On the Limits of Leakage Power Reduction in Caches

Yan Meng, Timothy Sherwood and Ryan Kastner
University of California, Santa Barbara
Santa Barbara, CA 93106-9560, USA
yanmeng@engr.ucsb.edu; sherwood@cs.ucsb.edu; kastner@ece.ucsb.edu

Abstract

If current technology scaling trends hold, leakage power dissipation will soon become the dominant source of power consumption. Caches, due to the fact that they account for the largest fraction of on-chip transistors in most modern processors, are a primary candidate for attacking the leakage problem. While there has been a flurry of research in this area over the last several years, a major question remains unanswered. What is the total potential of existing architectural and circuit techniques to address this important design concern? In this paper, we explore the limits in which existing circuit and architecture technologies may address this growing problem. We find that by using perfect knowledge of the address trace to carefully apply sleep and drowsy modes, the total leakage power from the instruction cache may be reduced to mere 3.6% of the unoptimized case, and the total from the data cache reduced to only 0.9%. We also present a complete parameterized model to determine the optimal leakage savings while the implementation technology changes over time. We futher suggest how such limits might be approached using a form of prefetching for low power.

1. Introduction

Power dissipation has become a major concern to those designing processors for high performance desktops, servers, and battery-operated portable devices. Higher energy dissipation requires more expensive packaging and cooling technology, which in turn increases cost and decreases system reliability. There are fundamentally two ways in which power can be dissipated: either dynamically (due to switching activity), or statically (which is mainly due to leakage in the gates). If current technology scaling trends hold [5], leakage will soon become the dominant source of power consumption, and as such new techniques are needed to battle this growing problem.

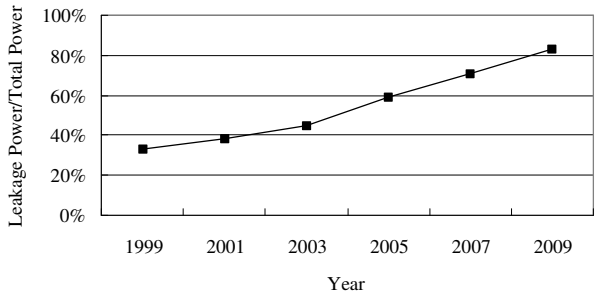


Figure 1. Projected leakage power consumption as a fraction of the total power consumption according to the International Technology Roadmap for Semiconductors [5].

The problem of leakage stems from the need for a tradeoff between dynamic power and performance. One of the most effective ways of reducing the dynamic energy dissipation is to scale down the transistor supply voltage. To maintain high switching speed under reduced voltages, the threshold voltage must be also scaled down accordingly. As the threshold voltage drops, it is easier for current to leak through the transistor resulting in significant leakage energy dissipation. New technologies targeted at reducing dynamic power and increasing performance, such as low threshold voltage and gate oxide scaling, further increase the relative importance of leakage power [5] (Figure 1).

In modern processors, a large and growing fraction of the total on-chip area, and an even larger fraction of the total number of transistors, is consumed by caches and local memory. Because they account for such a significant portion of the total chip real estate, caches provide a healthy-sized target for designers to try circuit and architectural optimizations with the goal of reducing leakage power. The central idea behind most of these techniques is to exploit some form of temporal locality. By putting infrequently or unused cache lines into low leakage mode, much of the power will

be reduced. By keeping frequently accessed cache lines active, total performance will not be reduced significantly.

Though there are several circuit techniques and management schemes concerning how and *when* to turn on or off individual cache lines, little work has been done to explore the *limit* of how well such techniques can work. What is the best we could hope to do with a given low power technology? The primary goal of this paper is to explore these limits under different architectural and design assumptions in the hope of guiding research effort on leakage power in much the way that Belady's MIN algorithm [1] helps (and continues to help) in the study of replacement policies.

There has been lots of work on leakage power reduction already, and any proposed methods for calculating the limits of their effectiveness must be both general enough to capture a variety of techniques, yet specific enough to provide useful bounds. Our methods capture both state-preserving and state-destroying techniques, and additionally we show how to optimally combine two such techniques into a hybrid scheme. We show that, given perfect knowledge of the future address trace, there exists a break-even point between Drowsy and Gated-V_{dd}. If the same cache line is accessed twice in an interval of time less than or equal to this break-even point then Drowsy mode should be used. If the same cache line is not used again within an amount of time greater than the break-even point then more power can be saved by turning off the cache line using Gated-V_{dd}. If these timings are known, then an optimal policy can be achieved.

Clearly perfect knowledge of the future trace is not always known, but it serves several purposes. First it provides an important bound. No management method will be able to beat our power reduction scheme under the given circuit assumptions. Second, it demonstrates that there is still a great deal of potential for policy decisions (when to turn a cache line on or off) to significantly reduce leakage power. Finally, while perfect knowledge of future references cannot be known, it can often times be approximated by architecture techniques such as address prediction or *prefetching*. We explore this relationship between prefetching and low power and show the counter intuitive result that prefetching has the potential to lead to lower power. In particular, our paper shows:

- We relate the potential savings that can be obtained from Drowsy and Gated-V_{dd} techniques, under various assumptions for both the instruction and data caches.
- We show that with oracle knowledge of future accesses, a simple optimal power management scheme can be derived from a small set of circuit parameters.
- In addition to showing the optimal leakage savings on a set of implementation parameters, we present a parameterized model to determine the optimal leakage savings while the implementation technologies change over time.
- We show that while both schemes are useful on their own, when combined, we can push the upper bounds of the leakage power savings to 96.4% for the instruction cache and 99.1% for the data cache with the 70nm implementation technology.
- Combining both the next-line and the Stride-based prefetching techniques, it is possible to approach the upper bound of the data cache (savings of 92.4%), which is about 7% better than turning off cache lines after 10K cycles.

The rest of the paper is organized as follows. We review related work and motivate our limit study in Section 2. In Section 3, we propose our method for combining the Gated-V_{dd} method and the drowsy method. We also explore the limit of leakage power saving that we can potentially achieve using our hybrid scheme. A model which parameterizes all the individual assumptions is also presented. Section 4 describes our simulation setup and the benchmarks in our study, and shows the results of our empirical study in exploring the upper bounds. In Section 5, we present the prefetching techniques employed in our method and experimental results. We offer concluding remarks in Section 6.

2. Circuits and Architectures of Reduced Cache Leakage Power

In order to derive a useful limit for leakage power reduction in caches, we must first begin with a discussion of those related technologies so that our model will be grounded in reality. In this section we review several circuit techniques, and develop the general ideas of our approach.

Leakage power comes from transistors that are simply left on, and the easiest way to think about reducing the amount of the consumed leakage power is to "turn off" those transistors that are not needed. While this is the easiest to think about, it is by no means the easiest to implement. One such approach, Gated-V_{dd} [12], attempts to solve this problem by reducing leakage through the use of a high threshold sleep transistor to break the connection from the supply voltage V_{dd}. This leakage reduction technique is often called *sleep mode*, and this is the naming convention that we use here. While efficient in saving leakage, *sleep mode does not preserve the state of the data*. When a cache line is needed again after it has been put to sleep, it must be refetched from lower levels of the memory hierarchy. This refetch is essentially an extra cache miss, and this process can take many cycles depending on the memory hierarchy, architectural assumptions, etc.

A different way of saving leakage power in the caches is to make use of *multiple supply voltages*. When the cache line is left fully on, it will dissipate too much leakage power. If V_{dd} is fully gated, it will use very little power, but the data is lost. A compromise is to use a *lower* supply voltage when

data is not needed for a while. This will reduce the leakage power without losing the data. The trade-off is that, while data will be preserved at this low supply voltage, it cannot be accessed while in this state. Thus there is a small wakeup time associated with changing from the lower voltage up to V_{dd} (hence the name "drowsy"). Drowsy mode does not fully turn off the memory, and thus does not reduce the leakage power as much as Gated-V_{dd}. For a piece of data that is not going to be accessed for a very long time, sleep mode will be better because it reduces more leakage power. For a piece of data that is accessed in a moderate amount of time, drowsy mode will be better because there is not a large re-fetch penalty. This sets up one of the fundamental questions answered in our paper — how long is long enough for each mode?

While our paper attempts to address a previously unanswered question, there is a great deal of prior work aimed at reducing leakage power in caches. While we cannot describe all of such techniques in full here, we briefly describe some representative studies here. DRI cache [12] uses the Gated-V_{dd} technique to dynamically adjust the size of the active portion of the cache by turning off a bank of cache lines based on the miss rates. Kaxiras et al. [6] proposed the cache line decay scheme to turn off the cache lines in the dead periods of their cache generations using the Gated-V_{dd} technique. Instead of placing both the tag and the data into the sleep mode, AMC [19] keeps the tag alive and tracks the miss rate with respect to the ideal miss rate. This helps to dynamically adjust the turn-off interval and control the overall performance. Velusamy et al. [16] used formal feedback-control theory to adaptively adjust the cache decay interval and cache lines are turned off accordingly. Another approach to reducing leakage power is called drowsy cache [8], which decreases the supply voltage of idle cache lines. Specifically, all cache lines are periodically placed into drowsy mode. [9] studies techniques for data retention with lower supply voltage. Parikh et al. [10] compared Gated-V_{dd} and drowsy cache at different L2 latencies with HotLeakage and showed Gated-V_{dd} is superior for a set of faster L2 latencies. Heo [4] reduced bitline leakage by leaving bitlines open whose cache banks are not accessed. In contrast, Zhang et al. [17] studied a software approach. They used compiler to insert power mode instructions that control the voltage for the cache lines to control leakage energy.

All of the above approaches strive to develop a scheme for predicting when a section of the cache should be put into a low power mode. They all use some heuristics based on either static analysis or run-time behavior to determine what mode each line should be in. One major open question is: what is the best that these approaches could hope to do? Clearly some of the cache lines will have to be left in a high V_{dd} mode so they can be accessed, but how many and for how long? Are these approaches the ultimate in policy leakage power reduction, or is there still room for improvement?

3. Calculating the Limits of Leakage Power Reduction Techniques

Now that we have reviewed the circuit and architecture techniques employed to reduce leakage power, we describe how to calculate the savings that could be achieved by an optimal approach.

3.1. Cache Intervals

Our analysis of the leakage power saving limit relies on the idea of breaking up the lifetime of each cache line into a series of *intervals*. An interval is the time that a cache line rests between two accesses. If an interval is very long then it would be beneficial to put that cache line in sleep mode for the duration of that interval. If an interval is very short, it should be simply left in a high-V_{dd} mode. If an interval is somewhere in the middle, perhaps drowsy mode would be the best.

To illustrate the above situations, let's take a two-level loop example (Figure 2) extracted from a human resource management application. It counts the total number of people employed during a year. In the example, the interval (I_{add}) of the two consecutive accesses to the same instruction *add* depends on the size of the inner loop. When the range of the inner loop variable j is large, the interval I_{add} is long, which indicates the cache line of add instruction should be put into sleep mode to save leakage power. And when the range is very small, the interval I_{add} is small, which means this cache line should be left in the high-V_{dd} mode for fast accesses. While the range is in the middle, the drowsy mode should be applied to save leakage power without much performance cost. The idea behind our optimal scheme is to determine what the best policy would be for each interval in the program, and then to apply the appropriate leakage technique to that interval.

In an optimal approach, each interval can be thought of as atomic in the eyes of the optimal policy. With oracle knowledge of the future address traces known (as would be for an optimal approach), there should be no reason to perform any new power saving techniques in the middle of an interval. Instead, the same technique should have been applied for the entire duration of the interval as less power would be consumed with the same penalty (for either wakeup or re-fetch).

One thing to note is the notion of live intervals and dead intervals. A live interval starts when a new memory is brought into the cache frame, and ends after the last access. Between the last access to a line of memory and the

```
.....
int i, j, sum, total;
int low(int);
int high(int);
.....
for (total = 0, i = 0; i < 12; i ++)
{
   for (sum = 0, j = low(i); j < high(i); j ++)
      sum += a[j];
   sum *= i;
   add: total += sum;
}
.....
```

Figure 2. The access interval example. The interval length of the consecutive accesses to the *add* instructions depends on the range of the inner loop $|high(i) - low(i)|$.

time it is evicted from the cache, it is regarded as dead. Besides turning off cache lines in dead periods as the cache decay scheme does [6], our method also explores the live period of a cache generation, which demonstrates high potentials in leakage reduction. In fact we found that dead periods did not contribute a large amount of leakage savings in the optimal case, because *any* long interval would be turned off whether live or dead. Thus the only additional savings that are achieved from considering dead intervals are from short dead intervals, of which there are very few. For the rest of this paper we ignore the effect of live and dead intervals, and instead concentrate on the durations of the intervals.

3.2. The Optimal Approach

Our optimal approach works as follows. Each cache access interval is first classified into one of the following three types: sleep-mode optimal, drowsy-mode optimal and active-optimal. If the size of an interval is very small (i.e. there are multiple consecutive accesses within a short period of time), then it is best to leave the cache line in a fully active (non-power saving) mode. If the size of an interval is long, then the best policy is to completely turn off that cache line (sleep) and then re-fetch it when it is needed again. The final case is if the interval size is neither very long nor very short. In this case it is best to put the cache line into a drowsy state, which consumes a small amount of power, has a small wakeup cost and has the advantage of retaining the data values.

The key to dividing intervals into these categories is knowing the precise length of an interval. The interval length where the power saving mode changes is an *inflection point*. There are two inflection points: one between sleep and drowsy modes and the other between drowsy and active modes.

One thing to note is that our optimal approach will have *no-effect* on the performance of the machine. Because we assume perfect access pattern knowledge, an optimal approach can *re-fetch any needed data just before it is needed* and avoid any performance impact. By exploiting this fact we can separate out the power problems from the performance problems. Even though a just-in-time re-fetch or perfect prefetching will not affect the performance of the machine, it does have a power cost which we do consider in this paper. Figure 3 is used to illustrate this point. In the sleep mode, due to turning off the cache line to save leakage power, the data is not preserved. If the data is accessed again, it needs to be refetched, and this refetching process may usually take 7 cycles [10]. Without just in time refetch (Figure 3(b)), the other parts of the whole system will have to stall for these 7 cycles, waiting for the data to be ready. The stall will lead to significant energy consumption as the big circle indicates. Similar things happen to the drowsy mode. But the drowsy mode preserves the data and only takes 1-2 cycles [8] to wake up the cache line. So, without just-in-time refetch (Figure 3(d)), the amount of energy the drowsy mode consumes is less than that of the sleep mode during the system stalling, which is indicated by a small cycle. By contrast, with just-in-time refetch (Figure 3(c) and (e)), the data will be ready when it is needed and the whole system does not need to be stalled and to consume energy for waiting for data.

For the convenience of illustrating how our approach works in general, we will use (Figure 4(a) and (b)) to show how the inflection points are calculated. Figure 4(a) shows that the sleep mode and the drowsy mode require time to reduce the voltage from high V_{dd} to off (s_1) and from high to $V_{dd_{low}}$ (d_1), respectively. Also, there is a similar time overhead in coming out of the mode (s_3 or d_3). For the sleep mode, since the latency D of fetching data from L2 cache is longer than s_3, there is another overhead ($s_4 = D - s_3$) before the next access. We divided the lifetime of an interval into several durations to illustrate these overheads. Figure 4(b) shows the length of each duration s_1, s_2, s_3, s_4, d_1, d_2, d_3 in an access interval of both a sleep mode and a drowsy mode. The total length of the cache access interval using the sleep technique is $s = s_1 + s_2 + s_3 + s_4$, and that of using the drowsy mode is $d = d_1 + d_2 + d_3$.

For the sleep mode, the data has been lost due to an induced miss [6] and must be re-fetched from the memory hierarchy. As such, there is a significant amount of power consumed by the dynamic activity required to fetch the data from the L2 cache, marked with "*" in Figure 4(b). This dynamic power cost can be obtained from CACTI [15].

The sleep-drowsy inflection point is derived as the access interval length when the sleep and the drowsy modes consume the same amount of energy. If the interval is of

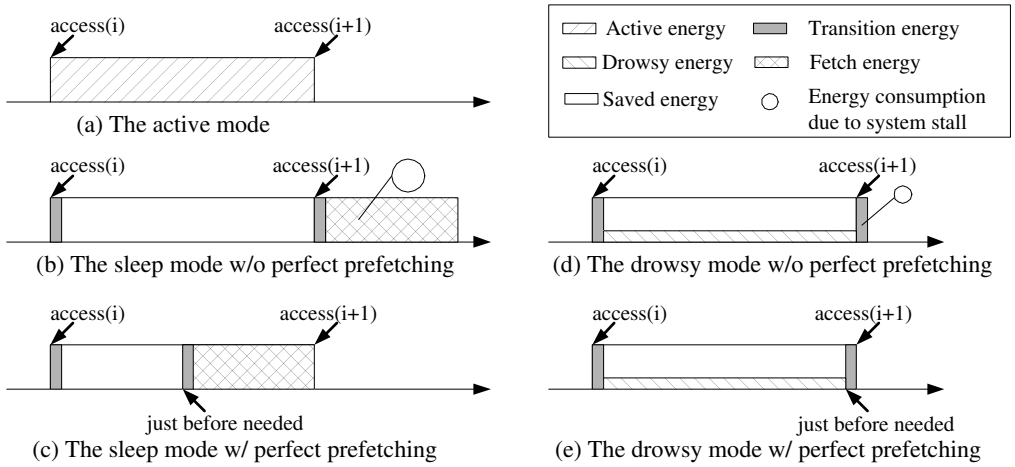
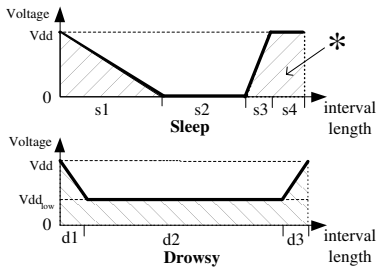


Figure 3. Using perfect prefetching to avoid performance degradation. Assuming perfect access pattern knowledge, an optimal approach uses perfect prefetching to refetch data just before it is needed and avoids stalling the whole system to reduce energy consumption.



Duration	*Voltage*
s1	High to off
s2	Always off
s3	Off to high
s4	High
d1	High to low
d2	Always low
d3	Low to high

✻ Dynamic power consumption due to an induced miss

(b)

Figure 4. Time-Voltage Diagrams of Sleep-Mode and Drowsy-Mode. In Sleep-Mode the cache line is essentially turned completely off and the power consumed drops to nearly zero. While beneficial over a long period of time, there is a more significant overhead due to re-fetch. Drowsy-Mode has a smaller overhead, but the cache line still consumes a measurable amount of power because the voltage has not been completely turned off.

a length less than the inflection point then drowsy mode would be optimal. If it is greater than the inflection point then sleep mode would be optimal. We denote the leakage power consumption of each cache line as P_L, which can be obtained from the HotLeakage tool [18], and the cost of dynamic power due to an induced miss for the sleep mode as C_D. The energy of a sleep mode interval can be calculated as Equation 1:

$$E_S = \sum_{i=1}^{4} P_L(s_i) * s_i + C_D. \quad (1)$$

Similarly the energy consumption using the drowsy model can be calculated as Equation 2:

$$E_D = \sum_{i=1}^{3} P_L(d_i) * d_i. \quad (2)$$

When the two modes consume the same amount of energy, we reach Equation 3:

$$E_S = E_D. \quad (3)$$

Applying the data in Figure 4(b) into Equation 3, we can calculate the sleep-drowsy inflection point.

The other inflection point is between drowsy and active modes. The drowsy-active inflection point is calculated as the sum of the durations d_1 and d_3, within which the voltage changes either from V_{dd} to $V_{dd_{low}}$ or from $V_{dd_{low}}$ to V_{dd}.

Note that the sleep-drowsy inflection point is the point at which sleep mode has the potential to save power of drowsy mode. Sleep mode does not provide benefit at small interval lengths because of the larger penalty associated with coming out of sleep mode (the power of re-fetch) as opposed to drowsy mode. The only way to save power on small interval lengths is to know exactly when the cache line will be accessed again so that it can be brought out of sleep mode before the data is needed. This is how an optimal leakage management scheme would take advantage of it's perfect knowledge. While clearly such an assumption is not easily translated to implementation, there are some ways in which we can hope to approximate perfect knowledge using a form of prefetching (see Section 5).

When an interval between two accesses to the same cache line is longer than the sleep-drowsy inflection point, using sleep mode has the potential to save more leakage power. When an interval is less than the sleep-drowsy inflection point but still greater than the active-drowsy inflection

point, the drowsy mode saves more leakage. When its interval length is less than the active-drowsy inflection point, the cache line is always active and cannot have its leakage power reduced without causing a delay in delivering the data.

```
Input: A set of intervals I
Output: Total leakage power saving
optimal_leakage(I)
    total_saving := 0
    i := 0
    while (I_i ∈ I) do
        if (|I_i| > b) then
            total_saving := total_saving + sleep_saving(|I_i|)
        else if (|I_i| > a) then
            total_saving := total_saving + drowsy_saving(|I_i|)
        else
            no leakage power saving can be obtained
        i := i + 1
    end do
    return(total_saving)
```

Figure 5. The algorithm to compute the optimal leakage power saving.

Figure 5 details our optimal leakage power saving approach. By classifying cache intervals into the three types and applying to them the appropriate leakage saving mode, the maximal leakage power saving can be obtained as the accumulation of the leakage saving over all access intervals, which provides us an upper bound for optimal leakage power savings. It can be proved that based on the perfect knowledge of the lengths of all intervals, the optimal leakage power saving can be achieved by applying the proper operating mode on each interval (see *Appendix*).

3.3. The Generalized Model for Optimal Leakage Power Savings

After illustrating our optimal leakage power saving approach, we evolve our approach to a complete model that can capture the optimal leakage savings while the configuration and technology changes over time. In this model, all the individual assumptions namely the durations, energy costs of transitions between modes, the leakage power consumption of each mode, and the intervals, are parameterized and used as inputs to the model. The transition energies can be obtained from CACTI [15], the leakage power consumption can be obtained from HotLeakage [18], and the intervals are from SimpleScalar [2]. The outputs of the model are the optimal leakage saving percentages of using the optimal sleep, optimal drowsy, and the optimal combining methods.

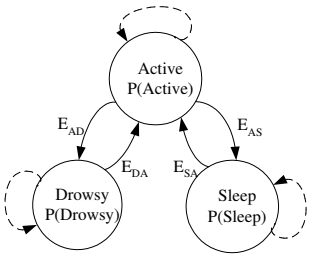


Figure 6. The Optimal Leakage Power Saving Model. The circles indicates states and edges represent transitions between states.

In the model (Figure 6), the circles indicate the states of Active, Drowsy, and Sleep, and the edges represent transitions between states (self edges means that the state remains the same in the next cycle). Each state is associated with its static power consumption (P), and the weights (E_{AD}, E_{AD}, E_{AD}, E_{AD}) on the edges are the transition energy consumptions. For example, E_{AD} is the energy consumption when transiting from the state Active to the state Drowsy.

The model for optimal leakage power savings serves two major functions. First, instead of being an abstract model, it is coded in C language and is publicly availabe for cache leakage studies[1]. Second and the most important is that this model was designed to explore the optimal leakage savings under different architectural and design assumptions with the hope of guiding research effort on leakage power study.

4. Empirical Study

In Section 3, we discussed the limits of leakage power reduction techniques and how they are calculated. In this section we show limit results gathered from actual benchmarks with parameters extracted from modern processors and prior work. Our objective is to evaluate the limits on some leakage power saving techniques as applied to both instruction and data caches. We show an upper bound on the possible savings using Sleep mode, Drowsy mode, or a potential hybrid of the two. We also hope to show how these bounds can be approached using the idea of prefetching.

4.1. Methodology

To test the amount of power that can be saved by using an improved leakage reduction technique, we employed detailed cycle-level simulation. The simulator we use is a version of SimpleScalar closely resembling Compaq Alpha 21264 [7]. The execution core is a 4-wide superscalar pipeline, and the memory hierarchy includes a 64KB, 2-way set associative L1 instruction cache with a single-cycle hit

1 http://www.engr.ucsb.edu/∼yanmeng/leakage.html

latency, a 64KB, 2-way set associative L1 data cache with a 3-cycle hit latency, and a unified 2MB direct-mapped L2 cache with a 7-cycle hit latency. We use LRU as the replacement policy throughout the memory hierarchy.

In order to capture the most important program behaviors while at the same time reducing simulation time to reasonable levels, we used the simulation points that were described and verified in SimPoint [14]. The benchmark suite for this study consists of a set of six SPEC2000 benchmarks: *ammp, applu, mesa, vortex, gcc*, and *gzip* compiled for the Alpha AXP ISA. We chose these benchmarks because they are frequently used in the computer architecture literature, or in the case of *ammp* and *applu*, they are singled out in other leakage power reduction papers.

4.2. Calculating Inflection Points

Inflection points are the keys to choosing the best power saving mode of a given interval distribution. An intervals with its length greater than the sleep-drowsy inflection point is put into the sleep mode, and an interval with its length less than the active-drowsy inflection point is left in the active mode. For an interval with its length between the two inflection points, it is put into drowsy mode to save power and has little performance impact.

Technology	70nm	100nm	130nm	180nm
Active-Drowsy point	6	6	6	6
Drowsy-Sleep point	1057	5088	10328	103084

Table 1. Active-Drowsy and Drowsy-Sleep inflection points dipicted in cycles for different technologies.

To calculate inflection points with respect to different technologies, we used the durations s_1=30, s_3=d_1=d_3=3 and s_4=4 cycles [10] (s_2 and d_2 are dependent on an interval length). When we applied the parameters into Equation 1, 2 and 3, we obtained the inflection points shown in Table 1. The table shows that the value of the sleep-drowsy point decreases while the technology scales down from 180nm to 70nm (These are the only currently available technologies provided by the Hotleakage [18] tool. If in the future Hotleakage is extended to incorporate more technologies, our approach can still be applied to obtain their inflection points). This is due to the fact that the leakage power consumption per cache line increases while the dynamic energy consumption caused by an induced miss decreases with technology scaling down (see Equation 3).

Since 70nm is the most advanced technology that will be reached in a few years according to ITRS [5], we employed it and its corresponding sleep-drowsy inflection point (1057 cycles) in the rest of our study.

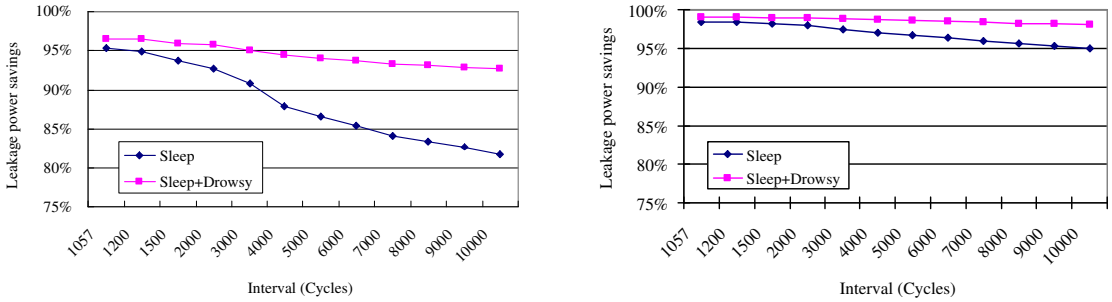
4.3. Combining Sleep and Drowsy Modes

With the inflection points calculated, the first question to be answered is how well a hybrid of sleep and drowsy modes can perform versus sleep mode. If, in the optimal case for sleep mode, we can perfectly predict the distances between access to cache lines then we can potentially make use of sleep mode even if the cache line is accessed every 1057 cycles. In this case, there will be little benefit from using drowsy mode for those cache lines that are accessed more frequently than every 1057 cycles. However, if the threshold was different, if the inflection point between drowsy and sleep modes changed dramatically, there would be a point at which using both drowsy mode (for occasionally accessed line) and sleep mode (for rarely accessed lines) would become beneficial. The purpose of Figure 7 is to demonstrate this point. In our experiments, when a sleep mode is applied, the dynamic power consumption due to an induced miss was removed from the total leakage power savings.

The results in Figure 7 are derived based on the average leakage power saving of all the given benchmarks. Through this figure, we examine the potential effectiveness of a pure sleep mode versus a hybrid sleep/drowsy method where we change the minimum interval length that can be put into sleep mode from 1057 to 10000. These results indicate that a hybrid method (Sleep+Drowsy) can work consistently better than the sleep or the drowsy method alone, especially if one is very conservative about which lines are put to sleep. However, as the minimum sleep length approaches the sleep-drowsy inflection point (decreases), the usefulness of applying the drowsy method in addition to the sleep mode decreases. Under such conditions, the sleep mode removes most of the leakage power and thus there is not much more for drowsy to save. While clearly a non-optimal scheme will not have the luxury of perfect future knowledge, for those that we do have knowledge for, sleep mode should be applied very aggressively.

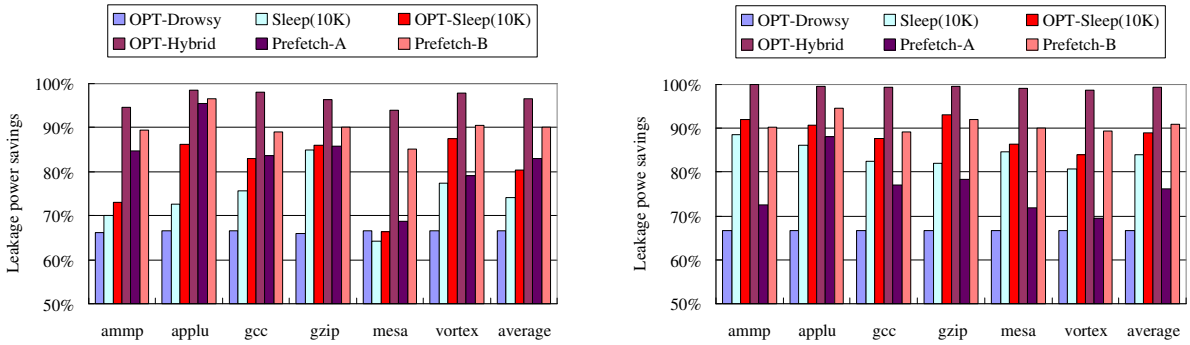
Moreover, the figure depicts that the gap between the hybrid method and the sleep mode for the data cache is much smaller than for the instruction cache. The reason is that the same cache block in the data cache tends to be less frequently accessed than in the instruction cache, and the interval-lengths between consecutive accesses are much longer. Hence, the sleep mode plays a much more important role in the data cache for the leakage power saving than in the instruction cache.

Finally, this figure also confirms that the small variances of the sleep-drowsy inflection point will not change our findings significantly.



(a) Instruction Cache (b) Data Cache

Figure 7. Comparison of the hybrid method vs. the sleep-mode method for the sleep interval-lengths between 1057 and 10000. The usefulness of applying the drowsy method to save leakage power decreases as the sleep length approaches the sleep-drowsy inflection point, and the sleep mode plays a more important role in the data cache for leakage power saving than in the instruction cache.



(a) Instruction Cache (b) Data Cache

Figure 8. Comparisons of Different Leakage Power Saving Schemes.

4.4. Exploring the Upper Bound

With the assumption of perfect-prefetching, the upper bound of the leakage power saving was derived in Section 3 based on the two inflection points. We now explore the leakage-power-saving limits of the following methods assuming perfect knowledge of the future address trace:

- *OPT-Drowsy*: The optimal drowsy cache that has no performance penalty for waking up data (although there is a power penalty as discussed in Section 3).
- *OPT-Sleep(10K)*: The optimal cache line sleeping technique that is capable of sleeping all intervals of a size greater than 10K with no performance penalty.
- *Sleep(10k)*[2]: Similar to the OPT-Sleep(10K) with the exception that instead of optimally turning off any cache line that has an interval larger than 10K, the line must now stay active for 10K and then may be optimally slept.
- *OPT-Hybrid*: The method that optimally combines drowsy and sleep modes based on the inflection points without any performance penalty.

The above listed techniques are depicted by the four leftmost bars of each benchmark in Figure 8. The y-axis is the percent of total leakage power in comparison with a cache in which all lines remain constantly active. As for the two rightmost bars, we will discuss them in Section 5.2.

For the convenience of discussing the implementation of each technique, we define an access interval of a cache line as T_i. OPT-Drowsy puts the line into the drowsy mode during T_i, if T_i is greater than 6; while OPT-Sleep(10K) puts the cache line into the sleep mode during T_i if T_i is greater than 10K. We also studied Sleep(10K) to simulate the cache-decay scheme, whose decay interval is 10K. In this case, a cache line is put into the sleep mode for (T_i-

2 The sleep(10K) is similar to the cache-decay scheme in [6], in which the decay interval was set to be 10K cycles, and the extra leakage power consumed by the counter per cache line was taken into account.

10K) cycles if T_i is greater than 10K. The OPT-Hybrid is to put a cache line into the sleep mode during T_i if T_i is greater than 1057, and to put it into the drowsy mode if T_i falls into the range of (6, 1057]. When T_i is less than 6, all the above methods keep the cache line active to insure fast access time.

Figure $8(a)$ shows that for the instruction cache, the limit of leakage power saving that OPT-Hybrid can achieve is 96.4%. It is 26% higher than Sleep(10K), 16% higher than OPT-Sleep(10K), and 30% higher than OPT-Drowsy. For the data cache (Figure $8(b)$), the leakage power saving limit is 99.1%, which is 15% higher than the Sleep(10K), 12% higher than the OPT-Sleep(10K), and 33% higher than the OPT-Drowsy. The results indicate that while the initial Drowsy and Sleep techniques devised are quite effective, there is still far more potential left in these techniques. Indeed, the leakage power savings for the optimal case are so large that it is fair to say that leakage power would become a less significant portion of the total overall power if these savings could be realized. All these savings could be realized with new policies for cache management. Of course realizing these optimal numbers requires perfect knowledge of the address trace and timing, which is not typically possessed by a management policy.

4.5. Empirical Study with the Generalized Model

To show the feasibility of our model while the implementation technology changes over time, we also study 100nm, 130nm and 180nm processes. Table 2 summarizes the optimal leakage saving percentages we can possibly achieve by using OPT-Drowsy, OPT-Sleep, and OPT-Hybrid methods. Instead of using OPT-Sleep(10K) on intervals that are greater than 10K cycles, we study OPT-Sleep to figure out what is the best leakage power saving we can achieve by aggressively turning off all intervals that are greater than the sleep-drowsy inflection point. The OPT-Drowsy and OPT-Hybrid methods are the same as before. The results in the table are the average results over all the benchmark applications.

The table illustrates that the leakage savings for both the instruction and the data caches of using OPT-Hybrid increase with the technology scaling down from 180nm to 70nm. The increment of the possible leakage savings is due to the decrement of the sleep-drowsy inflection point. Moreover, the table shows that for the 180nm technology implementation, the drowsy mode plays a more important role in saving leakage power than the sleep mode does; while for the others, the sleep mode plays a leader role. This can be also attributed to the large difference of the sleep-drowsy inflection points. Finally, the table also reveals that more leakage savings can be possibly achieved with the technology scaling down, which leaves us more space for further improvement on leakage power savings.

	Technology	70nm	100nm	130nm	180nm
	Vdd (V)	0.9	1.0	1.5	2.0
	Vth (V)	0.1902	0.2607	0.3353	0.3979
I-Cache	OPT-Drowsy (%)	66.4	66.6	66.6	66.7
	OPT-Sleep (%)	95.2	85	80.6	61.5
	OPT-Hybrid (%)	96.4	93.7	91.3	67.1
D-Cache	OPT-Drowsy (%)	66.1	66.6	66.7	66.7
	OPT-Sleep (%)	98.4	96.9	95.3	63.2
	OPT-Hybrid (%)	99.1	98.1	97.3	67.3

Table 2. Optimal Leakage Saving Percentages with Technology Scaling Down.

5. Approximating the Perfect Knowledge

Figure 8 also reveals that even though OPT-Sleep(10K) can achieve a good amount of leakage power saving, there is still great potential to approximate the optimal leakage saving that OPT-Hybrid can accomplish. For instruction cache, the gap is 14.4%, and for data cache, 12.1%. In this section, we will study how prefetching can be employed to approach the optimal in the real situations where perfect knowledge of the future address is unknown.

5.1. Approximation of the Perfect Knowledge with Prefetching

In Section 3, we provide the upper bound of leakage saving, yet it is derived assuming perfect knowledge of the timing of the address trace. This information is used to insure that no additional latency overhead will be added to the next access of the cache line for either scheme because performance overhead can be perfectly overlapped with the length of the interval (see Figure 3.) As a result, there will be no extra stalls that result in leakage power consumption of the whole system.

While the upper bound on the amount of savings that can be achieved is a useful notion to characterize, an actual implementation will require that we approximate this perfect knowledge. Luckily, such a set of future address predictions is already well studied in a different area of processor research — prefetching. The goal of prefetching is to accurately predict future access patterns so that they can be optimistically fetched from memory before their use. We propose that prefetching can optimistically re-fetch data that has been either turned off for sleep mode or put into a drowsy state. This would allow a more aggressive sleep mode policy as the penalty that is normally associated with sleeping a line can be fully or partially hidden by the approximated future knowledge that prefetching provides.

Prefetching is one of the primary techniques used in modern processors to tolerate memory latency. Many models have been proposed for prefetching either instructions or data to reduce latency, ranging from compiler-based prefetching [11] to hardware-based prefetching [13]. In this paper, we examine the potential use of two hardware based schemes to capture simple but frequent forms of misses: next-line prefetching and stride-based prefetching.

The idea behind next-line prefetching is that a cache miss will likely be followed by one or more cache misses to consecutive cache blocks. Because programs exhibit a great deal of spatial locality, this form of prefetching is very effective in hiding latency, and meanwhile is fairly straightforward to implement.

Stride-based prefetching is to eliminate miss patterns that follow a regular pattern but access non-sequential cache blocks. This type of accesses frequently occurs in programs that use multidimensional arrays. To capture this type of misses, Farkas et al [3] showed that the most efficient way is by examining the access patterns on a per static load basis. A miss is taken as stride miss if the same stride has been seen at least twice for the static load accesses.

In this study, we employ the next-line prefetching technique for the instruction cache, and both next-line and stride-based techniques for the data cache, since most of the cache misses can be captured by these schemes based on the study in [13].

5.2. Employing Prefetching to Approach the Limit of Leakage Power Savings

With the attempt to approach the upper bounds, we took the access intervals within which one or more accesses to the previous cache line occurs as prefetchable (P) and the other intervals as non-prefetchable (NP). Since the intervals that are less than 6 are always kept active and there is no need to prefetch them, they are counted as non-prefetchable. The prefetchability is deduced as the number of prefetchable intervals over the total number of intervals.

Figure 9 shows the prefetchability of cache accesses. We divided the intervals into three types, $(0, 6]$, $(6, 1057]$, and $(1057, +\infty)$, based on their lengths. The shaded areas within the range of $(6, 1057] \cup (1057, +\infty)$ cover the percentage of the intervals that are prefetchable, and the other areas cover the intervals that are non-prefetchable. For the instruction cache, the prefetchability of using the next-line prefetching technique (P-NL) is 23%; and for the data cache, the prefetchabilities of using the next-line (P-NL) and the stride-based (P-stride) prefetching techniques are 16.3% and 5.1% respectively. So, the total prefetchability out of all intervals is 23% for the instruction cache and 21.4% for the data cache.

Interval	Prefetchable	Nonprefetchable
$(1057, +\infty)$	Sleep	Active
(6,1057]	Drowsy	Active

(a) Prefetch-A

Interval	Prefetchable	Nonprefetchable
$(1057, +\infty)$	Sleep	Drowsy
(6,1057]	Drowsy	Drowsy

(b) Prefetch-B

Table 3. Prefetch-A and Prefetch-B methods, emphasizing high performance and high leakage power saving, respectively.

When employing prefetching to approach the upper bound, we designed two methods: Prefetch-A and Prefetch-B based on the understanding of the two fundamental objectives of pursuing high performance and low power.

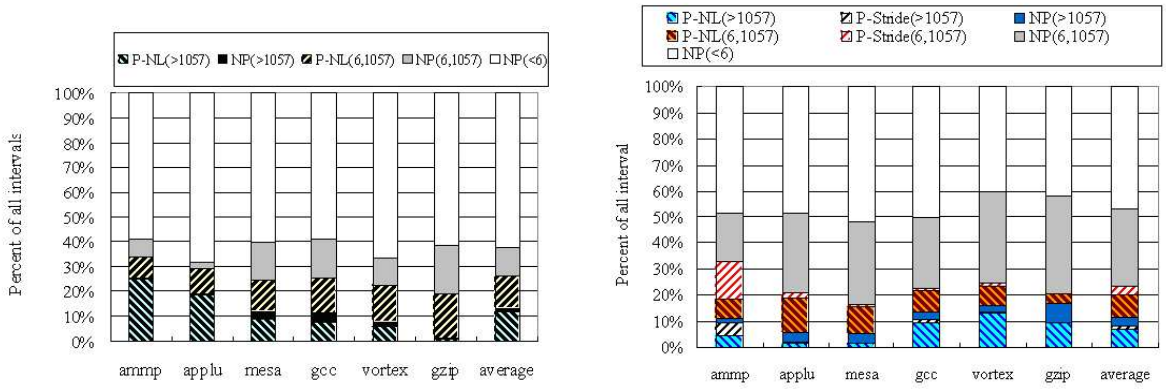
- *Prefetch-A* was designed to provide the best performance that the corresponding prefetching techniques could achieve.
- *Prefetch-B* was designed to provide the best power saving that the corresponding prefetching techniques could achieve.

Table 3 details the two methods. For both of them, when an interval was identified as prefetchable, a corresponding low power mode was applied onto it. For the non-prefetchable cases, Prefetch-A keeps the cache lines always on, while Prefetch-B puts them into drowsy mode.

The limit results with prefetching are shown as the two rightmost bars in Figure 8. From the figure, we can have the following four observations. Prefetch-B can closely approach the optimal (within 5.3% for the instruction cache and within 6.7% for the data cache). For the instruction cache, Prefetch-A achieved 10% better than Sleep(10K) and Prefetch-B achieved 21% better than Sleep(10K); and for the data cache, Prefetct-B achieved 7% better than Sleep(10K). The leakage power difference between Prefetch-A and Prefetch-B is due to the unprefetchable intervals within the range of $(1057, +\infty)$, and it can be alleviated by employing more sophisticated prefetching techniques. Finally, the best design trade-off of power and performance is somewhere in between of the Prefetch-A and Prefetch-B methods, which will be studied in our future work.

6. Conclusions

Leakage power dissipation is quickly becoming a major concern in designing high performance processors. In this paper we explore the limits to which known circuit level techniques can be combined and employed to save cache leakage power using new management methods and protocols. In addition, we developped a parameterized model to



(a) Instruction Cache (b) Data Cache

Figure 9. Prefetchability of intervals with different lengths.

determine the optimal leakage savings while the implementation technology changes over time. We also find that it is possible with perfect knowledge of the future address trace to reduce the amount of power dissipated by the instruction cache down by a factor of 5.3 from known techniques (2 for the data cache). At this level, the leakage power of the cache would become a less serious problem. We further show that, while a hybrid method that combines both sleep and drowsy modes is not very useful if each is used optimally, it can substantially reduce leakage power by applying the appropriate power saving mode on each access interval when the assumptions are less favorable. One such less favorable assumption is that perfect knowledge of the future is unknown, but it can be approximated using prefetching. We evaluate the potential usefulness of next-line and stride-based prefetching toward reducing leakage power and show that if prefetching can be used to guide sleep mode and drowsy mode is used the other times then the leakage power dissipation will be within a factor of 2.5 from the optimal.

References

[1] L. Belady. A study of replacement of algorithms for a virtual storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[2] R. Desikan, D. Burger, S. W. Keckler, and T. M. Austin. Sim-alpha: a validated execution driven alpha 21264 simulator. Technical Report TR-01-23, Department of Computer Sciences, University of Texas at Austin, 2001.

[3] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic. Memory-system design considerations for dynamically-scheduled processors. In *In ISCA-24*, June 1997.

[4] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. In *the ISCA-29*, Anchorage, Alaska, May 2002.

[5] International technology roadmap for semiconductors. http://public.itrs.net.

[6] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. In *the ISCA-28*, Gőteborg, Sweden, June 2001.

[7] R. Kessler. The alpha 21264 microprocessor'. In *IEEE Micro*, pages 24–36, Mar. 1999.

[8] N. Kim, K. Flautner, D. Blaauw, and T. Mudge. Circuit and microarchitectural techniques for reducing cache leakage power. *IEEE Trans. VLSI*, 12(2):167–184, Feb. 2004.

[9] N. Kim and T. Mudge. Single vdd and single vt super-drowsy techniques for low-leakage high-performance instruction caches. In *Proc. of the ISLPED*, Newport Beach, CA, Aug. 2004.

[10] Y. Li, D. Parikh, Y. Zhang, K. Sankaranarayanan, K. Skadron, and M. Stan. State-preserving vs. non-state-preserving leakage control in caches. In *DATE*, Feb. 2004.

[11] Todd C. Mowry, Monica S. Lam, and Anoop Gupta. Design and evaluation of a compiler algorithm for prefetching. In *ASPLOS 1992*, pages 62–73, Oct. 1992.

[12] M.D. Powell, S. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Reducing leakage in a high-performance deep-submicron instruction cache. *IEEE Transactions on VLSI*, 9(1), Feb. 2001.

[13] S. Sair, T. Sherwood, and B. Calder. Quantifying load stream behavior. In *the HPCA-8*, Feb. 2002.

[14] T. Sherwood, E. Perelman, G. Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *ASPLOS 2002*, San Jose, CA, Oct. 2002.

[15] P. Shivakumar and N.P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical Report WRL-2001-2, HP Labs Technical Reports, Dec. 2001.

[16] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *2002 Workshop on Memory Performance Issues in conjunction with ISCA-29*, Anchorage, Alaska, May 2002.

[17] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Complier-directed instruction cache leakage optimization. In *IEEE Micro-35*, Istanbul, Turkey, Nov. 2002.

[18] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. R. Stan. Hotleakage: An architectural, temperature-aware model of subthreshold and gate leakage. Technical Report Tech. Report CS-2003-05, Department of Computer Sciences, University of Virginia, Mar. 2003.

[19] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte. Adaptive mode control: a static power-efficient cache design. In *PACT*, Sep. 2001.

Appendix: Theorem of the Optimal Policy for Leakage Power Saving

In this section, after defining the relevant terms in our study, we provide the theorem of the optimal policy for leakage power saving.

Definition 1: We define **I**=$\{I_i\}$ as a set of intervals, and the length of interval I_i as $|I_i|$ ($|I_i| \in (0, +\infty)$).

Definition 2: For each interval $I_i \in \mathbf{I}$, we define three possible operating modes $T_j \in \mathbf{T}, where \mathbf{T} = \{T_1 = active, T_2 = drowsy, T_3 = sleep\}$, and the leakage energy saving of the interval I_i working in the mode of T_j is defined as $E(I_i, T_j)$.

Definition 3: We define two inflection points, the active-drowsy inflection point a and the sleep-drowsy inflection point b. The active-drowsy inflection point is defined as the sum of the durations within which the supply voltage changes either from high to low or from low to high. The sleep-drowsy inflection point is defined as the access interval length when the sleep and the drowsy modes consume the same amount of energy.

Lemma 1: The active-drowsy inflection point a is less than the sleep-drowsy inflection point b.

Proof: Because loads have physical capacities, the discharging process takes less amount of time for the voltage dropping from high to low than from high to off, i.e. $d_1 < s_1$. Similarly, the charging process takes less time for increasing the voltage from low to high than from off to high, i.e. $d_3 < s_3$. So, the sum of the durations $a = d_1 + d_3 < s_1 + s_3$. Since the sleep-drowsy inflection point b is greater than $s_1 + s_3$, we can arrive at the conclusion that a is less than b. Our study based on the 70nm technology process also justifies that a (6 cycles) is less than b (1057 cycles) from the experimental perspective.

Theorem 1: Under the context of the independent model, where access intervals of a cache block are independent from each other, we assume that for each interval $I_i \in \mathbf{I}$, one and only one of the three operating modes $T_j \in \mathbf{T}$ can be applied for reducing leakage energy consumption based on the following policy:

1. When the interval length $|I_i| \in (0, a]$, the active operating mode or non-power saving mode is applied.
2. When the interval length $|I_i| \in (a, b]$, the drowsy mode is applied.
3. When the interval length $|I_i| \in (b, +\infty)$, the sleep mode is applied.

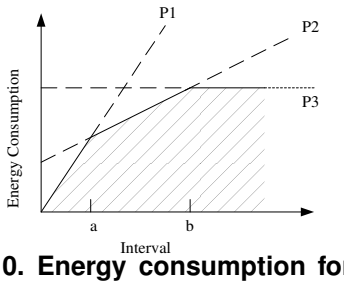


Figure 10. Energy consumption for each of the three operating modes and the lower envelope $E(I_i, T_j)$ function for minimal energy consumption.

Then the maximal leakage saving can be obtained as the combination of the power saving over all intervals $I_i \in \mathbf{I}$, which gives an upper bound for optimal leakage power saving.

Proof: We prove the theorem by contradiction. We divide the whole range of the interval length $(0, +\infty)$ into three independent portions based on the active-drowsy inflection point a and the sleep-drowsy inflection point b, i.e. $(0, a] \cup (a, b] \cup (b, +\infty)$ (see Lemma 1 that $a < b$). Suppose the energy saving M based on the above assumptions is not maximal, then there must be another energy saving M' that is greater than M, which indicates that there is at least one interval I_i whose operating mode T_j' is different from T_j.

Figure 10 shows the function of interval vs. energy consumption. In the figure, we can have the following derivations:

1. The function is continuous and monotonically increasing.
2. The slopes P_1, P_2 and P_3 indicate the power consumptions within the interval ranges of $(0, a]$, $(a, b]$ and $(b, +\infty)$ respectively.
3. For intervals in the range of $(0, a]$, the minimal energy consumption can be achieved through the active mode T_1. For intervals in the range of $(a, b]$, the minimal energy consumption can be achieved through the drowsy mode T_2. For intervals in the range of $(b, +\infty)$, the minimal energy consumption can be achieved through the sleep mode T_3.

For a set of independent intervals, if at least one interval I_i was applied with T_j', not the corresponding mode T_j, then $E(I_i, T_j')$ is greater than $E(I_i, T_j)$ (above the shadow area in Figure 10), giving the contradiction. Therefore, the maximal leakage power saving can be obtained by the proposed policy.