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 rely 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...
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 preceding 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 for 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., \( E(S_1) \) and \( E(S_2) \) formally below:

- **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 \( 1 \leq n \leq \frac{f_h}{f_m} \). Consider the nodal power consumption \( P \); we model it formally as follows:

\[
P = P_{\text{dynamic}} + P_{\text{leakage}} + P_{\text{other}}
\]

\[
P_{\text{dynamic}} = ACfV^2
\]

\[
P_{\text{leakage}} = I_{\text{sub}}V
\]

Substituting Equations 2 and 3 into Equation 1 yields:

\[
P = ACfV^2 + I_{\text{sub}}V + I'_{\text{sub}}V'
\]

In our scenario, \( P_{\text{other}} = I'_{\text{sub}}V' \) is independent of CPU voltage and frequency scaling, and thus can be regarded as a constant in Equation 4, so we denote \( P_{\text{leakage}} \) as \( P_l \) for simplicity. Further, although subthreshold leakage current \( I_{\text{sub}} \) has an exponential relationship with threshold voltage, results presented in [11] indicate that \( I_{\text{sub}} \) converges to a constant after a certain threshold voltage value. Without loss of generality, we treat \( P_{\text{leakage}} = I_{\text{sub}}V \) as a function of supply voltage \( V \) only. Thus, we model the nodal energy consumption \( E_{\text{node}} \) for both strategies individually below:

\[
E(S_1) = P(S_1) \times T + P(S_1) \times T'
\]

\[
= (ACf_hV_h^2 + I_{\text{sub}}V_h + P_c)T + (ACf_lV_l^2 + I_{\text{sub}}V_l + P_l)T'
\]

\[
= AC(f_hV_h^2T + f_lV_l^2T') + I_{\text{sub}}(V_hT + V_lT') + P_c(T + T')
\]

\[
E(S_2) = P(S_2) \times (T + T')
\]

\[
= (ACf_mV_m^2 + I_{\text{sub}}V_m + P_c)(T + T')
\]

\[
= ACf_mV_m^2(T + T') + I_{\text{sub}}V_m(T + T') + P_c(T + T')
\]

### Table 1: Notation in Energy Efficiency Analysis

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>Total nodal energy consumption of all components</td>
</tr>
<tr>
<td>( P )</td>
<td>Total nodal power consumption of all components</td>
</tr>
<tr>
<td>( P_{\text{dynamic}} )</td>
<td>Dynamic power consumption in the running state</td>
</tr>
<tr>
<td>( P_{\text{leakage}} )</td>
<td>Static/leakage power consumption in any states</td>
</tr>
<tr>
<td>( T )</td>
<td>Execution time of a task at CPU peak performance</td>
</tr>
<tr>
<td>( T' )</td>
<td>Slack of executing a task at CPU peak performance</td>
</tr>
<tr>
<td>( A )</td>
<td>Percentage of active gates in a CMOS-based chip</td>
</tr>
<tr>
<td>( C )</td>
<td>The total capacitive load in a CMOS-based chip</td>
</tr>
<tr>
<td>( f )</td>
<td>Current CPU working frequency</td>
</tr>
<tr>
<td>( V )</td>
<td>Current CPU supply voltage</td>
</tr>
<tr>
<td>( V' )</td>
<td>Supply voltage of components other than CPU</td>
</tr>
<tr>
<td>( f_{\text{sub}} )</td>
<td>CPU subthreshold leakage current</td>
</tr>
<tr>
<td>( f_{\text{sub}} )</td>
<td>Non-CPU subthreshold leakage current</td>
</tr>
<tr>
<td>( f_m )</td>
<td>Available frequency assumed to eliminate ( T' ) without using frequency approximation</td>
</tr>
<tr>
<td>( V_h )</td>
<td>The highest supply voltage set by DVFS</td>
</tr>
<tr>
<td>( V_l )</td>
<td>The lowest supply voltage set by DVFS</td>
</tr>
<tr>
<td>( V_m )</td>
<td>Supply voltage corresponding to ( f_m ) set by DVFS</td>
</tr>
<tr>
<td>( n )</td>
<td>Ratio between execution time and slack of a task</td>
</tr>
</tbody>
</table>
Table 2: Frequency-Voltage Pairs for Different Processors (Unit: Frequency (GHz) and Voltage (V)).

<table>
<thead>
<tr>
<th>Processor</th>
<th>AMD Opteron 846</th>
<th>AMD Athlon64 3200+</th>
<th>AMD Opteron 2218</th>
<th>Intel Pentium M</th>
<th>Intel Pentium 4 HT 530</th>
<th>Intel Xeon ES 2687W</th>
<th>Intel Core i7-2760QM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2.5</td>
<td>1.300</td>
<td>2.0</td>
<td>1.500</td>
<td>1.250</td>
<td>1.0</td>
<td>1.100</td>
</tr>
<tr>
<td>1</td>
<td>1.8</td>
<td>1.300</td>
<td>1.8</td>
<td>1.500</td>
<td>2.2</td>
<td>1.100</td>
<td>1.2</td>
</tr>
<tr>
<td>2</td>
<td>1.3</td>
<td>1.100</td>
<td>1.6</td>
<td>1.300</td>
<td>1.2</td>
<td>1.100</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>0.8</td>
<td>1.100</td>
<td>0.8</td>
<td>0.900</td>
<td>1.0</td>
<td>1.100</td>
<td>0.8</td>
</tr>
</tbody>
</table>

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

\[ E(\bar{S}_2) - E(\bar{S}_1) = AC \left( \frac{1}{n} f_n (V_m^2 - V_b^2) T + \frac{n-1}{n} f_n (V_m^2 - V_b^2) \right) + I_{sub} \left( (V_m - V_b) T + (V_m - V_b) T^2 \right) \]

Denote the first term as \( \Delta E_d \) and the second term as \( \Delta E_l \). Substituting the assumption that \( T' = (n-1) T \) and \( f_n = \frac{1}{n} f_h \) into both terms yields simplified formulae:

\[ \Delta E_d = AC \left( \frac{1}{n} f_n (V_m^2 - V_b^2) T + \frac{n-1}{n} f_n (V_m^2 - V_b^2) \right) \]

\[ \Delta E_l = I_{sub} \left( (V_m - V_b) T + (V_m - V_b) T^2 \right) \]

Given the fact that voltage has a positive correlation with (i.e., not strictly proportionate/linear to) frequency (scaling up/down frequency results in voltage up/down accordingly as shown in Table 2), from Equation 8 we conclude that \( \Delta E_d \) 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 \( V_m^2 - V_b^2 \) within \( \Delta E_d \) 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 \( (n-1)f_i V_i^2 \) is not significant either. Equation 9 indicates that \( \Delta E_l \) 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 \( \Delta E_d = ACT \times (2.4 \times (1.15^2 - 1.25^2) - (1.25 - 1) \times 1.0 \times 1.1^2) = -0.8785 \times ACT; \Delta E_l = I_{sub} T \times (1.25 \times 1.5 - 1.25 - (1.25 - 1) \times 1.1) = -0.0875 \times I_{sub} T; E(\bar{S}_2) - E(\bar{S}_1) = \Delta E_d + \Delta E_l = -0.8785 \times ACT - 0.0875 \times I_{sub} T < 0 \). We can see that with slightly higher energy costs, Strategy I is comparable to Strategy II in energy efficiency.

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 (tallring 1728 cores) and 32 GB RAM running 64-bit Linux kernel 2.6.32. The range of CPU frequency on ARC was \( 0.8 \) to \( 2.0 \) 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 (tallring 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 \( \times \) 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 unfactored 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 runs manifest three different power variation patterns. The original run used the same highest CPU frequency for computation and communication, resulting in 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 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 slack and frequency calculation, while the latter only needs to know when the slack ends immediately when it ends. 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