Using Dead Block Information to Minimize I-cache Leakage Energy

Mohan G Kabadi[†] **and Ranjani Parthasarathi**[††],

S.J.C. Institute of Technology
Chickballapur, INDIA

Annauniversity
Chennai, INDIA

Summary
Power-conscious design using hardware and/or software means has become crucial for both mobile and high performance processors. This paper explores integrated software and circuit level technique to reduce the leakage energy in iL1-cache of high performance microprocessors by eliminating the basic blocks from the cache, soon after they become ‘*dead*’. The impact of eliminating dead blocks on processor performance and energy efficiency are analyzed in detail. The compiler normally identifies the basic blocks from the control flow graph of the program. At this stage candidate basic blocks that can be turned-off after use are identified. This information is conveyed to the processor, by annotating the first instruction of the selected basic blocks. During execution, the basic blocks are kept track of, and after use, the cache blocks occupied by these blocks are switched-off. Experiments have been conducted by considering two different initial states of the cache - on and off, and the leakage energy saved varies from 0.09% to a maximum of 96.664%.

Key words:
Power-efficient architecture, Low-leakage cache, Compiler-assisted energy optimization

1. Introduction

Power-aware design has become a prime design consideration, together with performance, in the computer systems of today. The fabrication technology of VLSI circuits is steadily improving and the chip structures are being scaled down. The on-chip transistor density is increasing at a higher ratio, resulting in an increased level of power consumption. Further, as devices shrink, gates move closer and more current leaks between them. Hence, an important technical challenge for designers is in reducing the current leakage. Thus controlling/reducing current leakage has become an active area of research.
Memory subsystems, especially on-chip caches, are a dominant source of power consumption. Caches often consume 80% of the total transistor budget and 50% of the area. Hence, on-chip caches are good candidates for controlling the on-chip leakage energy.
Several techniques have been proposed to reduce leakage energy dissipated by cache subsystems, both for multilevel I-caches [1,2,3,4,5,6] and d-caches [4,5,6,7,8]. Such techniques can be grouped under two categories, namely, (i) architecture alternatives adopting static [5] or dynamic methods [3,4] and (ii) software techniques using compiler support [1,6]. Many of the architectural level mechanisms work at the circuit level, at the cache bank or line granularity. They work by putting the idle cache blocks to low-leakage mode after a predefined interval of inactivity. In Dynamic ResIzable I-cache [3], size of the cache dynamically adapts to application demand during execution. In Cache-line decay [7], cache blocks are turned-off by monitoring the periods of inactivity using line-saturating counters with each block. In adaptive-mode-control cache[4], the individual cache lines of the data store is controlled by monitoring the performance. In Drowsy-cache [8], leakage energy of the cache blocks is controlled by periodically placing all the cache blocks in a *‘drowsy-state’* by reducing the supply voltage to the blocks.

The software methods for controlling the leakage energy are relatively simple and assist the microarchitecture in reducing the switching activity in on-chip caches by giving explicit information on program behavior [1,9] or by compiler optimization techniques [6]. For example, in the L-cache [1], holding loop-nested basic blocks designated by the compiler in a small cache reduces switching activity. The profile from the previous runs is used to select the best instructions to be cached. The compiler-assisted approach used in [9] works at a loop level granularity. The cache lines are put into low leakage mode when the next access to the instruction occurs only after a long gap or would never occur.
Here we present and analyze a generic, novel, compiler-assisted technique, called ‘*Dead Block Elimination in Cache*’ (DBEC)[10], wherein, selected lines of the cache are turned-off under program control, resulting in leakage

Manuscript received May 5, 2007
Manuscript revised May 20, 2007

energy savings. This approach takes care of both loop-intensive and non loop-intensive programs without sacrificing performance.

2. Dead Block Elimination in Cache

The *DBEC* mechanism uses software-assistance to turn-off the *'dead'* basic blocks of i-cache. The instructions that are not *'live'* at a particular point of program execution, and would not be used again before being replaced in the cache, are termed as *'dead'* instructions. The proposed scheme identifies the cache blocks that contain the *'dead'* instructions and turns-off the power to such cache blocks, for conserving the leakage energy. The information on whether an instruction is *'dead'* at a particular point of program execution is obtained from the compiler. The '*dead'* instructions are handled at the granularity of basic blocks. The compiler identifies the basic blocks from the control-flow graph (CFG) and indicates the start and end of the basic blocks in the code. During program execution, this information (in the form of annotations) is used by the microarchitecture to turn-off the 'dead' blocks.
The compiler uses three types of annotations. Each cache line is marked when it is taken up for execution. The first type of annotation indicates when the cache lines that have been marked are to be switched off. This type of annotation is normally done for the instructions that begin a basic block and would cause the processor to emit a 'turn-off signal'. In the case of loops, only the first statement of the first basic block is annotated so that the basic blocks with in the loops do not switch off other basic blocks of the same loop. This ensures that the cache blocks containing the loop code are not switched-off before the next iteration of the loop. These cache blocks are to be switched-off only after the last iteration, when the control exits the loop.
The second type of annotation indicates that the next instruction is a "call instruction" with in a loop. The third type of annotation indicates that the previous instruction was a function call, which is part of a loop. The second and third type of annotations ensure that 'functions', which are called from within a loop, do not switch off the cache blocks that contain the code of that loop. This mechanism is explained below with an example.
Figure 1 shows a segment of CFG. The BBi's are basic blocks. The first statement of each basic block is annotated during the compilation stage. The first type of annotation is used only for the first statement of the basic blocks BB1, BB2 and BB5. However, BB3 and BB4 are the basic blocks of the same outer loop and hence, the first statements of these blocks are not annotated. This is done to make sure that BB2 and BB3 are not turned-off when BB4 is under execution. When the annotated statement of BB2 is under execution, the execution of BB1 is completed. Hence, the cache blocks completely occupied by the instructions of BB1 are turned-off. Similarly, the execution of the first statement of BB5 will turn-off the blocks of BB2, BB3 and BB4. Thus, the compiler exactly determines the instant at which a particular cache block is going to be 'dead'. This approach has a negligible performance penalty as opposed to earlier approaches [3,5,6]. Further, this primarily being a static approach supported by the compiler, the run-time architectural overhead of this approach is also minimal. Two simple hardware mechanisms have to be added as explained below.

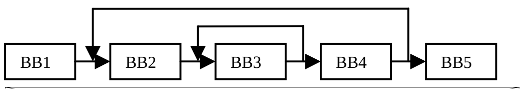

Figure 1. An example segment of CFG

2.1 Hardware Modification

The implementation of the DBEC scheme requires two simple modifications, - one on the cache side and the other in the processor module. Figure 2 shows the hardware modification required in the cache module. With each block of the i-cache, one bit called the *'turn-off bit'* is added to the tag bits. These turn-off bits are initially reset to zero at the start of program execution. The turn-off bits corresponding to blocks of cache that contain the instruction of the current basic block, or loop under execution, are set. These bits indicate the blocks that are to be turned-off, when the execution of the current basic block or loop is completed. In addition, one bit called the Flag bit is also added per block along with the Tag bits. This bit is used to take care of cache blocks which have instructions from more than one Basic block.

When an instruction is fetched from a new cache block, the Turn-off bit corresponding to that cache block is set to 1. During the fetch process, on the occurrence of the next annotated statement, a flag called the Basic Block Crossing (BBC) flag is set and the Turn-off bit corresponding to that cache block is reset. The BBC bit is used to indicate that this cache block should not be turned off as it contains the instructions from the next Basic block. Hence, once the BBC is set, during instruction fetch, the flag bit corresponding that cache block is set, and the *Turn-off bit* is reset.

When the annotated instruction is executed, the '*turn-off signal'* issued causes all blocks which have their

Turn-off bit set to 1 to be turned off. On the falling edge of *'turn-off signal'*, the flag and its corresponding Flag bits are copied to the turn off bits, and the flag bits are reset. For this to work, it is necessary that the annotation field of the instructions is pre decoded (as the instruction is fetched). This will help to identify the annotated instruction at an early stage.

The blocks are turned-off and put to a state-destroying mode using a mechanism similar to the *drowsy-cache mechanism* used in [8]. This mechanism works well for both direct as well as set-associative caches.
This mechanism can be explained using the same example given in the previous section. Against the execution of the first annotated statement of BB1, the processor sets the turn-off bits of those cache blocks from which the instructions of BB1 are fetched. An instance of BB1 occupying an arbitrary number of bytes is shown in figure 2. Also, the turn-off bits for those blocks at the end of execution of BB1 are shown in the same figure. The turn-off bit of cache block X3 is not set, indicating that the cache block is to be retained even after the execution of the BB1. This is because cache block X3 is only partially filled with instructions from BB1, and contains some code from the BB2 also. When the control reaches the beginning of BB2, the annotated first statement of BB2 will trigger the invalidation and a *'turn-off-signal'* is emitted. This will turn-off all those blocks whose turn-off bits were previously marked for this purpose. Hence, the cache block X1 and X2 would be turned-off. The turn-off bit of X3 is set only when the basic block BB2 is taken up for execution. When the execution of basic block BB2 is completed, the cache block X3 is turned-off.
In the processor module, a counter is needed to track the depth of execution when the function(s) are called from the main or other functions. This counter is used with the second and the third type of annotations described in the previous section. The annotation before the call instruction increments the counter, whereas the instruction following it decrements it. Switching off of cache blocks is performed only when this counter value is zero indicating that the code under execution is not part of any loop. Instructions to manipulate the dedicated counter, which keeps track of entry and exit of calls, are needed. An alternative is to have a dedicated register and use the increment and decrement instructions of the existing ISA before and after the function calls.

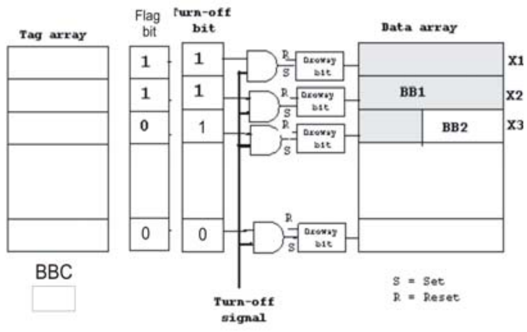

Figure 2. Hardware details of DBEC mechanism.

3. Experimental Methodology and Results

3.1 Experimental Setup

The Simplescalar-3.0 [11] simulator has been modified to simulate switching-off of the cache blocks when annotated instructions are encountered in the instruction stream. A set of benchmark programs from the SPEC 2000 [12] suite and Media-benchmark [13] suite have been used to evaluate the performance of the proposed scheme.
The simulator is used to collect the results of energy consumption. The compiler is modified to annotate the instructions that should switch-off the cache blocks. The various phases in the compilation are depicted in figure 3. The programs selected from SPEC 2000 have been run for 4billion instructions, and the Media-benchmarks have been run to completion to collect the statistics of simulations. Simulations are performed for two different initial states of the cache blocks namely, - on and off. The initially-on case is more or less a conventional cache with leakage control mechanism, incorporated. However, when the cache blocks are initially-off, blocks are brought to active state only on the first access to them. They remain in active state till their last use. The turn-on latency should not affect the performance of the processor. Typically the turn-on latency of L1-cache will be 1 cycle for 0.07micron process [8], and the L2-cache latency will be in the range of 6 to 8 cycles. Thus, the latency of turning-on the cache block will be completely hidden by the microarchitecture.

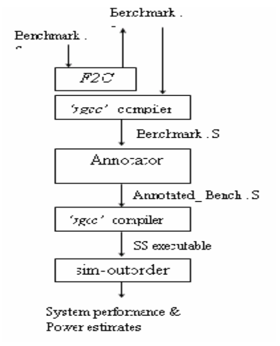

Figure 3. Various phases simulation in DBEC scheme

3.2 Simulation Results

The main parameter considered in the evaluation of the DBEC scheme is the leakage energy savings in the i-cache. To study the impact on performance, the parameters, instructions per cycle (IPC) and miss rate are considered. The leakage energy of the cache is proportional to the total number of cache blocks that are switched-on in the cache. Hence, the total number of blocks switched-off is taken to

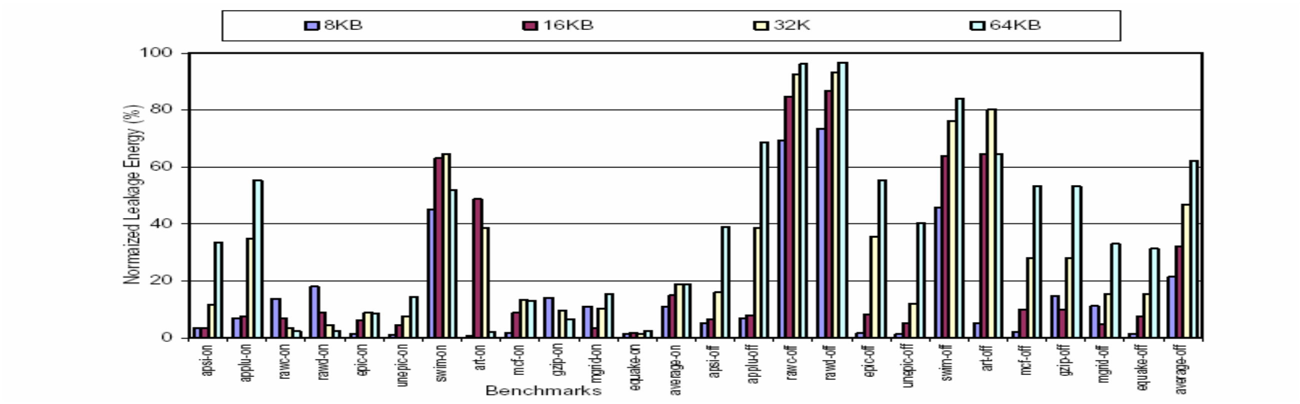

Figure 4. Leakage Energy saved for various benchmarks with two different initial states of cache.

be an estimate of the power savings achieved. The leakage energy saved is normalized with respect to the energy of the base model and expressed in percentage.

$$\% \text{ Energy saved} = \left[1 - \frac{\sum_{i=1}^{n} N_{Li} * T_{ai}}{T_{NB} * T_{D}} \right] \quad (1)$$

Where N_{Li} = Number of active lines during the execution of ith Basic block.

T_{ai} = Duration of activity or duration for executing the ith Basic block.

T_{NB} = Total number of blocks.

T_D = Total duration

Leakage energy saved/cycle (in Joules)

= Number of lines turned off * A ---(2)

Where A = energy dissipation per line per cycle

= 0.33 pJ /32Bytes for 0.07 micron technology [9]

Table 1. Parameters Used in the DBEC Simulation.

Parameter	Value
Fetch Width	4 Instructions Per Cycle
Decode Width	4 Instructions Per Cycle
Commit Width	4 Instructions Per Cycle
iL1 Cache	8k, 16k, 32k and 64k Direct-Mapped
iL1 Cache Block Size	32 Bytes
iL1 Cache Latency	1 Cycle
dL1 Cache	16 K 4-way, 32 bytes Block
iL2 Unified Cache	256 K 4-way, 64 bytes Block
iL2 Cache Latency	6 Cycles
iL1 cache leakage energy	0.33 pJ per block per cycle

The parameters chosen for the simulation is shown in table1 and results of the study are presented below. Figure 4 shows the percentage leakage energy saved for various benchmark programs as the cache size is varied from 8KB to 64KB. Figure 5 shows the absolute leakage energy saved for these programs. The xxx-on/xxx-off indicates the results obtained by keeping all the cache blocks in initially-on/initially-off state. The energy saved varies from a minimum of 0.09% to a maximum of 64.8% when all the cache blocks are initially-on and 1.279% to 96.664% when cache blocks are initially-off. Compared to the results obtained for initially-on, the leakage energy savings keeps increasing consistently as the cache size is increased from 8KB to 64KB. Further, the leakage energy saving is higher when the cache blocks are initially-off. This mechanism is able to control the leakage energy of the blocks that are 'occupied' and 'unoccupied' by the instructions of the program. This is due to the reason that, the blocks that are occupied will be turned-off after they become 'dead', where as the blocks that are unoccupied remain in off state from the beginning itself resulting in higher leakage energy savings.

In both the 'initially-on' and 'initially-off' cases, the leakage energy saved shows some interesting behavior for different programs. For instance, from figure 4, it is observed that in some programs namely apsi-on, applu-on and unepic-on, the leakage energy saved gradually increases as the cache size is increased. The IPC and iL1-miss rates of these programs are shown in table 2. From table 2, it is observed that as the cache size increases, the iL1-miss rate for these programs decreases and correspondingly IPC increases. Hence, it may be inferred that the capacity misses have been reduced and the basic blocks residing in the augmented blocks are brought

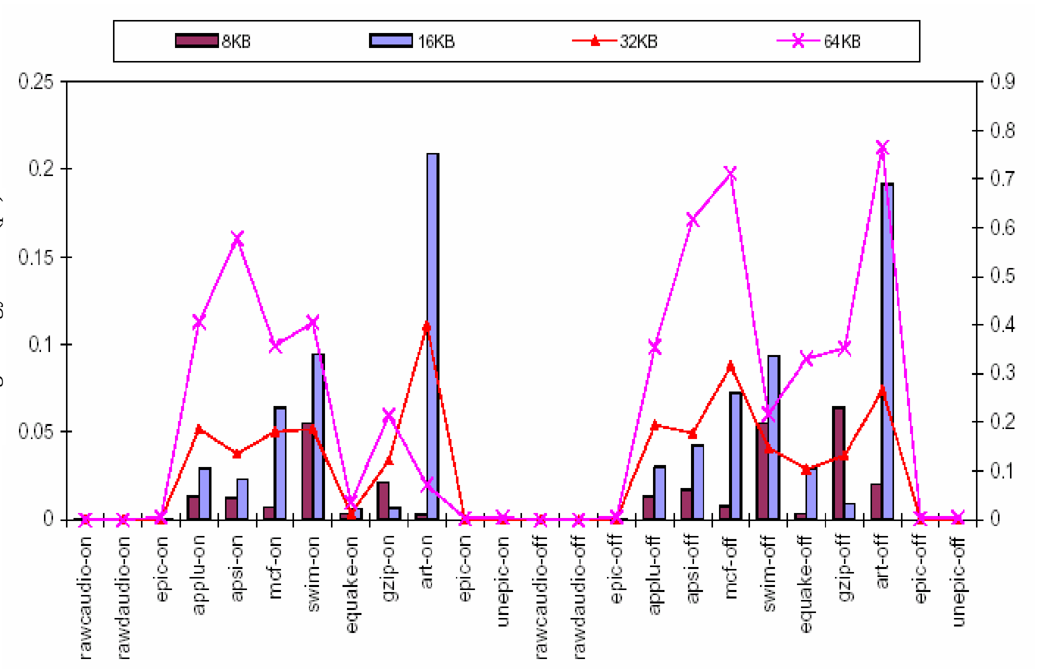

Figure 5. Leakage Energy saved for various Benchmarks and different cache sizes

under the control of the leakage control mechanism, there by increasing the leakage energy savings.

In two other programs shown in figure 4, namely *rawcaudio-on* and *rawdaudio-on*, the percentage leakage energy saved decreases gradually as the cache size is increased. From table 2, it is observed that the miss rates are zero and the IPC remains almost constant as the cache size is increased. That is, the augmented portion of the cache due to increase in cache size has not been brought under the leakage control mechanism. In effect, (in the initially-on case,) the DBEC mechanism controls the leakage energy of only the cache blocks that are occupied by the code and does not control leakage energy of the portion that is never used for storage. For such programs, the initially-off should give better energy savings, which is obvious from the results obtained for these programs. From figure 4, it is observed that for rawcaudio-off and rawdaudio-off, the energy saved for

64KB cache size is 96.13% (2.29% for rawcaudio-on) and 96.64% (2.24% for rawdaudio-on) respectively.

In another set of programs, namely *epic-on, swim-on, art-on* and *mcf-on*, the percentage leakage energy saving increases as the cache size increases and beyond 16KB or 32KB cache size, it decreases gradually. In *'art'*, the IPC and iL1 miss-rate remain the same (refer table 1) when the cache size changes from 16KB to 32KB. Hence, the percentage leakage energy saving decreases when the cache size increases from 16KB to 32KB. Similarly, in *'swim'*, the IPC and miss rates remain the same when the cache size increases beyond 8KB. Though a small percentage of increase in leakage energy saved is observed when the cache size increases from 16KB to 32KB, it decreases from 63.93% down to 51.8684% when the cache size is increased to 64KB. A similar observation is found in the mcf program too.

To understand the variation observed in leakage energy savings across these programs, a detailed study has been conducted. Three programs (namely equake-on, swim-on and apsi-on) that have shown wide variation in energy savings are selected. These programs are run for a sufficiently large number of instructions. The CFG of these programs are also analyzed in detail to understand the performance behavior. The number of blocks that remains in active state and the energy saved is recorded at every 50million instructions of simulation and the results are shown in figure 6. In the case of figure 6(a) and 6(b), the results are shown till the point, beyond which the variations in the results are negligible.

In *'equake'*, the energy saved is not considerable till 2.7billion cycles and during this interval none of the blocks have been turned-off. From 2.85billion cycles to 8.15billion cycles the number of blocks that remains in active state is 337 showing about 33% reduction in iL1-cache power demand and a steep increase in leakage energy saved.

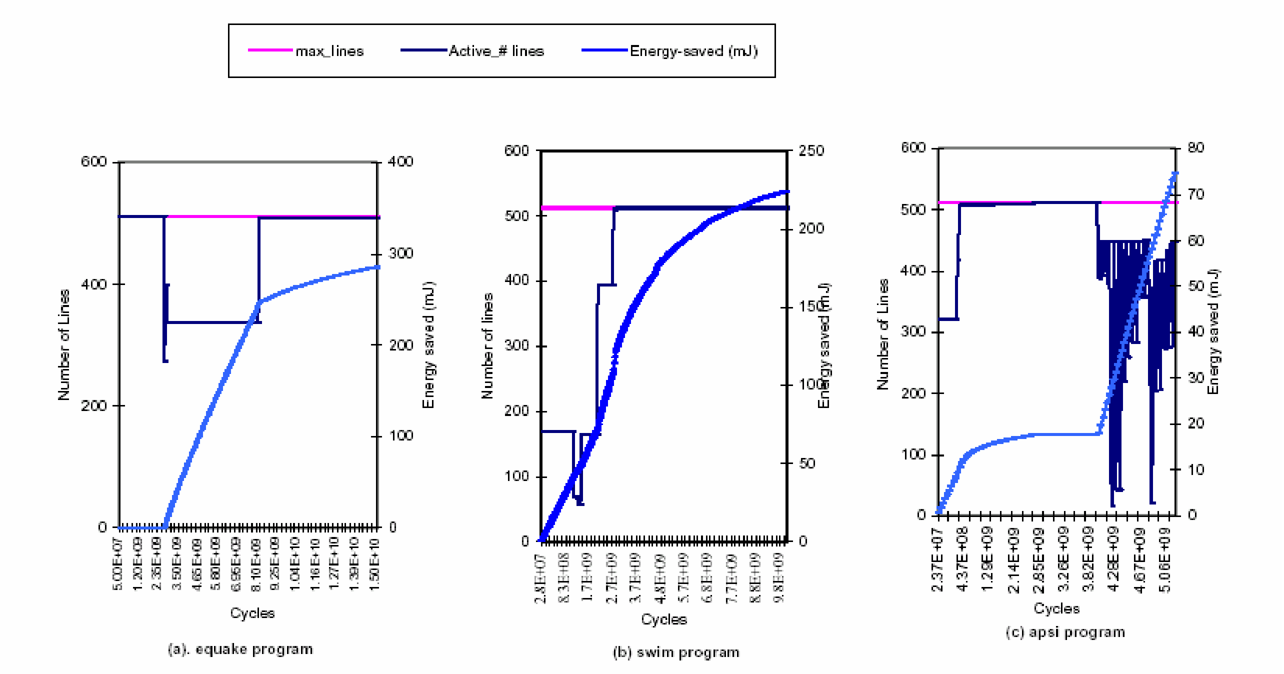

Figure 6. Active number Lines and Leakage Energy saved for three different programs.

In *'swim'*, the leakage energy saved keeps increasing gradually in the beginning, till 3billion cycles, beyond which the change in the leakage energy saved is negligible. This program has few independent basic blocks at the beginning containing the code for initialization. Except these few basic blocks, the remaining f this program are enclosed in a loop. Hence, none of the cache blocks gets turned-off beyond basic blocks o this point until the end of the simulation. In contrast to the above two, *'apsi'* has relatively large number of independent basic blocks from the start till the end of program. The energy saving also varies accordingly.

To demonstrate that the proposed mechanism does not have any negative impact on the performance, the IPC and iL1 miss rates of the modified cache are compared with those of the base model cache of similar size. Column 4, 7 and 8 of Table 2 show the IPC and miss rates for the modified and base model caches for a size of 16KB. It is observed that the performance is the same as the base model cache. Hence, the DBEC mechanism does not degrade the performance of the processor. This is as expected due to the reason that, in the proposed scheme, a cache block is put to low leakage-state only after its last use in the program.

3.3 Minimizing Overheads

DBEC is not without its overheads. It requires additional bits per cache block to track the *dead* and *live* cache blocks. These additional bits are always on and hence end up consuming some energy. The overhead due to these additional bits may be estimated as follows:
Assuming a cache size of 512 sets, 32 Bytes direct mapped cache, the additional overhead due to *Turn-off bits* (TOBs) and *Flag bits* (FBs) will be 512 bits each. This is equivalent to 128 bytes or the overhead equal to that of 4 additional cache blocks. Thus, with 0.33 pJ per block per cycle, the total leakage energy overhead will be 1.32 pJ per cycle.
One may look at options to reduce this overhead. These bits are used to precisely control the cache blocks under execution, since a state destroying mode is used. An alternative could be to just turn off all blocks (as done in Zhang et al., [9]). A comparison with such a scheme is provided in Table 4. It can be seen that there is small variation in the miss-rate (number of misses), and IPC, but they are not significant, and the leakage energy saved is almost similar.

Table 2. Comparison of IPC and iL1 miss rates of various programs

Bench marks	Parameter	8 KB Initially-on	16 KB Initially-on	32 KB Initially-on	64 KB Initially-on	16 KB Initially-off	16 KB Base Cache
apsi	IPC	0.9354	0.9701	1.0153	1.0417	0.9701	0.9701
	miss_rate	0.0722	0.0145	0.0065	0.0021	0.0145	0.0145
applu	IPC	1.6105	1.6267	1.6420	1.6420	1.6267	1.6267
	miss_rate	0.0026	0.0013	0.0000	0.0000	0.0013	0.0013
unepic	IPC	1.6098	1.6120	1.6161	1.6170	1.6120	1.6120
	miss_rate	0.0005	0.0004	0.0003	0.0002	0.0004	0.0004
rawcaudio	IPC	1.4603	1.4604	1.4604	1.4604	1.4604	1.4604
	miss_rate	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
rawdaudio	IPC	1.6559	1.6560	1.6560	1.6560	1.6560	1.6560
	miss_rate	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
epic	IPC	1.6416	1.6421	1.6434	1.6438	1.6421	1.6422
	miss_rate	0.0001	0.0001	0.0000	0.0000	0.0001	0.0001
swim	IPC	1.5174	1.6632	1.6632	1.6632	1.6632	1.6632
	miss_rate	0.0138	0.0000	0.0000	0.0000	0.0000	0.0000
art	IPC	0.8078	0.8079	0.8079	0.8083	0.8079	0.8079
	miss_rate	0.0001	0.0001	0.0001	0.0000	0.0001	0.0001

4. Estimating DBEC Performance

Let
'L' represent the number of lines or blocks that the cache contains,
'B_i' represent the number of cache blocks or cache lines occupied by the i^{th} basic block,
'T_p' represent the total duration of the program execution,
't_i' represent the time for executing basic block 'i',
'I_i' the number of instructions in basic block 'i',
'p_i' the Cycles / Instruction (CPI) for the basic block 'i'
'τ' the time period of one cycle.

The total leakage energy spent in a conventional cache is proportional to the product of number of cache lines and the time (T) necessary for executing the program and is given by,

Total leakage energy spent = $k * L_N * T_p$ (2)
where, k is the leakage energy per block per cycle.
Leakage energy '*LE*' spent in the cache while executing basic block 'i' is given by,

$$LE_{spent\ i} = k * B_i * t_i$$

or $$LE_{spent\ i} = k * B_i * (I_i * p_i * \tau) \quad (3)$$

If the basic block is enclosed in a loop that is executed 'n_i' times, then the leakage energy spent can be expressed as:

$$LE_{spent\ i} = k * B_i * (n_i * I_i * p_i * \tau) \quad (4)$$

Now, in the DBEC scheme, after the execution of one basic block or loop, the cache block(s) occupied by the basic block is(are) turned off. Hence, for initially-off case, normally when a basic block is in execution, the cache block corresponding to that basic block alone is turned on. However, the last cache block of the previous basic block may also be turned on, making the number of cache blocks turned on as (B_i + 1). Thus, the leakage energy saved after 'm' basic blocks are executed, can be computed as:

$$LE_{saved\ for\ init\text{-}off} = k\left[L_N * T_p - \sum_{i=1}^{m}\left((B_i + 1) * n_i * I_i * p_i * \tau\right)\right] \quad (5)$$

The estimation for the initially-on case, is a little more complicated. Since, all the cache blocks are initially on, they have to be occupied /used by the program, to be brought under the control of DBEC and then turned off. Since this is not deterministic, it is difficult to estimate the leakage energy spent. However, a worst case estimate may be obtained.

Initially, the leakage energy spent by basic block 'i' may be computed as:

$$LE_i = k * Max\left[\left(L_N - B_{i-1}\right), B_i\right] * t_i \quad (6)$$

The (L-B_i) term in the expression corresponds to the number of blocks that are likely to be active, when B_i is under execution after B_{i-1} blocks have been turned off. However, if B_i is greater than this value, then B_i number of blocks will be turned on. At this point, all the cache blocks are under the control of DBEC. Hence, for all the subsequent basic blocks, the leakage energy spent will be the same as for the initially-off case; i.e.

$$LE_i = k * B_i * t_i \quad (7)$$

Hence, energy saved for the entire program consisting of 'm' basic blocks will be given by:

$$LE_{saved\ for\ init\text{-}on} = k * L_N * T_P - \sum_{i=1}^{m} LE_{i_{init\_on}} \quad (8)$$

In order to validate this estimation, a comparison of the estimated values and the values obtained from the simulation has been done. The results for one program namely, *apsi* is given in Table 3, for both initially-on and initially-off case.
It can be seen that the estimated values are useful in getting the range of savings that could be obtained.

Table 3. Comparison of Estimated v/s Simulated Values (for *apsi* program)

Parameter	For *initially-off* case	For *initially-on* case
Sim_num_instructions	475045907	475045930
Sim_CPI	0.4737	0.4738
CPI (manually estimated)	0.7222	0.7222
Leakage energy spent; (from estimation)	$0.33962*10^{-12}$	$7.24533*10^{-12}$
Leakage energy spent; (from simulation)	$0.22618*10^{-12}$	$9.509*10^{-12}$
Leakage energy saved (from estimation)	99.9962%	6.4904 %
Leakage energy saved (from simulation)	99.8 %	5.72 %

5. Comparison With Related Work

It is worth contrasting the DBEC approach with the dynamic hardware based mechanism used in DRI I-cache proposed by [3}, the Adaptive Mode Control Cache proposed by [4] and the Cache Line Decay mechanism proposed by [7]. The Dynamic ResIzable I-cache '*DRI I-cache*', dynamically reacts to application demand and adapts to the required cache size during execution to reduce leakage energy of level 1 I-cache. Kaxiras et al.[7], have proposed a mechanism to control the cache leakage energy at block level granularity i.e., cache block granularity. This technique monitors the periods of inactivity in cache blocks by associating saturating counters with each block. If a cache block is not accessed within a predefined fixed interval, the block is turned-off. In an adaptive-mode-control cache, the miss rate and performance factors are dynamically monitored to adjust the turn-off interval to ensure that its performance closely tracks the performance of an equivalent cache without leakage-control mechanism.

In these schemes, the time at which a block can be put into the low-leakage mode is determined at run time based on a saturating counter. The choice of the saturation value directly affects the performance, as the saturation value is only an estimate of the non-usage of a line. Too large a value would result in reduced power saving, while too small a value would result in higher power saving, but, would result in early eviction of cache blocks, thereby causing performance degradation. The saturation value may vary widely across applications as well as within a given application. An optimal value is to be chosen based on these. Even the application dependent static choice of saturation value is only an estimate. This may very well result in performance penalty. However, in the DBEC scheme, more precise information on when a block is going to be ‘*dead*’ is exactly obtained directly from the compiler.

Bellas et al. [1] have proposed L-cache, buffers the instructions of innermost loops. However, this approach does not consider the basic blocks within loops that contain function calls, for placement in the L-cache. The profile from the previous runs is used to select the best instructions to be cached. The performance improvement causes the application to be executed in lesser time and hence energy savings can be achieved. Compiler optimizations namely loop unrolling, loop tiling, loop permutation and loop fusion are shown to improve program performance and reduce the energy spent. However, these techniques are effective only for loop intensive programs.

Zhang, et al. [9], have presented a compiler-assisted approach for placing a cache line in the low-leakage mode at a loop level granularity. This parallel and independent work is very similar to DBEC. In fact they discuss about four schemes, namely, Conservative and Optimistic schemes with state-preserving and state-destroying modes. The DBEC scheme is almost identical to the conservative state destroying scheme. Zhang et al., [9] have attempted to put the cache lines into the low leakage mode when the next access to the instruction will never occur or occur only after a long gap. The compiler is used to insert power mode instructions that control the supply voltage for the cache lines. In this scheme, instead

Table 4. Comparison of DBEC performance with alternate scheme

Program	**cache size KB**	**sim_cycle**	**IPC**	**Number of Misses**	**miss rate**	**Leakage Energy saved (%)**	**Scheme**
Art	8	2475182013	0.8080	68467	0.0000	50.686	All_blocks_off
	8	2475178569	0.8080	68231	0.0000	50.304	DBEC scheme
	64	2474501022	0.8082	25082	0.0000	90.836	All_blocks_off
	64	2474497500	0.8082	24846	0.0000	90.789	DBEC scheme
rawc	8	11478100	1.4604	701	0.0000	69.506	All_blocks_off
	8	11478185	1.4603	703	0.0000	69.509	DBEC scheme
	64	11477487	1.4604	574	0.0000	96.085	All_blocks_off
	64	11477566	1.4604	575	0.0000	96.133	DBEC scheme
rawd	8	7765251	1.6559	699	0.0000	73.386	All_blocks_off
	8	7765301	1.6559	700	0.0000	73.386	DBEC scheme
	64	7764612	1.6560	570	0.0000	96.664	All_blocks_off
	64	7764664	1.6560	571	0.0000	96.664	DBEC scheme

Epic	8	58774036	1.6416	14313	0.0001	1.571	All_blocks_off
	8	58774163	1.6416	14312	0.0001	1.570	DBEC scheme
	64	58692407	1.6439	3236	0.0000	55.649	All_blocks_off
	64	58692534	1.6438	3235	0.0000	55.649	DBEC scheme
unepic	8	10621110	1.6096	9281	0.0005	1.289	All_blocks_off
	8	10619174	1.6099	9239	0.0005	1.279	DBEC scheme
	64	10572841	1.6170	4374	0.0002	40.444	All_blocks_off
	64	10572009	1.6170	4337	0.0002	40.428	DBEC scheme

of turning off only the cache blocks storing the *dead instructions*, it attempts to turn-off the supply to all the cache blocks. The only advantage of DBEC with its precise control is that it will also take advantage of prefetch mechanism i.e., prefetched blocks will not be turned off.
The DBEC approach takes care of both loop-intensive and non loop-intensive programs without sacrificing performance. However, the DBEC scheme performs well if the program structure has more independent basic blocks.
DBEC scheme is useful for the basic blocks of Main and the other functions defined with in the program (which are annotated as explained earlier). The DBEC scheme will not consider turning-off the cache blocks occupied by the DLLs (if they are not annotated).

6. Summary

The DBEC approach presented in this chapter identifies precisely the basic blocks which are *dead* at a particular point of program execution with the help of the compiler. This mechanism works well when the program contains a large number of independent basic blocks. This approach is found to reduce the leakage energy by 10.8% for *initially on* case and 21.6 % for *initially off* case for a cache size of 16 KB. For a cache size of 64 KB, it is found to reduce the leakage energy by 18.9% for *initially on* case and 62 % for *initially off* case. The performance degradation is negligible for this approach. Hence, this approach is applicable to general purpose high performance microprocessors too. Further, this being a primarily static approach supported by the compiler, the run-time architectural overhead of this approach is also minimal.
The energy behavior of an energy-controlling strategy using software technique mainly depends on the program profile. Secondly, it depends to a large extent on the execution time of the loops also. In case of DBEC scheme, the cache lines occupied by an independent basic block/loop are turned off soon after its execution, to control the leakage energy. If the cache lines occupied by an independent basic block are turned off, it will definitely contribute to significant percentage of energy savings provided the independent basic block is before the loop in the program order and the loop is executed relatively a longer period time. The reason for significant percentage energy saving is that such cache lines remain in low leakage control mode for a large time fraction of the program execution (the general rule of Pareto principle, i.e., 20 % of the code executes for 80% of the total time). Such cases are cleverly exploited by the DBEC scheme.

References

[1] Nikolaos Bellas et al., "Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Micrprocessors," ISLPED, ACM Press, New York, USA,1998,pp70-75.
[2] Michael D Powell et al., "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," ISLPED,2000,pp 90-95.
[3] Se-Hyun Yang et al., "An Integrated Circuit / Architectural Approach to Reducing Leakage in Deep-Submicron High Performance I-caches," Proceedings of the International Symposium on High Performance Computer Architecture (HPCA),Janaury2001.
[4] Huiyang Zhou et al., " Adaptive Mode Control: A Static-Power-Efficient Cache Design," Proc. International Conference on Parallel Architectures and Compilation techniques, (PACT 01) 2001.
[5] Kanad Ghose et al., " Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line

Buffers and Bit-Line Segmentation," ISLPED, ACM Press, New York, USA, 1999,pp 70-75.
[6] Hongbo Yang et al., "Power and Energy Impact by Loop Transformations," http://research.ac.upc.es/pact01/colp/paper12.pdf
[7] Stefanos Kaxiras et al., "Cache-Line Decay: A Mechanism to Reduce Cache Leakage Power" IEEE Workshop on Power Aware Computer Systems (PACS), Cambridge, MA, USA, pp82-96.
[8] K Kristner et al., "Drowsy-Cache: A Mechanism to Control Leakage Energy in SRAM cells," Proc. ISCA 2002.
[9] W.Zhang et al., " Compiler-Directed Instruction Cache Leakage Optimization," Proc. of 35th International Symposium on Microarchitecture, Istanbul, Turkey, November2002.
[10] Mohan G Kabadi et al., " Dead Block Elimination in I-Cache: A Mechanism to Reduce Power in I-Cache of High Performance Microprocessors," Proc. International Conference on High Performance Computing (HiPC). India, Dec.2002,pp79-88.
[11] D Burger et al., "The Simplescalar Tool set Version-2.0:," CSD Technical Report #1342, University of Wisconsin-Madison, June1997.
[12] "SPEC CPU 2000 benchmark suite," http://www.spec.org
[13] C.Lee et al., "MediaBench: A Tool for Evaluating Multimedia and Commnications Systems", Proc. Micro-30,pp.330-335,Dec.1997

Mohan G Kabadi received the B.E.(Electrical Power) from University of Mysore and M.Tech. degrees in Energy System from NIT-K. He has carried out Research work in Anna University. Currently he is the Head of Computer Science in S.J.C.Institute of Technology, INDIA. His area of interest includes Embedded Systems an Low power architecture.

Dr. Ranjani Parthasarathi received the Ph.D from IIT Madras. Currently she is the Professor in the department of Computer Science, in Anna University, Chennai, INDIA. Her area of interest includes re-configurable computing, computer Networks and Network Processors.