Folklore Confirmed: Compiling for Speed = Compiling for Energy

Tomofumi Yuki Sanjay Rajopadhye

August 27, 2013

Colorado State University Technical Report CS13-107

Abstract

As we move towards exa-scale computing, energy is becoming increasingly important, even in the high performance computing arena. However, the simple equation, Energy = Power × Time, suggests that optimizing for speed already optimizes for energy, under the assumption that Power is constant. When power is not constant, a strategy that achieves energy savings at the cost of slower execution is Dynamic Voltage and Frequency Scaling (DVFS). However, DVFS is currently applicable only to the processor, and the entire system has many other sources of power dissipation. We show that there is little to gain in compilers by trying to trade off speed for energy using DVFS. It is best to produce code that runs full-throttle, completing as quickly as possible, an approach called "race to sleep." Our result is based on analyses of a high-level energy model that characterizes energy consumption, related to survey of power consumption trends of recent processors for both desktop and server, as well as Cray supercomputers.

1 Introduction

The main motivations behind the arrival of multi-core processors were power and energy considerations. Increasing power density coupled with heat problems rendered untenable the premise that steadily increased performance could be achieved merely by steadily increasing processor clock speed. Multi-core processors were introduced based on the observation that multiple processors with lower frequency consume less total power, while preserving performance throughput [8]. Power and energy have been of great interest in the embedded systems community, where they were constrained by limited power capacity or battery life.

Even in the High Performance Computing (HPC) community, where the term "performance" had previously been synonymous with speed, power and energy are becoming more and more important. The annual cost for powering supercomputers, including their associated cooling systems, is now reaching 50% of the purchase cost of the machines and is expected to grow even further [24]. Power and energy are acknowledged to be the most difficult and pervasive challenges in order to achieve exa-scale computing [6]. In fact, if current hardware trends hold, there will remain a significant gap (a factor of 10 to 100) between predicted and required performance per watt, even under optimistic assumptions. It is therefore natural to explore possible compiler optimizations for power/energy efficiency.

It is known that current compiler optimizations also reduce total energy cost [29, 32]. Since the basic optimizations seek to speed up the computation, the equation, Energy = Power × Time, implies that optimizing for speed also optimizes energy, provided the average power remains constant. Moreover, many of the speed enhancing optimizations have a second order benefit that also reduces the power. For example, locality improving transformations like tiling increase the number of references that access local memory, such as caches, rather than off-chip memory. In addition to the low latency, caches also consume less power per access. Many authors have made this observation, and there seems to be a view in the folklore that in order to optimize for energy, compilers need to do no more than what they have always been doing—optimize for speed.

However, this naïve analysis assumes that power remains constant, which may not be true. Dynamic Voltage and Frequency Scaling (DVFS) is a technique that allows to dynamically change the operating frequency and voltage. As we shall see, DVFS implies that energy can be minimized by running as *slowly* as possible, or at least, as slow as one can get away with, until the response time becomes unacceptable and/or the components of the system, *not* governed by DVFS rules, come into play.

A number of studies [7, 14, 17, 27, 31] show that a significant fraction of the total power (more than 30%) comes from various components of the system that are not influenced by DVFS, such as motherboard, power supply unit, and memory. Moreover, around half of the power consumed by the processor comes from leakage power, where DVFS is significantly less effective. Thus, the effectiveness of DVFS must be considered with the energy consumption of the entire system included in the picture.

These considerations lead to the question whether there is any trade-off, where compilers need to perform any "special" optimizations that solely target energy savings, *without* necessarily reducing, or possibly even *increasing* execution time. We want to answer the question whether "is compiling for speed also compiling for energy," with respect to the use of DVFS. In this paper, we present analyses based on a high-level energy model that characterize this trade-off.

The main focus of our work is compute-bound programs, including as a limiting case, compute-I/O balanced programs and DVFS for processors. For these class of programs, we identify conditions under which using the highest frequency is most energy efficient. We show that on a large number of recent machines this condition is met. Therefore, we conclude that compilers should simply work on optimizing for speed.

2 Background

We first present an overview of various power/energy related aspects of processors that influence our model and analyses. Energy (E), Power (P), and Time (T) are related by the equation: $E = PT$ (more precisely, it is the integral over T when P changes over time). If an optimization keeps P unchanged, and reduces T, total energy consumption will decrease. The claim that optimizing for speed implicitly optimizes for energy comes from this observation.

Equation 1 below gives the simplified model of power dissipation of CMOS circuits [8]. The first term models the *dynamic* power consumption, where C is the total capacitance, V is the supply voltage, f is the clock frequency, and α is the "activity rate." The second term is the *static* power consumption (the power dissipated regardless of switching activity) where I_0 is the leakage current.

$$P_{proc} = \alpha C f V^2 + I_0 V \tag{1}$$

Dynamic Voltage and Frequency Scaling (DVFS) is an architectural feature that allows the supply voltage, and the corresponding running frequency to be changed at run time. Voltage and frequency are known to be linearly related. From Equation 1, power dissipation increases quadratically with voltage and linearly with operating frequency, DVFS can lead to cubic improvement in power dissipation. However, because of the linear relationship between voltage and frequency, there is also a linear degradation in speed. But reduction in power dissipation is cubic, and the degradation in execution time is only linear. To a first approximation, this leads to a quadratic reduction of energy as supply voltage is reduced.

Only the dynamic power component is amenable to DVFS optimization, the static power component decreases only linearly, and there is no net energy savings (in fact it is worse as we shall see later). Previously, dynamic power dominated the power consumption by processors, and thus power/energy optimizations focused on this component. It was predicted, and now observed, that static power consumption would reach 50% of the total power [14, 23].

3 Energy Model and Implications

We now present our energy consumption model, starting from a base model and progressively enhancing it. The following equation gives the energy consumption at maximum voltage and frequency:

$$E_{base} = (\alpha C f_{max} V_{max}^2 + I_0 V_{max} + P_c) T_{min} = (P_d + P_s + P_c) T_{min} \tag{2}$$

where the variables are defined as follows:

- P_d: maximum frequency dynamic power consumption of the processor,
- P_s: maximum frequency static power consumption of the processor,
- P_c: constant power; power consumed by various system components not influenced by DVFS, but excluding those due to program activity (such as memory/disk accesses), and
- T_{min}: is the execution time at the maximum frequency,

The energy consumed per access to memory/disk is not included in the model, since the number of accesses to memory/disk does not change as a result of frequency scaling. This is essentially a combination of $E = P \times T$ and Equation 1.

The above is a crude approximation as DVFS may indirectly influence energy consumption of various system components. For example, frequency of disk accesses may change, which in turn make the disk to switch between active and idle states more often, leading to larger energy consumption and vice versa.

Although we mentioned that the energy is the integral over time, product is sufficiently precise for our analysis. This is because when applying DVFS, programs are separated into relatively large regions where the frequency is fixed for each region. Since changing the frequency via DVFS comes with a cost in terms of both energy and time, frequent changes are not desirable.

3.1 Normalized Energy model for DVFS

Under DVFS, let the operating voltage be $V = x_v v_{max}$, where x_v is the scaling factor, $0 < x_v, \leq 1$. Similarly, for frequency, let the operating frequency be $f = x_f f_{max}$, with $0 < x_f \leq 1$. Finally, let the increased execution time be $T = x_t T_{min}$, with $x_t \geq 1$. We express energy as a function of the three scaling factors

$$\begin{aligned} E(x_f, x_v, x_t) &= (\alpha C(x_f f_{max})(x_v V_{max})^2 + I_0(x_v V_{max}) + P_c) x_t T_{min} \\ &= \left(x_f x_v^2 P_d + x_v P_s + P_c\right) x_t T_{min} \end{aligned}$$

We now normalize this by dividing by $P_d T_{min}$ to obtain the normalized energy consumption,

$$E_n(x_f, x_v, x_t) = \left(x_f x_v^2 + x_v R_s + R_c\right) x_t \qquad (3)$$

where

- R_s: ratio of static power with respect to dynamic power, and
- R_c: ratio of constant power with respect to dynamic power.

3.2 Relationship between Voltage and Frequency scale factors

Although voltage and frequency are linearly related, a few subtle issues arise when we precisely model their combined effect. The two scale factors are related as given below. The widely accepted formula is based on a study of recent processors, by a number of authors [16, 20, 30].

$$x_v = \frac{2}{3} x_f + \frac{1}{3} \qquad (4)$$

We use this to eliminate x_v in Equation 3 to obtain:

$$E_n(x_f, x_t) = \left(x_f \left(\frac{2}{3} x_f + \frac{1}{3}\right)^2 + \left(\frac{2}{3} x_f + \frac{1}{3}\right) R_s + R_c\right) x_t \qquad (5)$$

3.3 Properties of the Energy Model

For now, we let the slowdown factor, be $x_t = \frac{1}{x_f}$. For compute-bound programs, execution time scales directly proportional to scaling of frequency [14]. Since x_f is normalized, execution time for such programs can be expressed as $\frac{1}{x_f}$ (a more nuanced analysis is provided in Section 3.4).

Let us show some of the important properties of our model that give insights to how dynamic, static, and constant powers influence overall energy consumption.

Distributing $\frac{1}{x_f}$ and further expanding x_v^2 gives:

$$E_n(x_f) = \left(\frac{4}{9} x_f^2 + \frac{4}{9} x_f + \frac{1}{9}\right) + \left(\frac{2}{3} + \frac{1}{3} x_f^{-1}\right) R_s + R_c x_f^{-1} \qquad (6)$$

Taking the derivative of the above with respect to x_f yields:

$$\frac{dE_n}{dx_f}(x_f) = \left(\frac{8}{9} x_f + \frac{4}{9}\right) - \frac{1}{3} R_s x_f^{-2} - R_c x_f^{-2}$$

Further taking the second derivative with respect to x_f yields:

$$\frac{d^2E_n}{dx_f^2}(x_f) = \frac{8}{9} + \frac{2}{3}R_s x_f^{-3} + 2R_c x_f^{-3}$$

The second derivative is always positive if $R_s, R_c > 0$, which leads to:

- $\frac{dE_n}{dx_f} = 0$ will give the frequency with minimal energy consumption, and
- optimal frequency is less than 1 iff $\frac{dE_n}{dx_f} > 0$ when $x_f = 1$.

Based on the above, we compute the condition for optimal frequency being 1 (f_{max}):

$$\begin{aligned} \frac{dE_n}{dx_f}(1) &\leq 0 \\ \implies \left(\frac{8}{9} + \frac{4}{9}\right) - \left(\frac{1}{3}R_s + R_c\right) &\leq 0 \\ \implies 4 &\leq R_s + 3R_c \end{aligned}$$

When static power is 50% of the processor power, $R_s = 1$, we obtain $R_c \geq 1$ as the solution, indicating that if components of the system unaffected by DVFS consume about as much as the dynamic power of processors, then executing at the highest frequency level is the optimal choice.

One additional remark we make is that the static power also works against DVFS, and its degree is related to the fraction of voltage that do not scale along with frequency in Equation 4. This is because its linear power saving is cancelled by the linear increase in execution time.

3.4 Reducing the Impact on Execution Time

In the above, the influence of x_f on execution time was expressed as $x_t = \frac{1}{x_f}$. One may argue that many programs do not slow down as rapidly as frequency is scaled. Although accurate modeling of the impact on execution time is out of our scope, we provide additional analysis to show the implications of reduced impact on execution time. As mentioned earlier, the impact on execution time as a direct inverse of the normalized frequency may seem too steep for some programs that frequently access memory. In this section, we extend our model in Equation 3 and add a variable to control the speed degradation.

We use a variable x, $0 \leq x \leq 1$ and let $x_t = 1 + x(\frac{1}{x_f} - 1)$. The variable x controls the speed degradation as frequency is scaled in a linear fashion. At $x = 1$, $x_t = \frac{1}{x_f}$, which is what we used in the above, and at $x = 0$, $x_t = 1$, no degradation as frequency scales. We substitute x_t in Equation 3 to obtain:

$$E_n^x(x_f, x_v, x) = \left(x_f x_v^2 + x_v R_s + R_c\right)\left(1 + x(\frac{1}{x_f} - 1)\right)$$

To simplify our analysis, we write the energy as $E_n^x = E_n^A + E_n^B$, the sum of two different sub-functions:

$$\begin{aligned} E_n^A(x_f, x_v) &= \left(x_f x_v^2 + x_v R_s + R_c\right) \\ E_n^B(x_f, x_v, x) &= \left(x_f x_v^2 + x_v R_s + R_c\right)\left(\frac{x}{x_f} - x\right) \end{aligned}$$

The respective derivatives[1] after eliminating x_v with Equation 4 are:

$$\begin{aligned}
\frac{dE_n^A}{dx_f}(x_f) &= \left(\frac{12}{9}x_f^2 + \frac{8}{9}x_f + \frac{1}{9}\right) + \frac{2}{3}R_s \\
\frac{d^2E_n^A}{dx_f^2}(x_f) &= \left(\frac{24}{9}x_f + \frac{8}{9}\right) \\
\frac{dE_n^B}{dx_f}(x_f, x) &= x\left[\left(\frac{8}{9}x_f + \frac{4}{9}\right) - \frac{1}{3}R_s x_f^{-2} - R_c x_f^{-2}\right] \\
&\quad - x\left[\left(\frac{12}{9}x_f^2 + \frac{8}{9}x_f + \frac{1}{9}\right) + \frac{2}{3}R_s\right] \\
\frac{d^2E_n^B}{dx_f^2}(x_f, x) &= x\left(\frac{2}{3}R_s x_f^{-3} + 2R_c x_f^{-3} - \frac{24}{9}x_f\right)
\end{aligned}$$

We can again observe that the second derivative of $E_n^x(x_f, x)$, $\frac{d^2E_n^x}{dx_f^2}(x_f, x) = \frac{d^2E_n^A}{dx_f^2}(x_f) + \frac{d^2E_n^B}{dx_f^2}(x_f, x)$, is always positive if $R_s, R_c > 0$, $0 < x \leq 1$, and $0 \leq x_f \leq 1$. The second derivative also always positive if $R_s, R_c > 0$, $0 < x \leq 1$, and $0 \leq x_f \leq 1$. Thus, the optimal frequency is 1 (f_{max}) when:

$$\begin{aligned}
&\frac{dE_n^x}{dx_f}(1, x) && \leq 0 \\
\implies &\left(\frac{21}{9} + \frac{2}{3}R_s\right) + x\left(\frac{12}{9} - \frac{1}{3}R_s - R_c\right) - x\left(\frac{21}{9} + \frac{2}{3}R_s\right) && \leq 0 \\
\implies &\left(\frac{7}{3} + \frac{2}{3}R_s\right) - x\left(1 + R_s + R_c\right) && \leq 0
\end{aligned}$$

The above leads to the following remarks:

- As expected, the above indicates that as x decreases, which means as penalty on execution time with DVFS decreases, the inequality is less likely to be satisfied.
- Static power (R_s) work for DVFS when $x < \frac{2}{3}$. This is when the linear decrease in static power dissipation by DVFS starts to benefit overall energy consumption.
- With lower x, especially below $\frac{2}{3}$, much larger R_c will be required to satisfy the condition for f_{max} to be optimal.

The key implication is that as the program is less and less penalized by scaling the operating frequency, the ratio of constant power (R_c) to processor must become larger for the "go as fast as possible" strategy to hold, but the general property is unchanged.

When the program execution time is not dominated by processor speed, we can expect that other system components, such as the memory or network card, are stressed, and therefore ratio of processor power in the total system load to decrease [7, 17, 28].

Therefore, the behavior when degradation in speed is scaled is largely dependent on the application characteristics. When the x is small, it is likely that slowing the processor will be beneficial, since it is approaching memory-bound programs. For relatively large x, required R_c will become larger, but it is probable that going as fast as possible is still optimal. We also note that some of the recent machines have R_c much larger than 1 as we show in Section 4, further increasing the likelihood of this being the case.

3.5 Parallelism

So far, our analysis was completely independent of parallelism, although energy is intimately tied to parallelism. Indeed, the advent of multi-core and many-core processors was dictated by the needs of energy

[1]Derivations are not shown, as they are similar (but slightly more complicated) to the derivation from Equation 3.

efficiency. We now tie the results to parallelization. Our main message remains that energy efficiency is attained by optimizing for speed, and that using DVFS to slow down the application to achieve total energy gains will yield limited benefits at best. However, optimizing for speed is not necessarily the same as maximizing parallelism, and hence there are a few special considerations.

Let us first assume that the program is perfectly parallelizable on an N-core processor. Even in this optimistic situation, some of the components of the processor, like cache or other on-chip memory, are shared among the cores. In addition, regardless of the number of cores, the thermal envelope/budget is usually allocated for a processor chip, and therefore, R_c is computed for a processor chip, and not on a per-core basis. Therefore, if only one of the N cores is being used, it is likely that the constant power is greater than $\frac{1}{N}$. This leads to the conclusion that utilizing all the cores if possible, is the optimal strategy, unless parallel efficiency is low.

Now consider the situation where the program is not perfectly parallelizable. The question of whether or not to parallelize, and if so, how aggressively, is beyond the scope of this paper, and we do not attempt to answer it. Rather, let us suppose that the decision to use some number p out of the N cores has been made. Our analysis indicates that now, the best strategy is to make the program execute as fast as possible. Basically, if a processor cannot save energy by slowing down in sequential case, then trying to slow down processors in parallel case cannot save energy as well. Note that one may apply our analysis to each core, if per-core DVFS is supported, but the result remains the same.

The choice of the optimal p may involve a trade-off similar to that pointed out by Cho and Melhem [10, 11], but is also related to the application itself and how scalable its parallelization is. If the program has poor parallel speedup, and the decision is nevertheless to allocate an increasing number of processors to it, then some of the other, non-energy related issues (i.e., the response time of the program) are deemed to be important enough, to possibly override the gains of energy savings by using fewer cores. This means that any compiler (and possibly the programmer) should seek to provide the maximally scalable parallelization possible.

4 Trends in Recent Machines

In this section, we present trends in recent machines based on a survey ranging from desktop processors to Cray supercomputers. The goal of this section is to verify the observation based on previous studies that the constant power is around $\frac{1}{3}$ of the total power consumption under load, so that even if a significant fraction of the remaining $\frac{2}{3}$ is used by processor, $R_c \geq 1$ would still be true, satisfying the condition for f_{max} to be the optimal frequency for energy efficiency [7, 17, 27, 31].

For desktop and server processors, we show that, even with conservative estimates, ratio of constant power in the total system power under load is close to $\frac{1}{3}$. We also show that the ratio of constant power has been relatively constant over the last 5+ years. This is to be expected, since designers of different components of the system try and fit their component to the same thermal envelope as previous generations. Therefore, if the ratio of static power consumption increases in processor power, then R_c will also increase.

For Cray supercomputers, we present estimates of R_c for two recent machines, and show that they are highly likely to exceed 1, also satisfying the condition.

4.1 Sources of Constant Power

Let us first describe various sources of constant power we use to estimate the lower bound. Constant power is power consumed under high load that are not affected by DVFS. Although there may be some relationship, it is not closely related to idle power. Especially with recent architectures, where aggressive power-gating is performed, idle power is likely to be much less than the constant power.

4.1.1 Stand-By Memory Power Consumption

One of the sources of constant power consumption is stand-by memory power. Recent study show that a 4GB DDR3 memory consume around 4W in stand-by state [13]. Although memory can also be put into low-power states that consume less power, unless the program does not use memory at all, it cannot be put into low-power states for long under heavy load. Therefore, we count 1W per 1GB of memory as part of the constant power consumption.

4.1.2 Power Supply Unit

When a system draws power, alternate current must be transformed to direct current, and significant amount of power may be lost during this process. Efficiency varies greatly depending on the quality of Power Supply Unit (PSU) and load, and it is considered efficient if the efficiency is higher than 80% [1]. We assume 85% efficiency for commodity desktop machines, 90% for servers, and 95% for supercomputers.

4.1.3 Chipsets and Fans

Prior studies show that older chipsets consuming 20-30W, while some new designs reduce its consumption to 6W [12, 15, 28]. Fans also consume 10-15W when active [15, 28]. For the purpose of our estimation, we consider 20W per chip for processors with 45nm or older process, and 10W for 32nm and 22nm processors as constant power for both chipset and fan combined. We believe this to be a safe lower bound based on the numbers above.

4.2 Desktop and Server Processors

We have collected a number of power consumption measurements for desktop and server processors under heavy load. We show that the ratio of conservative estimate of constant power; the sum of memory stand-by power, efficiency loss by PSU, and estimated power consumption by chipsets and fans; is more than 30% in most cases. This means that even if most of the remaining power is used by the processor, the value of R_c will be around 1. Since there are other sources of power consumption that are not included, such as accesses to memory/HDD, and network cards, it is highly likely that R_c is well above 1 in most cases.

We collected total system power consumption measurements from AnandTech [2], an online hardware review site, for various desktop processors. They provide measurements under compute-intensive workload (x264 encoding), and many components are kept consistent across different processors (e.g., same memory and video card, but not motherboard). Since GPUs consume significant amount of idle power, we exclude GPU idle power (25W) from the measured power consumption (the benchmark does not use GPU). The data set contains 46 data points, with Intel and AMD processors from 2008 to 2012. All machines measured were equipped with 4GB of memory.

The data set for server processors are from published SPECpower_ssj2008 results [4]. The data set contains 255 data points, with Intel and AMD processors from 2007 to 2012; excluding results that are either for a system with multiple nodes, labeled non-compliant, or with imprecise processor name (i.e., only Xeon with out specifying which model). The benchmark models server applications with large number of user requests ("ssj" in the name stands for "Server Side Java"). The metric we use from the published results is Average Active Power (W) with 100% target load, where the target load is calibrated to be the maximum throughput of the server computed as part of the benchmark run. Due to the nature of the work load, 100% load does not necessarily mean 100% processor utilization.

Figure 1 shows individual data points and means for each year for both desktop and server processors. The means are more than 30% in all cases, where 79% of desktop processors, and 69% of server processors have more than 30% of constant power. This is an indication that the constant power is large enough such that R_c will exceed 1. Moreover, we emphasize that our estimate of constant power is conservative, and that actual constant power is likely to be higher than our estimate.

Also, the benchmark used for server processors stress both disk and memory to a great extent. In addition to our assumption that servers use more efficient PSU, the difference in work is another explanation that we can provide for the estimated constant power to consist lesser fraction of the total power in server machines.

4.3 Cray Supercomputers

We also show that R_c is likely to be higher than 1 for Cray supercomputers. Since precise power breakdown of supercomputers are not available, our analysis is based on specifications of recent Cray supercomputers,

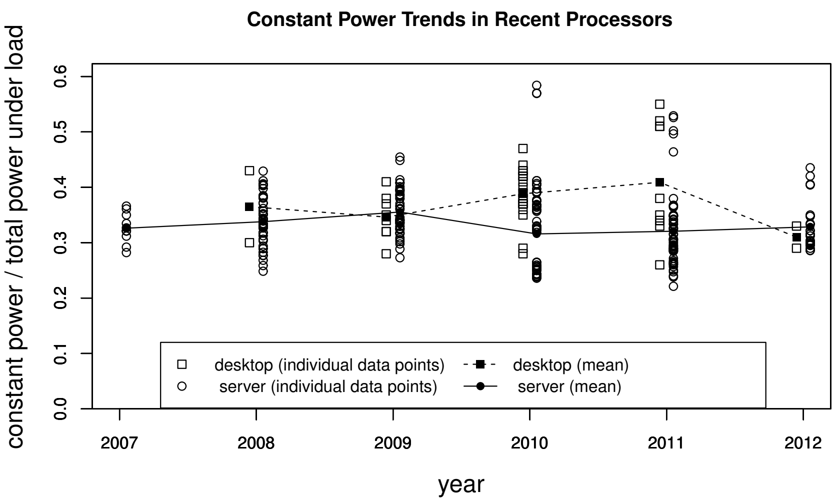

Figure 1: Constant power in recent machines. Constant power is more than 30% of the total power under load in many cases. Also, the fraction of estimated constant power in total system power is staying flat at the same level for both cases. Therefore, increase in static power will gradually increase R_c used in our model.

Table 1: Power consumption of a recent Cray supercomputer cabinet, based on manufacturer specifications [3]. All these machines use AMD Opteron processors.

Cray	nodes	chips per node	memory per node	total system power	TDP per chip	total chip TDP	memory stand-by power
XT5	96	2	16-32GB	32-42.7kW	95W	18.24kW	1.54-3.07kW
XT6	96	2	32-64GB	45-54.1kW	115W	22.08kW	3.07-6.14kW
XE6	96	2	32-128GB	45-54.1kW	115W	22.08kW	3.07-12.29kW

summarized in Table 1. Thermal Design Power (TDP) is the thermal envelope assigned to the Opteron processor used within Cray, and we use TDP as the upper bound on power dissipation by the processor.

We assume that Cray machines have PSU with 95% efficiency, and only 5% is counted towards constant power. We further assume that 10% of the power is used for cooling fans, based on the measurements from Cray XT5 in Oak Ridge National Laboratory [33].

In Table 2, we present estimates of R_c based on Table 1. Since the total system power is specified as range, we compute the percentages for 2 scenarios, highest and lowest total power. For each scenario, we estimate R_c where constant power is memory + PSU (5%) + cooling fans (10%) and dynamic power is 50% of the total chip TDP. The estimate on R_c is further divided into 2 cases, one with largest memory and the other with smallest memory. Finally, we compute how much additional constant power (as a percentage of the total power) is required to make $R_c \geq 1$, when smallest memory size, which alway give the smallest R_c, is used.

In the newer machines (XT6 and XE6), power consumed by processor as a fraction of total power is lower than the previous generation. Although the higher total power consumption is likely to be a combined effect of various factors, part of the increase may be attributed to increase in memory capacity. As a result, R_c is at least 0.89 for the new generation of machines.

For the earlier generation (XT5), the processor power can dominate up to 57% of the total power, and R_c can be as low as 0.69. However, the first scenario assuming lowest total system power is likely to be too strong, since we assume processor is running at its TDP. When we assume the highest total system power and largest memory, all of the three machines exhibit $R_c \geq 1$.

Moreover, we have not included power consumed by the network among nodes and other cabinets, which is likely to add another few percent to constant power. Thus, we conclude that current generations of Cray supercomputers is highly likely to have R_c greater than 1, satisfying the conditions for the optimal frequency to be f_{max}.

Table 2: Estimated R_c for Cray machines assuming different total system power. R_c is computed by assuming dynamic processor power is 50% of the TDP, constant power is the sum of memory stand-by power, cooling fan (10%), and PSU loss (5%). The last column is the fraction of total power that must be additionally included as constant power for $R_c \geq 1$ to hold, assuming the Cray was configured with smallest memory size.

Cray	assumed system power	percentage CPU TDP	percentage memory (min)	percentage memory (max)	R_c max memory	R_c min memory	additional P_s for $R_c \geq 1$
XT5	32kW	57%	5%	10%	0.86	0.69	8.70%
	42.7kW	43%	4%	7%	1.04	0.87	2.76%
XT6	45kW	49%	7%	14%	1.17	0.89	2.71%
	54.1kW	41%	6%	11%	1.29	1.01	0.00%
XE6	45kW	49%	7%	27%	1.72	0.89	2.71%
	54.1kW	41%	6%	23%	1.85	1.01	0.00%

5 Impact of DVFS for Memory

In the energy model we presented in Section 3, the power consumed to keep the memory in active state was included as part of constant system power consumption. When memory with DVFS support becomes widely available, power consumed by memory can no longer be considered as constant. In this section we extend our analysis handle this case.

David et al. [13] show that the power consumption of memory with respect to voltage scaling also has components that scale linearly and quadratically with V. The components that scale with V^2 also scale with frequency and thus the high-level power model for memory looks similar to that for the processor (Equation 1):

$$\begin{aligned} P_{mem} &= \alpha_m f V^2 + I_m V \\ &= P_{circuit} + P_{array} \end{aligned}$$

where we name the V^2 scaling component $P_{circuit}$ as it mostly comes from I/O circuitry of the memory, and V scaling component P_{array} as it comes from storage arrays. The authors also report that about 25% scales with V^2 and the remaining 75% scales with V [13].

5.1 Compute-Bound and Memory-Bound States

When there are two components, processor and memory, that can utilize DVFS, the frequency scalings can heavily influence whether a program (region) is compute-bound or memory-bound. In fact, any program may be both compute-bound *and* memory-bound, assuming that it has both some computation and memory accesses to perform. Instead of compute- and memory-bound *programs*, we use the notion of compute- and memory-bound phases or *states* of a given program to emphasize the transition.

The program is in compute-bound state if slowing down the memory does not affect the execution time. Similarly, the program is in memory-bound state if slowing down the processor does not change the execution time.

Let a program be in compute-bound state at some initial frequency settings for processor and memory. Then scaling the processor frequency down will keep the program compute-bound, since the compute power decreases as frequency is scaled. However, scaling down the memory frequency will eventually make the program memory-bound. As frequency is scaled, tolerable bandwidth is reduced [13], and at some point, the bandwidth requirement by the program can no longer be satisfied. Thus, the memory accesses will start having impact on overall execution time due to increased latency.

Conversely, consider a program in memory-bound state at the initial frequency. Then scaling down memory frequency will keep it memory-bound, and scaling the down the processor frequency will eventually reduce the compute power such that the amount of data being read cannot be processed fast enough, and thus increasing execution time.

With the exception of programs that do not use any memory, DVFS of processor or memory will, at some point, change a program from compute-bound to memory-bound and vice versa. Since when a program is compute-bound or memory-bound with respect to DVFS is highly dependent on the program characteristics, we use the above definition to keep our analysis independent of detailed program analysis.

5.2 Compute-I/O Balanced State

Given the above definition there are two obvious cases where DVFS can provide energy savings. One is scaling down memory frequency of compute-bound programs, and the other is scaling down the processor frequency of memory-bound programs. Since execution time is not traded off with power savings, frequency and voltage scaling will improve energy efficiency.

We define the program to be in compute-I/O balanced state when neither scaling the processor or memory give these obvious savings. Again, a program will eventually reach this state, but when is highly dependent on the program.

There are a number of techniques already developed for reaching the balanced state for memory-bound programs using processor DVFS [19, 20]. Once DVFS for memory become available, we can expect similar techniques to be developed to achieve the balanced state for compute-bound programs.

5.3 Energy Model with Memory DVFS

Once the balanced state has been reached, further improvement in energy efficiency involves trading off with speed. Our analysis in Section 3 focused on such cases when only processor DVFS was available. Now we extend the model to the case where we have the capability to independently scale memory frequency/voltage.

We take the base model without memory DVFS (Equation 2), and introduce $P_{circuit}$ and P_{array} to obtain:

$$E'_{base} = (P_d + P_s + P_{c'} + P_{circuit} + P_{array})T_{min}$$

The power components of memory were previously modeled partially as P_c (stand-by power) and were partially excluded from the model as constant energy (access cost).

One additional difference is that in the above equation the frequencies are set to the highest frequencies that make the program balanced. In other words, when both processor and memory are at the highest frequency, one of the two that has "slack"; i.e., the one that is not currently the bottleneck; can be scaled to reach the balanced state. This is the new starting point in analyzing the trade-off of power and speed.

Now, recall that $P_{circuit}$ and P_d both scale with fV^2, and that P_{array} and P_s both scale with V. Thus the terms can be merged to produce:

$$E_{merged} = (P_q + P_l + P_{c'})T_{min} \tag{7}$$

where

- P_q: maximum power consumption, at highest frequencies that make the program in balanced state, of the power component that scales quadratically with V,
- P_l: maximum power consumption, at highest frequencies that make the program in balanced state, of the power component that scales linearly with V,
- $P_{c'}$: constant power; as defined as P_c in Section 3, but now without including memory power,
- T_{min}: execution time at the maximum frequencies.

Thus, with this level of abstraction, the model essentially does not change from Section 3. However, the ratio of P_l, and $P_{c'}$ with respect to P_q will be different from R_s or R_c.

5.4 Influence on the Condition to "Race to Sleep"

Since a portion of the constant power has now been moved to P_q and P_l, the ratio of constant power will decrease. Thus, DVFS for trading off power with execution time may become viable again with the introduction of DVFS for memory. The power consumption benchmarks for desktop processors were compute-intensive, and thus the memory power consumption is on average only 3.1% of the total power under load. However, the numbers we have collected for server processors show that memory power consumption is on average 6.5% of the total power under load, ranging from 2 to 20%. Since the constant power was around 30% of the total power, memory power consumption and its DVFS capability can significantly change the picture.

We have shown in Section 3 that the optimal frequency is always greater than the maximum if $R_s+3R_c \geq 4$ holds. This relationship directly translates to the ratio of P_l and $P_{c'}$ with respect to P_q in Equation 7 with memory DVFS taken into account. Let R_l and $R_{c'}$ be the ratio of P_l and $P_{c'}$ with respect to P_q then we are interested in the values of R_l and $R_{c'}$ that satisfy $R_l + 3R_{c'} \geq 4$.

Since the power component that linearly scales with V is significantly larger than that which scales quadratically (75% of memory power scales linearly [13]), P_q is likely to be smaller than P_l. In addition, it has been projected and observed that the power density of a chip due to leakage power exhibits exponential growth following Moore's Law. Although new technologies have been developed to mitigate this exponential growth, the ratio of leakage power to total power consumption is increasing and is expected to continue increasing. As this ratio increases, the ratio of P_l with respect to P_q also increases, and as a result DVFS will become less and less effective.

To understand when "race to sleep" is optimal as P_l grows, we re-formulate the condition $R_l + 3R_{c'} \geq 4$ with two parameters l and c where

- $l = \frac{P_l}{P_q+P_l}$ is the fraction of dynamic power (non-constant power) that scales linearly; this corresponds to the aggregate of P_s in processors and P_{array} in memory, and
- $c = \frac{P_{c'}}{P_q+P_l+P_{c'}}$ is the fraction of total power consumption that is constant system power, such as PSU loss, chipset, and cooling fans.

The resulting function $e(l,c)$ is the following:

$$e(l,c) = \frac{l}{1-l} + 3\frac{c}{(1-l)(1-c)}$$

where

$$R_l = \frac{P_l}{P_q} = \frac{\frac{P_l}{P_q+P_l}}{\frac{P_q}{P_q+P_l}} = \frac{l}{1-l}$$

and

$$R_{c'} = \frac{P_{c'}}{P_q} = \frac{\frac{P_{c'}}{P_q+P_l+P_{c'}}}{\frac{P_q}{P_q+P_l+P_{c'}}} = \frac{\frac{P_{c'}}{P_q+P_l+P_{c'}}}{\frac{P_q}{P_q+P_l}\frac{P_q+P_l}{P_q+P_l+P_{c'}}} = \frac{c}{(1-l)(1-c)}$$

Based on power measurements we have collected, constant power is still around 30% for desktop processors, since they only had 4GB of memory, and is around 16% on average for servers, and 15% for Cray supercomputers.

Figure 2 shows the values of $e(l,c)$ evaluated for various range of l when $c = 0.3$ (desktop) and when $c = 0.15$ (server and Cray). The figure shows that when $l > 0.7$, or when linearly scaling component is larger than 70% of the sum of non-constant power, the optimal strategy becomes "race to sleep" for desktop machines. Similarly, server and Cray machines require around $l > 0.75$ for "race to sleep" to become optimal.

Since 75% of the memory power is linearly scaling, this suggests that the "race to sleep" becomes optimal when the fraction of static power in the total power consumed by a processor exceeds 70%. Although current processor may not have reached this line, we believe that it will in the near future.

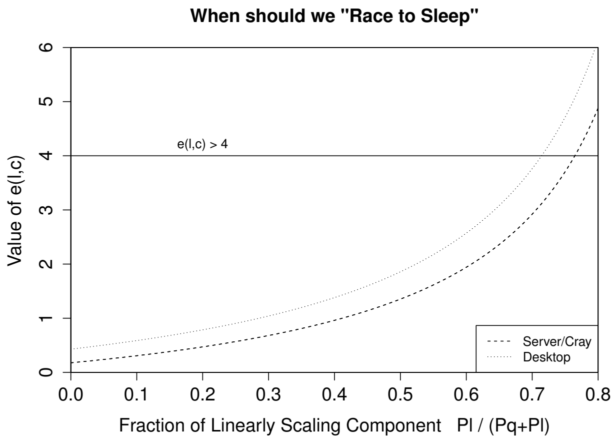

Figure 2: Plot of $e(l, c)$ evaluated with $c = 0.15$, $c = 0.3$ with l ranging from 0 to 0.8. As we saw in Section 3, we need around 70% of the non-constant power to scale linearly with respect to voltage for "race to sleep" to be optimal.

6 Related Work

Our work is definitely not the first to show that simply going as fast as possible can also be energy efficient. Cho and Melhem [10, 11] developed an analytical model of energy consumption under DVFS with multiple processors. They identify that there are cases where slowing the processor may not lead to reduced energy consumption, and that it is related to the fraction of total power unaffected by DVFS. We distinguish our work in three key aspects:

- (i) we tie our analysis results back to current machines, to verify that the constant power is significant enough for running as fast as possible to be optimal,
- (ii) our model separates the influence of dynamic and static power consumption of the processor, since the ratio of the two is also important, and
- (iii) we also extend the model to gain some insights for the case when execution time does not linearly degrade as frequency is scaled.

Dawson et al. [14] have empirically shown that constant power dominates (50% to 80%) total power by measuring power consumption of two processors. They also conclude that running as fast as possible and then going to sleep is likely to be energy efficient. However, they do not have a model of how much constant power is required for "racing-to-sleep" to be optimal. We complement their work by showing that it holds for large range of more recent processors, and also with Cray supercomputers. Our estimate of constant power is much more conservative, since we do not include GPU idle power, and assume much lower power consumption for chipsets. They have included 50W for chipset and GPU as constant power, whereas we only include 10-20W for chipset and cooling fan combined.

Our work focus on compute-bound (including compute-I/O balanced) applications. For memory-bound programs, where DVFS is considered more beneficial due to reduced degradation in speed as frequency is scaled, the work by Le Sueur and Heiser [25] had shown that the benefits of DVFS is also diminishing. One of the observations was also that the constant power of the full system, which has been overlooked in some energy optimizations based on DVFS, plays a significant roll in the overall picture.

When DVFS Can Help

In this paper, we show that trading off speed with energy with DVFS is not possible in most cases. However, there are a number of prior work that use DVFS to save energy without increasing the execution time [9, 18, 19, 20, 21, 22, 26]. The common idea behind these work is to utilize load imbalance across components of the system.

For example, memory-bound computations allow processors to be slowed down without affecting the execution time [19, 20]. Similarly, different components such as disks [18], or link interconnects [22, 26] that are not utilized all the time, can be turned off for energy efficiency. Load imbalance in parallel applications [9, 21] is another candidate for saving energy.

All of the above corresponds to techniques to bring programs into compute-I/O balanced state in our terms, and are still useful optimizations to improve energy efficiency. However, these techniques should be applied as a "last resort", after optimizing for speed. For instance, it does not make sense to make a program (more) memory-bound such that DVFS can be applied. Efficient access to memory will reduce the execution time and energy consumption. Similarly, it does not make sense to increase load imbalance of parallel applications such that DVFS can be applied. Developing methods for better load balancing will simultaneously improve speed and energy efficiency.

In addition to the above, recent processors employ sophisticated frequency/voltage scaling themselves, such as the Turbo Boost on Intel processors [5] These hardware controls are likely to be able to detect memory-bound regions of programs, and employ scaling themselves. Therefore, even such opportunities for energy saving by compilers may also be diminishing.

Another domain where DVFS may help is embedded systems, where you have much more flexibility than general purpose processors. Although the analysis in this paper remains the same, the significance of processor power with respect to the whole system power can vary widely between applications.

For some applications, the processor may be the dominant source of power usage, and hence DVFS is more effective. However, the opposite is also true. For example, the screen and the wireless card are the dominant power consumers in a smartphone, making DVFS even less interesting.

7 Conclusion

We have presented our analysis based on our high-level model of power consumption under DVFS. When the constant power in a system is comparable to the dynamic power consumption of the processor, using DVFS to trade speed with energy efficiency cannot be done, and it is best to run as fast as possible to completion.

We showed through a survey of number of recent machines that it is highly likely that most machines today fall under the condition where running as fast as possible wins in terms of energy.

Therefore, we confirm the "folklore" we have been hearing regarding energy optimization, and conclude that simply compiling for speed will also give better overall energy efficiency.

Our analysis is based on a high-level model, and it is possible that some class of problems can still benefit from DVFS. However, in this paper we have ignored the cost of changing DVFS states, and also assumed that arbitrary frequency/voltage can be selected. In practice, the cost of transition is not negligible, and available frequency/voltage configurations are limited, further limiting the applicability of DVFS.

Our result may seem negative, but from compilers' perspective, the problem has been made simpler. We can focus on speed, and the resulting code will be energy efficient. Until the time when the leakage power becomes a negligible component again, which is when the game entirely changes, invalidating many analyses including ours, compilers should focus on speed.

References

[1] 80plus power supplies. www.plugloadsolutions.com/80PlusPowerSupplies.aspx.

[2] Anandtech. www.anandtech.com.

[3] Cray products. www.cray.com/Products/Products.aspx.

[4] Specpower. Published at www.spec.org as of 6 May 2012. SPEC and the benchmark name SPECpower_ssj2008 are registered trademarks of the Standard Performance Evaluation Corporation. For more information about SPECpower_ssj2008, see www.spec.org/power_ssj2008/.

[5] Intel® Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors. White Paper, November 2008.

[6] Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, 2008.

[7] W.L. Bircher and L.K. John. Complete system power estimation: A trickle-down approach based on performance events. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software*, pages 158–168, 2007.

[8] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[9] Guangyu Chen, Konrad Malkowski, Mahmut Kandemir, and Padma Raghavan. Reducing power with performance constraints for parallel sparse applications. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, pages 8–pp, 2005.

[10] S. Cho and R. Melhem. Corollaries to Amdahl's law for energy. *IEEE Computer Architecture Letters*, 7(1):25–28, 2008.

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

[12] B.G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. An energy case for hybrid datacenters. *ACM SIGOPS Operating Systems Review*, 44(1):76–80, 2010.

[13] H. David, C. Fallin, E. Gorbatov, U.R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. *Memory*, 300:400, 2011.

[14] S. Dawson-Haggerty, A. Krioukov, and D.E. Culler. Power Optimization–a Reality Check. Technical report, Technical Report UCB/EECS-2009-140, EECS Department, University of California, Berkeley, 2009.

[15] X. Fan, W.D. Weber, and L.A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23, 2007.

[16] V.W. Freeh, N. Kappiah, D.K. Lowenthal, and T.K. Bletsch. Just-in-time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs. *Journal of Parallel and Distributed Computing*, 68(9):1175–1185, 2008.

[17] R. Ge, X. Feng, S. Song, H.C. Chang, D. Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, 2010.

[18] Taliver Heath, Eduardo Pinheiro, Jerry Hom, Ulrich Kremer, and Ricardo Bianchini. Application transformations for energy and performance-aware device management. In *Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques*, pages 121–130, 2002.

[19] C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1, 2005.

[20] C.H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, page 48, 2003.

[21] I. Kadayif, M. Kandemir, G. Chen, N. Vijaykrishnan, MJ Irwin, and A. Sivasubramaniam. Compiler-directed high-level energy estimation and optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(4):850, 2005.

[22] E.J. Kim, K.H. Yum, G.M. Link, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, M. Yousif, and C.R. Das. Energy optimization techniques in cluster interconnects. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 459–464, 2003.

[23] N.S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):75, 2003.

[24] Jonathan G. Koomey, Christian Belady, Michael Patterson, Anthony Santos, and Klaus-Dieter Lange. Assessing trends over time in performance, costs, and energy use for servers. Technical report, Lawrence Berkeley National Laboratory, Stanford University, Microsoft Corpotation, Intel Corporation, Hewlett-Packard, Coporation, 2009.

[25] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8, 2010.

[26] F. Li, G. Chen, and M. Kandemir. Compiler-directed voltage scaling on communication links for reducing power consumption. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, page 460, 2005.

[27] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. *Power-Aware Computer Systems*, pages 165–180, 2005.

[28] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: eliminating server idle power. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, pages 205–216, 2009.

[29] John S Seng and Dean M Tullsen. The effect of compiler optimizations on pentium 4 power consumption. In *Proceedings of the 7th Workshop on Interaction Between Compilers and Computer Architectures*, pages 51–56, 2003.

[30] A. Sinha and A.P. Chandrakasan. Jouletrack-a web based tool for software energy profiling. In *Proceedings of the 38th Design Automation Conference*, pages 220–225, 2001.

[31] B. Subramaniam and W. Feng. Understanding power measurement implications in the green500 list. In *Green Computing and Communications, 2010 IEEE/ACM International Conference on & International Conference on Cyber, Physical and Social Computing*, pages 245–251, 2010.

[32] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. Reducing power in high-performance microprocessors. In *Proceedings of the 35th Design Automation Conference*, page 737, 1998.

[33] T. Wenning and M. MacDonald. High performance computing data center metering protocol. *Federal Energy Management Program, US Department of Energy, Resources on Data Center Energy Efficiency*, 2010.