowing down processors during identified slack instead of placing them in an idle mode, *CP-aware slack reclamation* can be superior to *race-to-halt* in terms of power scalability. We next use an example to illustrate the case.

Consider a IBM Blue Gene/Q configured cluster that has a power range from 9 MW at full load (e.g., running the High Performance LINPACK benchmark) to 0.1 MW when idle. Assume a CPU-bound and load-imbalanced application is running on the cluster, where 1% of nodes need to run 10% longer than other nodes. When 99% of nodes have completed their tasks and been placed into an idle mode by *race-to-halt*, the total system power costs amount to around 0.2 MW (0.1 MW $\times$ 0.99 + 9 MW $\times$ 0.01) for the rest 10% execution time, when there is a huge drop from 9 MW to 0.2 MW in the total system power. The case is even worse if the power variation happens within a loop. If the interval of power variation is small enough, the power gap can be absorbed in the capacitors on the motherboard or in the nodal power supply. Otherwise the huge power spike will be reflected on the transmission lines, which jeopardizes the hardware reliability of the whole system. The case is however greatly mitigated if the load is balanced, or the load imbalance is caused by inevitable data dependencies among tasks, without considering the effect of looping.

5 Experimental Evaluation

In this section, we validate our findings aforementioned. We applied both energy saving solutions individually to an MPI implementation of one widely used numerical linear algebra operation Cholesky factorization to assess their energy efficiency empirically. Experiments were performed on a large-scale power-aware cluster ARC,

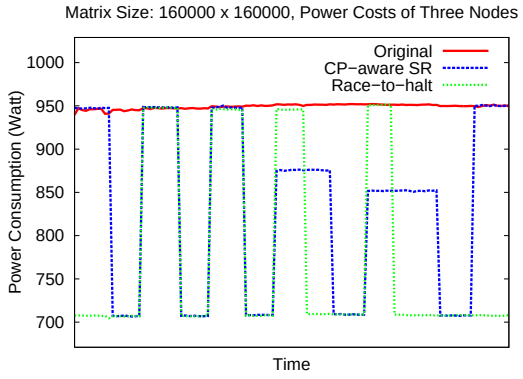

Figure 2: Power Consumption of Cholesky Factorization with Two Energy Efficient Approaches on Cluster ARC.

equipped with an 40 GB/s Infiniband switch and consisting of 108 computing nodes with two 8-core AMD Opteron 6128 processors (totalling 1728 cores) and 32 GB RAM running 64-bit Linux kernel 2.6.32. The range of CPU frequency on ARC was {0.8, 1.0, 1.2, 1.5, 2.0} GHz. The total of static and dynamic power consumption was measured using Watts up? PRO [3] power meter, which is shared by three ARC nodes. Thus the power consumption measured is the total value of three nodes. CPU frequency scaling was implemented via CPUFreq [1] which directly modifies CPU frequency system configuration files. We did not utilize the whole cluster but only a 16 $\times$ 16 process grid (totalling 256 cores), which is sufficient to demonstrate solid power results.

Next we present preliminary results on power and performance efficiency of the two approaches for the target application by fine-grained comparison.

POWER SAVINGS. First we evaluate the capability of saving power from the two energy efficient approaches, taking distributed Cholesky factorization running on the ARC cluster for example, where power consumption is measured by sampling at a constant rate through the execution of the application. Figure 2 depicts the total system power consumption of three nodes (out of sixteen nodes in use) running the application with the two approaches individually using a 160000 $\times$ 160000 global matrix. Here we present time durations of the first few iterations, where the core loop performs alternating computation and communication with decreasing execution time of each iteration, as the remaining unfactorized matrix shrinks. Thus we can see that for all curves, from left to right, the durations of computation (i.e., the peak power values) decrease as the factorization proceeds.

The three executions manifest three different power variation patterns. The *original* run employed the same highest CPU performance for computation and communication, resulting almost constant power costs around 950 Watts. The *CP-aware slack reclamation* approach

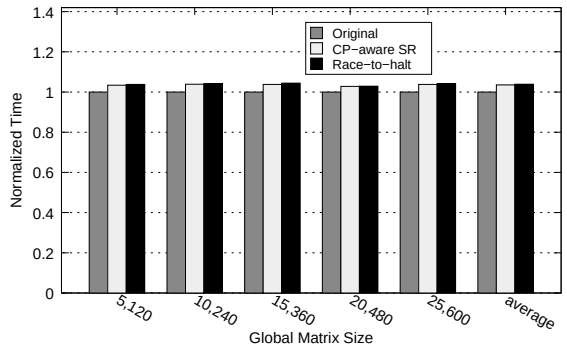

Figure 3: Performance of Cholesky Factorization with Two Energy Efficient Approaches on the ARC Cluster.

slowed down computation to eliminate slack, while the *race-to-halt* approach lowered down CPU performance to the minimum scale for all durations other than computation. Both energy saving approaches scaled down CPU performance during the communication, i.e., the five low-power durations around 700 Watts, and resumed the peak CPU performance when computation started.

Energy saving solutions only slow down processors during communication are semi-optimal. Thus both *CP-aware slack reclamation* and *race-to-halt* are expected to utilize computation slack for further energy savings. The difference lies in that the former requires detection of CP and calculation of the extent of slowing down per the amount of slack, while the latter only needs to know when the slack arises, which is much easier to implement and deploy. Figure 2 demonstrates that *CP-aware slack reclamation* succeeded to lower power states down to an intermediate scale, i.e., the two medium-power durations around 850 Watts during the third and the fourth computation as the blue line shows. Whereas *race-to-halt* observed when the computation started and ended, and utilized the peak CPU performance when it started and switched to the lowest power state immediately when it ended. Moreover, the nature of *race-to-halt* also guarantees no high-power states are employed during the waiting durations resulting from load imbalance and data dependency, i.e., the two low-power durations in green where the application started and ended.

PERFORMANCE TRADE-OFF. Both energy saving approaches incur minor performance loss while achieving considerable power savings. Figure 3 illustrates slowdown of the two approaches compared to the original runs. The time overhead on employing *CP-aware slack reclamation* and *race-to-halt* are negligible: 3.5% and 3.9% on average respectively. Besides the time overhead on employing the DVFS techniques [12], additional performance loss is caused by both approaches individually. Detection of CP and slack and frequency calculation (in some cases frequency approximation is also needed) are necessary to perform *CP-aware slack reclamation*. *Race-to-halt* requires to monitor the completion of tasks to determine the appropriate timing for power state switching. Generally the time overhead incurred by both approaches are acceptable in an HPC environment.

6 Future Work

The debate between energy efficiency of *CP-aware slack reclamation* and *race-to-halt* is an ongoing issue as hardware technologies advance. Preliminary theoretical and experimental results provide us solid support to extend this work. We intend to model and generalize the idea of *race-to-halt* for more scientific applications and present complete empirical evidence to validate our observation. The implementation of both energy saving solutions is in the prototype stage, and we also plan to apply more energy efficient approaches on emerging accelerated architectures for investigating the optimal energy savings.

References

[1] *CPUFreq - CPU Frequency Scaling*. https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling.

[2] *US Department of Energy Exascale Computing Initiative 2012*. http://science.energy.gov/ /media/ascr/ascac/pdf/meetings/aug12/2012-ECI-ASCAC-v4.pdf.

[3] *Watts up? Meters*. https://www.wattsupmeters.com/.

[4] DAVIES, T., KARLSSON, C., LIU, H., DING, C., AND CHEN, Z. High performance linpack benchmark: A fault tolerant implementation without checkpointing. In *ICS* (2011), pp. 162–171.

[5] DINAN, J., LARKINS, D. B., SADAYAPPAN, P., KRISHNAMOORTHY, S., AND NIEPLOCHA, J. Scalable work stealing. In *SC* (2009), p. 53.

[6] FREEH, V. W., AND LOWENTHAL, D. K. Using multiple energy gears in MPI programs on a power-scalable cluster. In *PPoPP* (2005), pp. 164–173.

[7] GE, R., FENG, X., FENG, W.-C., AND CAMERON, K. W. CPU MISER: A performance-directed, run-time system for power-aware clusters. In *ICPP* (2007), p. 18.

[8] ISHIHARA, T., AND YASUURA, H. Voltage scheduling problem for dynamically variable voltage processors. In *SC* (1998), pp. 197–202.

[9] RIZVANDI, N. B., TAHERI, J., AND ZOMAYA, A. Y. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *Journal of Parallel Distributed Computing 71*, 8 (Aug. 2011), 1154–1164.

[10] ROUNTREE, B., LOWENTHAL, D. K., DE SUPINSKI, B. R., SCHULZ, M., FREEH, V. W., AND BLETSCH, T. Adagio: Making DVS practical for complex HPC applications. In *ICS* (2009), pp. 460–469.

[11] ROUNTREE, B., LOWENTHAL, D. K., FUNK, S., FREEH, V. W., DE SUPINSKI, B. R., AND SCHULZ, M. Bounding energy consumption in large-scale MPI programs. In *SC* (2007), pp. 1–9.

[12] TAN, L., CHEN, L., CHEN, Z., ZONG, Z., GE, R., AND LI, D. HP-DAEMON: High performance distributed adaptive energy-efficient matrix-multiplication. In *ICCS* (2014), pp. 599–613.

[13] TAUR, Y., LIANG, X., WANG, W., AND LU, H. A continuous, analytic drain-current model for DG MOSFETs. *IEEE Electron Device Letters 25*, 2 (Feb. 2004), 107–109.