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

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

ABSTRACT

The presence of pervasive slack provides ample opportunities for achieving energy efficiency for HPC systems 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 that state-of-the-art CMOS technologies allow insignificant variation of supply voltage as operating frequency of a processor scales. Experimental results on a large-scale power-aware cluster validate our findings.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling, Multiprocessing/multiprogramming/multitasking*

Keywords

energy; power; DVFS; HPC; critical path; slack; scalable

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 (HPC), 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 HPC 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 HPC applications [6] [10] [9].

Critical Path (CP) is one particular task trace from the beginning task of a task-parallel HPC run 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 energy saving capability of two classic energy saving approaches, and formally calculate and compare energy savings from both solutions. Previous formal proof shows that *CP-aware slack reclamation* beats *race-to-halt* in terms of energy efficiency [7] [8]. 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.

2. CLASSIC ENERGY SAVING STRATEGIES

Existing energy efficient approaches that save energy strategically during slack of HPC runs 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 HPC run 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

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.
Copyright © 2015 ACM 978-1-4503-3248-4/15/01 ...$15.00.
http://dx.doi.org/10.1145/2668930.2695530.

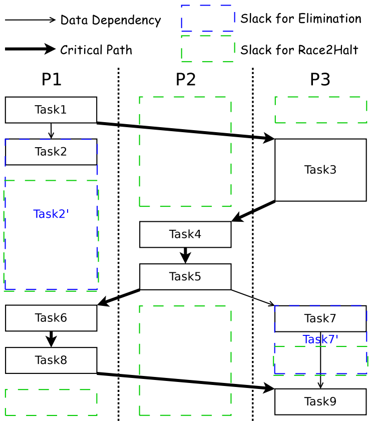

Figure 1: DAG Notation of Slack Handling of Two Energy Saving Solutions for a 3-Process HPC Run.

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 this *scheduled communication* [10] strategy for communication slack. However, 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, i.e., at the highest frequency and voltage of processors, until the finish of the workloads, while *halt* means to slow down processors to the minimum 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 run. 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 CAPACITY 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 [7] [8]. However, the gap between energy saving capability of *race-to-halt* and *CP-aware 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 formalize 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:

Table 1: Notation in Energy Efficiency 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

- Strategy I (Race-to-halt): Execute t at the highest frequency f_h until the end, and then switch to the lowest 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 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 (memory-intensive tasks can be discussed with minor changes in the model), i.e., $T + T' = nT$, when $f_m = \frac{1}{n} f_h$, where $1 \leq n \leq \frac{f_h}{f_l}$. Consider the nodal power consumption P, we model it formally as follows:

$$P = P^{CPU}_{dynamic} + P^{CPU}_{leakage} + P^{other}_{leakage} \tag{1}$$

$$P_{dynamic} = ACfV^2 \tag{2}$$

$$P_{leakage} = I_{sub}V \tag{3}$$

Substituting Equations 2 and 3 into Equation 1 yields:

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

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 [11] 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:

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

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

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

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

$$E(\mathsf{S}_2) - E(\mathsf{S}_1) = AC\left((f_m V_m^2 - f_h V_h^2)T + (f_m V_m^2 - f_l V_l^2)T'\right) + I_{sub}\left((V_m - V_h)T + (V_m - V_l)T'\right) \quad (7)$$

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

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

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

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 $\Delta E_d \leq 0$, 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 $(n-1)f_l V_l^2$ 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, and thus quantify energy efficiency of the two approaches. 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 efficient solution prevails for today's supercomputers. Due to the nature of slowing down processors during identified slack instead of switching to 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 × 0.99 + 9 MW × 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, 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 × 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 HPC runs.

Power Savings. First we evaluate the capability of saving power from the two energy efficient approaches, taking 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 × 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.

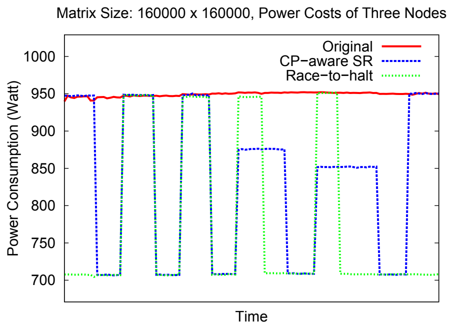

Figure 2: Power Costs of Cholesky Factorization with Two Energy Saving Solutions on Cluster ARC.

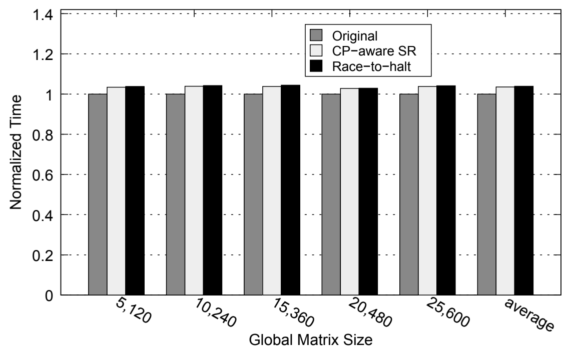

Figure 3: Performance of Cholesky Factorization with Two Energy Saving Solutions on Cluster ARC.

The three runs manifest three different power variation patterns. The *original* run used the same highest CPU frequency for computation and communication, resulting almost constant power costs around 950 Watts. The *CP-aware slack reclamation* approach 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 approaches employed the lowest CPU frequency during 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 slow-down 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, 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 a message-passing 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. We intend to model and generalize both solutions for more scientific applications and present complete empirical evidence to validate our observations. We also plan to apply improved solutions on emerging accelerated architectures for gaining the optimal energy savings.

7. REFERENCES

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

[2] *US DOE 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] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen. High performance linpack benchmark: A fault tolerant implementation without checkpointing. In *ICS*, pages 162–171, 2011.

[5] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha. Scalable work stealing. In *SC*, page 53, 2009.

[6] V. W. Freeh and D. K. Lowenthal. Using multiple energy gears in MPI programs on a power-scalable cluster. In *PPoPP*, pages 164–173, 2005.

[7] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *SC*, pages 197–202, 1998.

[8] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *Journal of Parallel Distributed Computing*, 71(8):1154–1164, Aug. 2011.

[9] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making DVS practical for complex HPC applications. In *ICS*, pages 460–469, 2009.

[10] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Bounding energy consumption in large-scale MPI programs. In *SC*, pages 1–9, 2007.

[11] Y. Taur, X. Liang, W. Wang, and H. Lu. A continuous, analytic drain-current model for DG MOSFETs. *IEEE Electron Device Letters*, 25(2):107–109, Feb. 2004.