

Exploiting Morphable Microarchitectures for Saving Energy*

Michael G. Kirkpatrick Vincent W. Freeh Peter M. Kogge Robert J. Minerick

Department of Computer Science and Engineering

University of Notre Dame

Notre Dame, IN 46556

{mkirkpat,vin,kogge,rminerick}@nd.edu

August 22, 2001

Abstract

Much current work focuses on microarchitectures where power can be saved in various ways when the computing demands are less than peak. However, the techniques for managing these *energy gears* are relatively *ad hoc*. As part of the Morph project, this paper discusses a model for extracting a key time-varying parameter from real applications, the *time dilation* factor, and using that to set the available gears at run time. Some initial experimental results are discussed, with projections of what that might translate into in terms of overall energy savings for power-saving microarchitectures.

1 Introduction

Current *energy efficient* microprocessors start with complex designs optimized for peak performance, and then add energy conservation modes by relatively simple techniques such as idle modes, clock reduction, and voltage scaling. Some recent research into alternative microarchitectures are adding the potential for dynamically changing inherent energy/performance characteristics. Controlling these techniques, however, is today at best *ad hoc*. It is easy to determine what to do in periods of peak or zero performance needs, but what to do under intermediate loads, and how important it is to have accurate information about the dynamic situation, is largely an open issue.

In this paper, we focus on one particular parameter, *time dilation*, which measures by how much a particular time slice of computation could be slowed down without interfering with future

*This work was done as part of the Morph project at the University of Notre Dame, and partially funded by DARPA under the Power Aware Computing and Communication (PAC/C) program under contract FC 306020020525.

scheduling events. Knowing it allows a runtime to set all the available *energy gears* low enough to finish *just in time* for the next event.

Section 2 reviews what energy gears are available. Section 3 constructs an analytical model for discussing dilation. Section 4 describes our experiments and results. Sections 5 and 6 discuss related and future work.

2 Energy in Microarchitectures

The classical power equation is of the sum of the dynamic and static power, with the dynamic term (the classical $\frac{1}{2}CfV^2$) typically dominant. Alternatively, however, a CPU's dynamic power may be expressed as the energy expended per clock cycle (EPC) times the clock in cycles per second. This alternate is valuable because it mirrors the performance relationship: instructions per second equals instructions retired per cycle (IPC) times clock.

Modern microarchitectures have two major departure points. For high performance needs, superscalar, deeply pipelined, heavily cached designs provide high IPC numbers at very high EPC values. For “low power” applications, simpler single issue moderately pipelined designs provide IPCs that are a fraction of that for the superscalar designs, but at EPCs which are literally orders of magnitude less. In [ZK00, ZK01], we show evidence that issue width, w , is a distinguishing parameter between such designs, with $w = 1$ corresponding to low power configurations and $w = 4$ to 6 to the high performance ones. To a first approximation, IPC grows as $w^{0.5}$, while EPC can grow as w^2 [ZK98]. Thus EPC can grow as fast as IPC^4 .

Any system, especially embedded, needs to be designed so that its peak performance meets peak demand. The problem for low power designs is that they typically cannot meet the peak performance demanded by certain applications. We now pay a huge energy penalty (high EPC) during the large percent of time when low performance applications are in effect.

The first new features to add to such designs are idle, snooze, or sleep modes where the whole CPU or large portions of it are turned off when not needed. This reduces power by the ratio of idle time to busy time, but during those busy times, the machine is still running at peak EPC, IPC, and clock. Adding clock management by itself does little—the EPC and IPC are the same. We are only exchanging idle time for longer cycles. Adding voltage scaling to the mix now has a real effect—the EPC decreases as the square of decrease in voltage. The trick now is to select the two “gears:” clock and voltage, to minimize power for an application while still meeting timing constraints. This is the state of the art as exhibited by Intel's Speedstep, Transmeta's Longrun, and AMD's PowerNow! They are not the only ones, however.

In [KFG⁺00], we propose a microarchitecture where the w may be changed dynamically. Using the above approximations, deep reductions in w reduce IPC only slightly, while EPC decreases by a square factor. Thus, under non-peak loads, in combination with clock and voltage scaling we can now reduce the IPC down to the point where there is just enough performance to meet the load, and reap a huge decrease in the $EPC \times clock$ product. The run time can now manipulate clock, voltage, and w as “gears.”

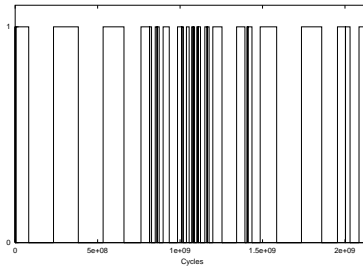


Figure 1: Execution burst for ordinary workstation load.

3 Execution Burst Model

On a modern microprocessor and runtime, applications go through a series of *execution bursts*: where the microprocessor alternately executes and idles. Each burst consists of a single loaded period and a single unloaded or idle period. A heavily loaded machine rarely enters idles, in which case the loaded component can be very long. Similarly, an unloaded machine has long idle periods. Figure 1 shows the execution bursts during ordinary workstation activity (editing, viewing, etc.). The x-axis is the time in cycles; the y-axis is execution: either 1 or 0 (on or off). Initially, there are long *on* periods and long *idle* periods, with burst intervals on the order of 100s of million instructions. In the middle, the trace rapidly switches between *on* and *idle*—it appears as a solid vertical line. The execution burst intervals are short, as small as a few thousand cycles.

Although this model is simplistic, it does not lack for generality or detail because it captures perfectly what happens in the runtime. The scheduler dispatches a process, if the ready queue is not empty. If the ready queue is empty, the runtime enters the idle period (called the *idle loop* in Linux). If the runtime is conserving power and the microarchitecture supports it, the runtime puts the machine into a reduced-energy state during the idle period.

It is trivial for a runtime to support two modes: running and idle. This paper considers the energy consumed in such a bi-modal runtime as the *baseline*. For a single execution burst, the baseline energy usage is $P_{on}t_{on} + P_{idle}t_{idle}$. $P_{on} = P \times 1$ (or full power) and $P_{idle} = P\lambda$, where λ is the relative energy used at idle. (P is full power.) The lengths of each period and the load average (\hat{L}) are related as follows: $t_{on} = \hat{L}t$ and $t_{idle} = (1 - \hat{L})t$, where $t = t_{on} + t_{idle}$. The baseline power becomes: $P_{baseline} = P[\hat{L} + \lambda(1 - \hat{L})]$.

As explained in Section 2, in the Morph architecture the energy required grows as the performance demand increases. Through a combination of scaling the frequency, voltage, and issue width (w or “gear”), the Morph architecture has an energy-performance profile that is approximately: $E = kM^\alpha$, where M is the performance setting and k is a constant [KFG⁺00]. Thus, energy grows with the performance setting raised to an exponent. This simple profile is fairly accurate except where M is small. In this region, the energy usage due to performance is so low that other effects start to dominate. The range for α in the Morph architecture is predicted to be 2 to 4 [KFG⁺00].

Because it is more power efficient to execute for a longer time at a reduced performance level than to execute at full performance and idle for some time, the optimal energy consumption occurs

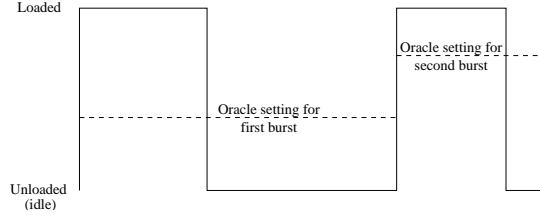


Figure 2: Optimal power setting.

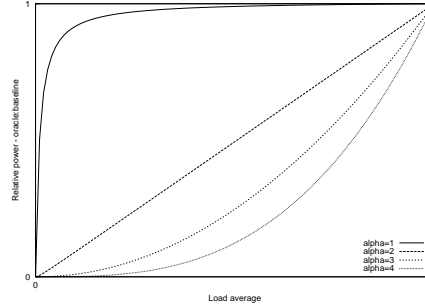


Figure 3: Relative power usage, $\Pi(\hat{L}, \alpha)$, $\lambda = 0.01$.

with a performance setting equal to the load average for the *upcoming* execution burst, see Figure 2. This performance setting provides just enough performance to meet the upcoming work with no idle period. Assuming infinite scaling of the microarchitecture performance, the optimal or *oracle* energy consumption is: $P(\hat{L})t$. Using the idealized energy profile of the Morph architecture, the oracle power becomes: $P_{oracle} = P(\hat{L}) = P\hat{L}^\alpha$, where $2 \leq \alpha \leq 4$.

The relative power savings is:

$$\Pi(\hat{L}, \alpha) = (P_{oracle}/P_{baseline}) = \hat{L}^\alpha / [\hat{L} + \lambda(1 - \hat{L})].$$

For energy efficient microarchitectures, λ is small. Therefore, the relative power saving tends toward $\Pi(\hat{L}, \alpha) = \hat{L}^{\alpha-1}$. Figure 3 show the relative power of the oracle to the baseline for several α s. If the energy profile of the microarchitecture is linear ($\alpha = 1$), then there is little gain over the baseline energy. But as α increases the potential saving is huge.

The *time dilation* factor is how much one can stretch the *on* period of an execution burst without affecting the subsequent burst. Under the best possible conditions, the time dilation factor is $\delta = t/t_{on} = 1/\hat{L}$. The load ranges from 0 to 1; therefore, dilation ranges from 1 to infinity. However, in reality there are diminishing returns, so a dilation of greater than 100 is unlikely to reap any benefits. Because dilation is the reciprocal of load, as δ increases, the energy consumption decreases. Intuitively this means the more one can expand the time to complete some work, the less energy consumed.

This model only considers dilation within a single execution burst. Although one can dilate through more than one burst and theoretically achieve lower energy, it has limited utility. Moreover,

there are many complications introduced, and it does not appear to be productive for a dynamic mechanism used by the runtime.

The model presented here is idealized in several ways.

- The model assumes infinite scaling of performance. In actuality, the Morph architecture has discrete performance settings or “gears.”
- There are several effects that occur when operating at very low power levels that are not considered. Therefore, the energy cost for very low performance settings is unlikely to follow the \hat{L}^α curve. (For this reason, experiments in Section 4 avoid this region.)
- The model does not account for the effect of slowing down the execution stream. In reality, slowing the machine could easily change the subsequent load and the shape of execution bursts occurring later. For example, if a read is delayed, the work that occurs when the read is satisfied is also delayed.

The model will be refined as future work delves into areas where the model’s limitations cause problems. However, the simple model adequately serves the purposes of this initial study into the potential energy savings.

4 Initial Experimental Results

The primary purpose of this study is to determine the potential energy savings of time dilation of execution bursts. This information would then be used to develop a runtime mechanism that selects the appropriate gear settings. This section describes the nature of the data gathered and the manner in which it was captured. It then presents an analysis of this information and characteristics of it which were observed.

4.1 The Test System

The test system is a 300-MHz G3-based Power Macintosh running Linux. A modified Linux kernel records the periods in the idle loop. It records the time the idle loop begins and the time it stops. Time is expressed in terms of machine cycles, which are obtained from hardware counters on the microprocessor. Because this microprocessor runs at a fixed 300-MHz clock and the modified kernel captures cycles elapsed (not instructions retired), the trace data presents true elapsed time at maximum precision.

The modified kernel also records pertinent system information at each scheduling event. A scheduling event occurs whenever a process is started or stopped. This includes entering and exiting the idle loop, as well as context switching to another process. Some of the other information that can be obtained is: instructions retired, process queue length, TLB misses, and page faults. This other information is not used in the initial study, which is the subject of this paper.

The overhead of capturing these traces is negligible. Trace data is written to a kernel buffer, which is dumped to a file *after* the test run is completed. Therefore, no I/O is performed during the trace. The consequence of using an in kernel trace buffer is that it limits the amount of data that

	gcc	gzip	PostMark	wget
Len(B)	3.14	1.72	5.99	1.26
Bursts	4	50	10	97
E_b/t	0.996	0.772	0.787	0.708
E_o/t	0.993	0.772	0.783	0.695
E_o/E_b	0.997	0.999	0.995	0.982
$\hat{\delta}$	1.00	1.30	1.27	1.42

Table 1: Real loads.

can be captured. Additionally, the trace buffer should not be too large because the performance of the kernel changes when large memory buffers are added. In this study, we used a 64KB buffer, which is sufficient for traces up to 10–30 seconds.

The primary analysis tool is the energy profiler. This software takes as input (1) a trace of execution bursts and (2) a profile of energy versus load. The profiler calculates the energy consumed for the trace and a few related statistics. The energy consumed is the sum of the energy at each execution burst in the trace.

The baseline energy is calculated as: $E_i = P(1)t_{on}^i + P(0)t_{idle}^i$. That is, it is the energy cost at full load (1) times the length of the *on* period plus the energy consumed in the idle period. The total baseline energy for the trace is $\sum E_i$. The oracle energy is calculated as: $\hat{E}_i = P(\frac{t_{on}^i}{t_{on}^i + t_{idle}^i})(t_{on}^i + t_{idle}^i)$. That is, it is the energy cost at a reduced load (the load average) for the entire burst. The total oracle energy is obviously $\sum \hat{E}_i$.

The energy profile used in this analysis is essentially $E(x) = x^\alpha$. The notable exception is that the minimum energy output of the profile is E_{idle} . Unless stated otherwise, the discussions below use $\alpha = 2$ and $E_{idle} = 0.01$. An α of 2 is easily obtained; we expect the Morph architecture to have an α in excess of 6. An idle state that uses 1% the energy of the full state represents a well-tuned micro-architecture. These values are very favorable to the baseline energy. Because the bias is against the oracle, the results presented are pessimistic and we expect to see greater savings with the Morph architecture.

Additionally, we built a synthetic load generator that periodically executes work. The load generator is parameterized for length of period and amount of work, including a random amount. Using the load generator, we are able to easily create various kinds of load profiles on the system and capture traces. There are advantages to using a synthesized load. First, one can more easily explore the space of different loads. Second, the experiments are more easily repeated, adding confidence to the results.

4.2 Analysis

The energy profiler uses the energy model discussed in Section 3. This section applies the model to actual traces obtained on the test system. Traces were taken during several different types of loads and are summarized here. Table 1 shows results from several real loads. These are a Linux kernel compile using gcc, gzip file compression, a file system benchmark (PostMark), and a download of

	mp3	tar	wkstn
Len (B)	2.89	9.96	2.18
Bursts	645	103	418
E_b/t	0.061	0.091	0.265
E_o/t	0.012	0.053	0.238
E_o/E_b	0.190	0.582	0.898
$\hat{\delta}$	13.30	12.23	3.88

Table 2: Real loads.

	Load 1	Load 2	Load 3	Load 4	Load 5
Len (B)	3.31	3.34	3.32	3.45	3.43
Bursts	121	112	110	109	108
E_b/t	0.177	0.341	0.508	0.648	0.813
E_o/t	0.042	0.128	0.285	0.477	0.744
E_o/E_b	0.236	.377	0.561	0.736	.915
$\hat{\delta}$	5.91	2.99	1.99	1.55	1.23

Table 3: Various loads at 100ms intervals.

a file (wget). The table shows the length of the run in billions of cycles, the number of execution bursts, the average baseline energy per cycle (E_b/t), the average energy per cycle for the oracle (E_o/t), the potential for energy savings (E_o/E_b), and the average dilation factor ($\hat{\delta}$). The dilation factor represents the degree to which the load of an execution burst can be “spilled” into the idle period following it where, the larger the number, the greater the potential for spreading out the load. Conversely, E_o/E_b represents the amount of energy required by the oracle relative to the baseline. Here a smaller number is better. The baseline and oracle energies per cycle give some measure of comparison between columns.

There is almost no improvement with the oracle for the loads in Table 1. This is because these loads are very heavily loaded. The average dilation is close to 1, which means there is little opportunity to spread the load. The execution burst are very long, up to 100s of millions of instructions.

Table 2 shows loads with a greater potential for energy savings than those in Table 1. The loads are MP3 decode, tar file decompression, and a workstation load (as shown in Figure 1). The relative energy used by the oracle ranges from 19% to 90%.

Table 3 lists five loads produced by the load generator described above. The loads are periodic, at 100 millisecond intervals, with increasing amount of load. This table shows that, as the load on the system is increased, the potential for saving energy decreases. This is because reduced idle periods are a consequence of a greater load and smaller idle periods means less opportunity for the load of an execution burst to spread out.

Table 4 shows another set of periodic loads in which the load is more or less constant and frequency changes. The columns are labeled with frequency, in execution intervals per second. As

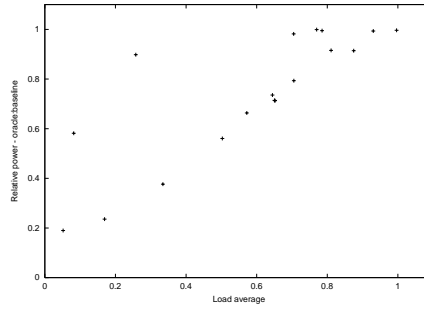


Figure 4: Scatter plot: Load average versus relative power usage.

	5 Hz	3.3 Hz	2.5 Hz	1.25 Hz
Len (B)	6.42	6.78	6.46	7.65
Bursts	134	84	70	40
E_b/t	0.654	0.576	0.656	0.708
E_o/t	0.467	0.382	0.467	0.562
E_o/E_b	0.714	0.663	0.713	0.793
$\hat{\delta}$	1.54	1.78	1.53	1.42

Table 4: Fixed load at various frequencies.

the frequency was changed, the amount of work to be performed during each burst was adjusted proportionately. This table shows that the energy savings is nearly the same.

Figure 4 shows a scatter plot of load average versus relative energy savings ($E_{oracle}/E_{baseline}$). Most of the loads shown fall on the diagonal line, which corresponds to the $\alpha = 2$ line in Figure 3 on page 4. This is not surprising because this analysis uses an energy profile with $\alpha = 2$. Several values, however, lie above the line. Each of these points corresponds to a real load. The anomalous loads are not regular. Consequently, some execution bursts are not dilated as well as others. With the periodic synthetic loads, each execution burst is more or less equally dilated. Some real loads fall on the diagonal, in these cases the loads are more regular.

The loads tested exhibit a wide range of potential for energy savings. In general, smaller average loads (larger dilation factors) show a greater potential. This is somewhat to be expected since a smaller load suggests longer idle periods over which the load can be averaged. This is seen in the data from the synthetic loads shown in Tables 3 and 4 and in the placement of points for those runs along the diagonal in Figure 4. Attempting to draw a direct correlation between dilation factor and energy savings, however, can be misleading. The dilation factor for a given execution burst depends on the size of the idle period following it as well as on the load itself. It possible, then, for a run which, overall, has a low average load, to nevertheless not offer much in terms of energy savings if the idle periods following the most significant load periods are relatively short. This is more apparent in the real loads presented above and explains the placement of points above the diagonal for those runs in Figure 4.

5 Related Work

Several microprocessors use power saving techniques to increase the battery life of portable computers. Transmeta's Crusoe processor [Kla00, Hal00] can dynamically adjust both clock speed and voltage. It does so very quickly, allowing software to continuously monitor the demands on the processor, thus reducing power consumption and increasing battery life. Intel's SpeedStep [Int00] technology offers two core frequencies for the Mobile Intel Pentium III processor, using real-time dynamic switching between two performance modes. Similarly, AMD's PowerNow! [AMD] allows the selection of three modes to decrease power usage: high performance mode, battery saver mode, and an automatic mode in which voltage and frequency are dynamically controlled in response to processor load. Merchant et al [MMSS96] examine the use of a variable-speed processor that can be slowed to a lower operating speed for the purpose of reducing temperature.

A common approach to power management strategies are the prediction of idle patterns in CPU usage. Several projects focus on scheduling algorithms to reduce CPU power usage. Manzak and Chakrabarti [MC00] developed a polynomial time scheduling algorithm that uses the relationship between the operating voltages for the minimum energy assignment. Bellosa [Bel99] uses an OS-directed power management approach that adds the clock speed to the runtime context of a thread, which is then factored into scheduling decisions. There are various other scheduling algorithms for taking advantage of reduced clock-speed, and it has been shown that by adjusting the clock speed at a fine grain, substantial CPU energy can be saved with a limited impact on performance [WWS94].

Similarly, allocation of other resources in the operating system can have a dramatic effect on power usage. Vahdat [VLE00] took a broad approach to the effects of OS controlled power conservation, and found that revisiting techniques for application/OS interaction, memory allocation, resource protection and allocation, and communication may have significant effects on energy consumption of future computers. Wilkes [Wil92] focuses on determining the proper time to spin down a hard disk, while [LT95] focuses specifically on memory allocation techniques to maximize simultaneous data transfers.

6 Conclusions and Future Work

Conclusions This paper presents a model for determining the energy savings. This simplistic model is useful for initial experiments. The model shows that the greatest potential for savings is on a system that is about from one-half to two-thirds loaded. Interestingly, this is approximately the optimal load for rate-monotonic scheduling according to [LL73]—a good match for embedded systems.

Additionally, this paper discusses the capturing of traces of programs run in the Linux operating system. The capturing is performed by a modified kernel. The overhead is minimal and the information is precise.

Lastly, this paper analyzes a few loads. It tries to draw conclusions as to what loads have the greatest potential for energy savings. It shows that the amount of load on a system effects the energy savings. Additionally, it has shown that the extent of the idle periods is important. Highly fragmented loads (without long periods of idle) are less able to dilate load bursts.

Future Work There are several improvements that are in store for this work. The model is too simple. As the energy versus load profile of the Morph architecture comes into focus, the model must be updated. This will include the addition of discrete energy steps, as opposed to the infinitely scalable profile used in the current model. Second, the low load modeling is not accurate. Again, as the understanding of the Morph architecture grows, this area of the energy profile will be addressed.

The analysis was primarily concerned with finding the potential. There are some additional tests in this vein that can be performed, such as how periodic loads interact. Additionally, longer running traces should be explored. There should be further analysis of real work loads. Lastly, plans are in place to analysis applications based on the planetary rover.

References

- [AMD] AMD. Amd powernow!* technology overview.
- [Bel99] Frank Bellosa. Os-directed throttling of processor activity for dynamic power management. Technical report, University of Elangen-Nurnberg, 1999.
- [Hal00] Tom R. Halfill. Transmeta breaks x86 low-power barrier. *Microprocessor Report*, February 2000.
- [Int00] Intel. Pentium iii pocessor mobile module:mobile module connector 2 (mmc-2) featuring intel speedstep technology datasheet, 2000.
- [KFG⁺00] Peter M. Kogge, Vincent W. Freeh, Kanad Ghose, Nikzad (Benny) Toomarian, and Nazeeh Arank. Morph: Adding an energy gear to a high performance microarchitecture for embedded applications. In *Kool Chips Workshop*, pages 1–10, Monterey, CA, December 2000.
- [Kla00] Alexander Klaiber. The technology behind crusoe processors, January 2000.
- [LL73] C.L. Liu and J.W. Leyland. Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of the ACM*, pages 46–61, January 1973.
- [LT95] M. Lee and V. Tiwari. A memory allocation technique for low-energy embedded dsp software. In *Proc. Symp. Low Power Electronics*, pages 24–25, San Jose, CA, October 1995.
- [MC00] A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In *Proc. of the International Conference on Acoustics, Speech and Signal Processing*, June 2000.
- [MMSS96] Arif Merchant, Benjamin Melamed, Eugen Schenfeld, and Bhaskar Sengupta. Analysis of a control mechanism for a variable speed processor. In *IEEE Transactions on Computers*, pages 793–801, July 1996.
- [VLE00] A. Vahdat, A. Lebeck, and C. Ellis. Every joule is precious: The case for revisiting operating system design for energy efficiency. In *SIGOPS European Workshop*, September 2000.
- [Wil92] John Wilkes. Predictive power conservation, February 1992.
- [WWS94] M. Weiser, B. Welch, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. Symposium on Operating System Design and Implementation*, pages 13–23, 1994.
- [ZK98] Victor Zyuban and Peter M. Kogge. The energy complexity of register files. In *Int. Symp. on Low-Power Electronics and Design*, pages 305–310, Monterey, CA, August 1998.
- [ZK00] Victor Zyuban and Peter M. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Int. Symp. on Low Power Electronics and Design*, pages 84–89, Rapallo, Italy, July 2000.

- [ZK01] Victor Zyuban and Peter M. Kogge. Inherently lower-power high-performance superscalar architecture. To appear in *IEEE Trans. on Computers.*, 2001.