Energy Optimization of Memory Intensive Parallel Workloads

Energy optimization of memory intensive parallel workloads

Brief Announcement

Chhaya Trehan
c.trehan@qub.ac.uk

Hans vandierendonck
h.vandierendonck@qub.ac.uk

Georgios Karakonstantis
G.Karakonstantis@qub.ac.uk

Dimitrios S. Nikolopoulos
d.nikolopoulos@qub.ac.uk

School of Electronics, Electrical Engineering and Computer Science
Queen's University of Belfast
Belfast, UK

ABSTRACT

Energy consumption is an important concern in modern multicore processors. The energy consumed by a multicore processor during the execution of an application can be minimized by tuning the hardware state utilizing knobs such as frequency, voltage etc. The existing theoretical work on energy minimization using Global DVFS (Dynamic Voltage and Frequency Scaling), despite being thorough, ignores the time and the energy consumed by the CPU on memory accesses and the dynamic energy consumed by the idle cores. This article presents an analytical energy-performance model for parallel workloads that accounts for the time and the energy consumed by the CPU chip on memory accesses in addition to the time and energy consumed by the CPU on CPU instructions. In addition, the model we present also accounts for the dynamic energy consumed by the idle cores. The existing work on global DVFS for parallel workloads shows that using a single frequency for the entire duration of a parallel application is not energy optimal and that varying the frequency according to the changes in the parallelism of the workload can save energy. We present an analytical framework around our energy-performance model to predict the operating frequencies (that depend upon the amount of parallelism) for global DVFS that minimize the overall CPU energy consumption. We show how the optimal frequencies in our model differ from the optimal frequencies in a model that does not account for memory accesses. We further show how the memory intensity of an application affects the optimal frequencies.

Keywords

Energy Optimization; Global DVFS; Convvex Programming; Memory Intensive; Parallel Applications; Multicore

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

1. INTRODUCTION

While Silicon is available in abundance to build processors, the energy required to power them is not. Energy consumption and performance turn out to be the two most important and contradicting design criteria for the modern multicore processors. The practice of dealing with the two contradicting goals by optimizing one while imposing a threshold on the other leads to two flavors of energy-performance optimization called the *laptop problem* and the *server problem*. In the laptop problem, the goal is to maximize the performance given a fixed energy budget and in the server problem, the goal is to minimize the energy consumption given a fixed performance budget [1]. We deal with the *server problem* in this article.

The power consumption a CMP (Chip Multi Processor) is an increasing function of the operating frequency of the chip and thus can be reduced by reducing the frequency. Dynamic Voltage and Frequency Scaling (DVFS) is a popular energy minimization technique. Most of the theoretical work on the energy-delay trade off deals with the *local* dynamic voltage and frequency scaling [3], where frequency can be set separately for each core. We study the problem of energy minimization under a performance constraint using global DVFS where all the cores on the chip are set to run on the same frequency. While local DVFS has more freedom in choosing clock frequencies and can therefore save more energy, it is not easy to implement.

Often when we think of parallelism, we think of performance gains and we tend to ignore its ramifications on energy consumption. The fact that we can gain performance by increasing parallelism allows us to save energy by reducing the frequency without violating a performance constraint. The relationship of parallelism with energy and performance was first studied by Sangyeun and Melhem [2]. Gerards et al [3] further formalized the problem for task graphs. They showed that using a single clock frequency during the execution of an application does not lead to optimal energy consumption. They presented an approach for varying the frequency according to the variations in the amount of parallelism assigning a separate frequency to each number of active cores. The analytical model of [3], however, completely ignores the energy consumed by the CPU while it waits for data accesses to the main memory. Since it does not account for the time overhead of the access latency of memory, it can lead to an imprecise estimate of the slack between the time to completion and the given performance budget. The CPU energy optimization techniques that save energy by decreasing the operating frequencies of the cores at the cost of an increased delay need to be tuned to account for the memory access latencies. Another assumption in [3] is that the frequency of the idle cores can be brought down to zero by techniques like clock gating. This is not always possible in reality, the idle cores can't be completely shut down and do consume some dynamic energy. In this article, we present a new model for the energy and performance of multicore systems that accounts for the energy consumed by the CMP while waiting for memory accesses in addition to the energy consumed on CPU instructions without ignoring the dynamic energy consumed by the idle cores.

1.1 Model

We consider an application running on a multicore processor. The application itself consists of a set T of N tasks, denoted by $\{T_1, ...T_N\}$. The application can be depicted as a labeled DAG (Directed Acyclic Graph) where nodes represent the tasks and the (*Directed*) edges represent the precedence constraints (figure: 1). A task T_i is characterized by two attributes, namely: the *compute workload:* cw_i and the *data workload:* dw_i. The compute work load (label of the corresponding node in the task graph) is the number of clock cycles required to perform the computations of the task. The data workload is the number of memory accesses a task has to make during its execution. We assume an application wide parameter called *data to CPU* quotient d which is the ratio of data to compute workloads of the application. We consider an overall deadline t_{budget} for the entire application.

Power Model: As is common in literature, we consider two components of power, the *dynamic power* and the *static power*. Assuming f is the frequency of all the cores at some time t, the dynamic power of an active core at time t can be expressed as a function of frequency as $p_{DynamicActive}(f) = c1f^{\alpha}$

The constant $c1 > 0$ is a characteristic of the computing platform and the exponent α is a constant(≥ 2). At any given point in time, an inactive core consumes relatively less dynamic power owing to its reduced activity factor. We model this difference in dynamic power of active and inactive cores by assuming that the constant $c1$ for inactive cores is less than the $c1$ for active cores. Assuming $c1'$ to be the constant for inactive cores such that the ratio $K = \frac{c1'}{c1} < 1$, the dynamic power of an inactive core can be expressed as $p_{DynamicInactive}(f) = c1'f\alpha$. The static power can also be expressed as an affine function of frequency $p_{Static}(f) = c2f + c3$

At a given point in time, with m active cores running at an operating frequency of f, the total power of the processor chip can be expressed as:

$$p_m(f) = [m + k(M - m)]c1f^{\alpha} + c2f + c3 \qquad (1)$$

where M is the total number of cores on the chip. This is a convex and increasing function in f. From this point on, we will denote $[m + k(M - m)]$ as m' for the sake of brevity. Dividing equation 1 on both sides by f gives energy per CPU cycle which we will denote as $\bar{p}_m$ henceforth

$$\bar{p}_m(f) = m'c1f^{\alpha-1} + c2 + \frac{c3}{f} \qquad (2)$$

Before we go into the details of selecting the optimal frequencies in our model, we take a short diversion to understand what an interval $(t1, t2)$ in our model looks like and how the presence of memory accesses during an interval change the dynamics of energy optimization. In any interval during the execution, all the active cores are performing some memory accesses uniformly interleaved with the CPU instruction cycles. In any such interval, not all of the CPU cycles produced can be counted towards the work done (instructions) by the CPU. Moreover the time spent on memory accesses is independent of frequency whereas the time spent on executing the instructions can be increased (decreased) by decreasing (increasing) the frequency. We can think of an interval without any memory accesses like a spring and frequency as a force, the spring can be wound\unwound by applying the force. DVFS schemes for energy optimization exploit this ability to stretch an interval by decreasing the frequency to minimize energy consumption at the cost of

increased delays. An interval with memory accesses can be thought of as composed of many springs with some rigid material placed between them. Applying a force can only compress or decompress the springs and the rigid material (memory accesses) does not yield at all to the changes in frequency. Thus, only a portion of interval can be stretched by decreasing the frequency thus leading to a lesser potential for reduction in energy by decreasing the frequency.

Parallelism and Energy-Performance model: The overall energy consumption of an application can be expressed in terms of the amount of parallelism. In interest of brevity, we refer the reader to go through [3] to fully appreciate the concept of power modelling in terms of parallelism. For an application with N tasks running on a processor with M cores, its amount of parallelism for a given schedule can be defined formally as a vector $[w_1, w_2, ...w_m...w_M]$, where w_m is the total number of CPU cycles for which exactly m cores are active. Using the idea that a constant frequency for a fixed number of cores (parallelism) leads to an optimal energy consumption [3] ,the task of global DVFS for energy optimization is reduced to finding a vector $f = [f_1, f_2, ...f_m...f_M]$ of frequencies where f_m is the optimal frequency to be used when m cores are active. Energy consumed when m cores are active can be expressed as the product of energy per cycle $\bar{p}_m$ from equation 2 and w_m. For a given amount of parallelism w_m , $w_m d$ accesses to memory are made, where d is the application wide data to CPU workload ratio. The CPU keeps clocking at a frequency f_m for the duration of these $w_m d$ memory accesses. If t_a is the latency of memory accesses, $(w_m d)t_a$ is the duration for which the CPU waits for memory accesses. The additional cycles expended per core on memory accesses for w_m is thus $w_m dt_a f_m$. The total energy consumed by the CPU in terms of parallelism and the corresponding frequencies can be expressed as $E_{total}(f_1, f_2, ...f_M) = \sum_{m=1}^{M} [\bar{p}_m(f_m)(w_m + w_m dt_a f_m)]$

The time to completion of an application for a given schedule can also be expressed in terms of parallelism and the corresponding frequencies as
$t_{completion}(f_1, f_2, ...f_M) = \sum_{m=1}^{M} \frac{w_m}{f_m} + \sum_{m=1}^{M} w_m dt_a$

2. MAIN RESULTS

In this section, we present the main results of our research so far. We investigate how do the optimal frequencies relate to the memory intensity (data to CPU workload ratio, d) of an application and whether and how the relationship between optimal frequencies and the number of active cores change in the presence of memory accesses. We present our results in the form of three lemmas.

LEMMA 1. *It holds for every pair $n, m \in \{1, 2, ...M\}$ such that $m \geq n$ and $w_m, w_n > 0$ that:*

1. *for an optimal solution $f = [f1, f2, ...f_M]$ to the constrained energy optimization problem, $\frac{f_m}{f_n}$ lies in the interval $[\sqrt[\alpha]{\frac{n'}{m'}}, 1]$.*

2. *for an optimal solution $f = [f1, f2, ...f_M]$ to the constrained energy optimization problem without the static energy, $\frac{f_m}{f_n}$ lies in the interval $[\sqrt[\alpha]{\frac{n'}{m'}}, \sqrt[\alpha+1]{\frac{n'}{m'}}]$.*

LEMMA 2. *for an optimal solution $f = [f1, f2, ...f_M]$ to the constrained energy optimization problem without the static energy, the following holds for every pair $n, m \in \{1, 2, ...M\}$ with $w_m, w_n > 0$:*

$$\sqrt[\alpha]{m'[\alpha - 1 + \alpha dt_a f_m]} f_m = \sqrt[\alpha]{n'[\alpha - 1 + \alpha dt_a f_n]} f_n \quad (3)$$

Lemma 2 shows the relationship between the frequencies for two different parallel regions of a given schedule of an application. This is in contrast to the corresponding relationship in [3], which is, $\sqrt[\alpha]{n} f_n = \sqrt[\alpha]{m} f_m$.

Including d and k (m' instead of m), makes the relation more precise, this is particularity important in the case of memory intensive applications.

Having a relationship between the optimal frequencies for different parallel regions of a schedule, the next natural step is to be able to analytically relate an optimal frequency for a parallel region to the optimal frequency of the serial region. Lemma 3 gives such a relation for $\alpha = 2$.

LEMMA 3. *For $\alpha = 2$, the ratio $x_m = \frac{f_m}{f_1}$, of the optimal frequency f_m for a parallel region of the schedule with m active cores and the optimal frequency f_1 for the serial region is a solution to the following cubic equation:*

$$\frac{m'}{1'} 2dt_a f1 x_m^3 + \frac{m'}{1'} x_m^2 - (2dt_a f1 + 1) = 0 \quad (4)$$

where $1'$ is a constant equal to $KM + (1 - K)$

Note that it is possible for a schedule to have no serial region at all ($w_1 = 0$). The purpose of expressing x_m in terms of f_1 and m' is to help understand by how much does the frequency for a given parallelization differ from f_1. One can think of f_1 as a *reference* frequency for a given hardware and application combination such that each of the optimal parallel frequencies f_m is related to the reference frequency f_1 by multiplicative factor x_m.

Coming back to equation 4, the coefficient of the cubic term is a product of the parallelization m', the memory characteristic $2dt_a$ of the workload and the reference serial frequency f_1 which depends upon the deadline and the memory intensity of the application. For the sake of analysis we call the term $2dt_a f_1$, the *memory overload factor*.

A careful analysis of equation 4 in relation to the memory overload factor reveals that as $2dt_a f_1$ changes from 0 to 1, the optimal ratio changes very quickly and attains the mid point of $\sqrt[2]{\frac{1}{m}}$ and $\sqrt[3]{\frac{1}{m}}$ and then it changes more slowly and later becomes almost constant close to $\sqrt[3]{\frac{1}{m}}$. So, as memory overhead increases, the optimal frequency for m active cores tends to be inversely proportional to $\sqrt[3]{m}$. Without accounting for the memory accesses or for CPU intensive applications on the other hand the optimal frequency for m active cores is inversely proportional to $\sqrt[2]{m}$. Thus the accounting for memory accesses does not allow as much reduction in frequencies for parallel regions as predicted by the model in [3] which ignores the memory accesses altogether. This confirms that the energy savings predicted by [3] are over optimistic especially in the case of memory intensive applications (with a high memory intensity, d) running on a slow hardware (with a high access delay, t_a) on a tight performance budget (with a high reference frequency, f_1). In general, from lemma 1 and generalizing the above exposition, it can be established that the optimal frequency for m active cores for a memory intensive application (with $\alpha dT_a f_1$ sufficiently large) is inversely proportional to $\sqrt[alpha+1]{m}$.

3. REFERENCES

[1] A. Benoit, P. Renaud-Goud, and Y. Robert. Models and complexity results for performance and energy optimization of concurrent streaming applications. *International Journal of High Performance Computing Applications*, 25(3):261–273, 2011.

[2] S. Cho and R. Melhem. On the interplay of parallelization, program performance, and energy consumption. *Parallel and Distributed Systems, IEEE Transactions on*, 21(3):342–353, March 2010.

[3] J. L. H. Marco E.T. Gerards and J. Kuper. On the interplay between global dvfs and scheduling tasks with precedence constraints. *IEEE TRANSACTIONS ON COMPUTERS*, 64(06), 2015.

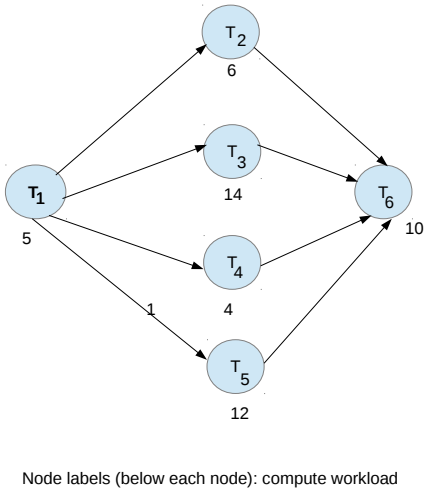

Figure 1: A task dependency graph