Abstract

Active power used to be the primary contributor to total power dissipation of CMOS designs, but with technology scaling, the share of leakage in total power consumption of digital systems continues to grow. Temperature is another factor that exponentially increases the leakage current. In this paper, we show the effect of temperature on the optimal (minimum-energy-consuming) cache configuration for low energy embedded systems. Our results show that for a given application and technology, the optimal cache size moves toward smaller caches at higher temperatures, due to the larger leakage. Our results show that using a Temperature-Aware Configurable Cache (TACC), up to 61% energy can be saved for instruction cache and 77% for data cache compared to a configurable cache that has been configured for only the corner case temperature (100°C). The TACC also enhances the performance by up to 28% and 17% for the instruction and data cache, respectively.

1. Introduction

Energy consumption is an important issue for battery powered embedded systems. On-chip cache consumes almost half of a microprocessor total energy [1][2]. Energy efficient cache architecture design is thus a critical issue in the design of processor-based embedded systems. Larger cache size may improve the hit rate for some programs, at the expense of more power consumed to fetch (dynamic power) and to hold (static power) the data. Performance-oriented applications benefit from a large cache size. Energy-oriented applications however want the cache size such that the energy savings from higher hit rate outweigh the energy increase from more leakage.

Leakage (also known as static) power is becoming the dominant contributor to the power consumption of CMOS integrated circuit. Unlike dynamic power, which depends on the activity in the circuit, leakage exists as long as power supply is applied to the circuit even if there is no activity. Consequently, bigger caches dissipate higher leakage in general. In addition, leakage is becoming more significant with technology scaling. Temperature has also an exponential effect on the leakage power [3][4].

Configurable caches have already been proposed to adapt the cache to the running application so as to minimize energy [1][5][6][7][8][9]. In this paper, we show that when the temperature changes, for each execution of an application, a new cache configuration may need to be selected, due to the increase of leakage power, if the minimum energy consumption is a must. Thus we propose to equip a configurable cache with temperature-awareness property and show that by using a temperature-aware configurable cache, up to 61% energy can be saved for instruction cache and 77% energy saving for data cache. The effectiveness of such a cache in energy reduction would increase with technology scaling due to the increasing leakage.

This paper is organized as follows. In Section 2, we highlight the related works. We formulate our energy evaluation equations, introduce the tools for calculating the dynamic and static energy, and define the problem in Section 3. Then in Section 4, the architecture and configuration flow of the temperature-aware configurable cache is given. The experimental results are presented in Section 5 and finally paper is closed by conclusions.

2. Related work

Several researchers have focused on dynamic energy and have investigated cache architectures for reducing that energy by modifying the lookup procedure, such as way predictive set-associative caches [10][11]. Reactive-associative cache (RAC) [12] also uses way prediction and checks the tags as a conventional set-associative cache, but the data array is arranged like a direct-mapped cache. PSAC [13] is physically organized as a direct-mapped cache; upon a miss, a specific index bit is flipped and a second access is made to the cache using this new index.

Several other works concentrate on static energy and propose new cache architectures. Cache decay [14] dynamically turns off cache lines that have not been visited for a designated period, reducing the L1 cache leakage energy dissipation without impacting performance. Zhou [15] proposed to dynamically determine the time interval for deactivating the cache lines that are put into sleep mode. In contrast with the shutting off mechanism that loses the information in the
cache, a drowsy cache [16] keeps the unused cache line in a low power mode by lowering the SRAM supply voltage while retaining the contents of the cache. These approaches are used dynamically by monitoring the hits and misses for a cache configuration and are independent of temperature, however in our study we try to show the effect of temperature on the cache configuration selection. Therefore, these approaches are orthogonal to ours.

Recently, configurable cache architectures have also been proposed for power reduction. Kim [7] proposed a multifunction computing cache architecture, which partitions the cache into a dedicated cache and a configurable cache. Another effort is way shutdown cache methods, proposed by Albonesi [5]. Zhang [1] proposed a technique called way concatenation that tunes the cache ways between one (direct-mapped), two and four. A dynamic online scheme that combines the processor voltage scaling and dynamic cache reconfiguration was proposed by Nacul et al. [8]. Zhang et al. [6] introduced an online heuristic that dynamically adjusts the cache size in order to minimize the cache energy. None of these techniques accounts for the change in leakage in different temperatures, but to adapt to these effects analyzed in this paper, one way is to use the above reconfigurable caches.

In [17] Cai et al. consider reliability as another factor, besides performance and energy, for cache size selection in time-constrained systems. Bai et al. [18] investigate the impact of gate-oxide thickness (\(T_{ox}\)) and threshold voltage (\(V_{th}\)) on power-performance trade-offs for on-chip caches. We analyze yet another factor: the temperature.

Kaxiras et al. [19] propose a mechanism that takes into account temperature in adjusting the leakage control policy at run time. They use a hybrid decay-drowsy policy. In this method they adapt the decay mode according to the temperature and use the drowsy mode to save the leakage in long decay intervals. However, in our method we adapt the configurable cache according to the temperature and select the cache configuration with minimum energy consumption.

### 3. Problem definition

The power consumption of CMOS circuits includes dynamic, static, and short circuit power. The short circuit power is much smaller than dynamic and static power, and is negligible. Dynamic power in cache memories attributes to signal transitions in activated bit- and word-lines, sense-amplifiers, comparators, and selectors during reads/writes, while static power is due to the total amount of leakage current through inactive or OFF transistors.

Energy equals power times time. The main sources of energy consumption for evaluating the memory subsystem energy include: i) cache dynamic energy, ii) cache static energy, iii) off-chip memory access energy, and iv) energy consumption during processor stall.

The dynamic energy consumed per access is a sum of the energy spent on searching within the cache, an extra energy required for handling the writes and energy consumed by block replacement on cache miss.

Dynamic energy per cache access equals the dynamic power of all the circuits in the cache multiplied by the time per cache access; while the static energy of a cache equals static power multiplied by the time.

Dynamic energy consumption causes most of the total energy dissipation in micrometer-scale technologies and lower temperatures, but static energy dissipation will contribute an increasingly larger portion of total energy dissipation in nanometer-scale technologies and higher temperatures. We consider both types of energies.

Fetching instructions and data from off-chip memory (due to cache misses) is energy expensive because of the high off-chip capacitance and large off-chip memory storage. Additionally, when accessing the off-chip memory, the processor may stall while waiting for the instruction and/or data, and such waiting still consumes some energy. Thus, we calculate the total energy due to memory accesses using Equation 2. In our evaluation the energy is a function of configuration of cache (\(C\)), temperature (\(Temp\)) and technology (\(Tech\)). Other parameters are assumed to be constant. First, some definitions are presented.

- \(cache\_accesses(C)\): number of accesses to the cache with configuration \(C\).
- \(cache\_misses(C)\): number of cache misses with cache configuration \(C\).
- \(execution\_clock\_cycles(C)\): number of execution clock cycles with cache configuration \(C\).

The above factors are computed by running SimpleScalar [21] for the given applications with the desired cache configuration.

- \(energy\_cache\_access(C, Tech)\): energy for accessing the cache with configuration \(C\) manufactured in technology \(Tech\).
- \(energy\_cache\_block\_refill(C, Tech)\): energy for cache block refilling after a cache miss, for a cache with configuration \(C\) manufactured in technology \(Tech\).
- \(leakage\_power(C, Temp, Tech)\): the leakage power of a given cache with configuration \(C\) at temperature \(Temp\), manufactured in technology \(Tech\).

The above items are computed using CACTI [20].
• energy_off_chip_access: the energy of accessing off-chip memory when there is a miss.
• energy_uP_stall: the energy consumed when the processor is stalled waiting for the memory system to provide the missed instruction or data.

According to the explanations and experiments done in [1], we assumed:

\[ \text{energy_off_chip_access + energy_uP_stall(Tech) = 20} \]  
\[ nJ = \text{energy_off_chip_stall} \]  

Now the total energy of the memory subsystem can be calculated by:

\[ \text{energy_memory(C, Temp, Tech) = energy_dynamic(C, Tech) + energy_static(C, Temp, Tech)} \]  

where

\[ \text{energy_dynamic(C, Tech) = cache accesses(C) *} \]  
\[ \text{cache_misses(C, Tech) + energy_uP_stall(Tech) = 20} \]  
\[ \text{energy_miss(C, Tech) = energy_off_chip_stall} \]  
\[ + \text{energy_cache_block_refill(C, Tech)} \]  
\[ \text{energy_static(C, Temp, Tech) = executed_clock_cycles(C) * clock_period} \]  
\[ \text{leakage_power(C, Temp, Tech)} \]

Using these equations, the energy is calculated for the Base Configurable Cache (BCC) and our proposed Temperature-Aware Configurable Cache (TACC).

We assume that the BCC has similar architecture to the configurable cache proposed in [1]. In [1] they show that by configuring the cache for each application, they can effectively reduce energy consumption. The proposed configurable cache in [1], due to the limitations of the proposed architecture, supports only a limited set of configurations which we call them valid configurations. We propose to use the same configurable cache and convert it to a TACC. The BCC is configured for each application for the corner case leakage (i.e. leakage at 100°C), however TACC is configured for each execution of an application considering the chip temperature at that time. The energy saving obtained by using TACC compared to the BCC is calculated by the following equation:

\[ \text{Energy Saving} = \frac{\text{energy_BCC_tempN} - \text{energy_TACC}}{\text{energy_BCC_tempN}} *100 \]  

where \( \text{energy_BCC_tempN} \) is the energy consumption of the BCC when temperature is \( N°C \) and \( \text{energy_TACC} \) is the energy consumption of the TACC.

Changing the cache configuration, not only affects energy consumption but also the execution time (performance). The following equation is used to calculate the performance enhancement obtained by TACC compared to the BCC.

\[ \text{Performance Enhancement} = \frac{\text{exec_time_BCC} - \text{exec_time_TACC}}{\text{exec_time_BCC}} *100 \]

where \( \text{exec_time_BCC} \) is the execution time of the application with the BCC cache and \( \text{exec_time_TACC} \) is the execution time with TACC. So in our experiments, the optimization problem is defined as follows:

“For a given application, processor architecture, technology, and valid configurations of the configurable cache, find the valid cache configuration that results in minimum energy consumption in different temperatures over the entire application run.”

4. Proposed approach

There are two issues regarding reconfigurable devices: i) when and ii) how to reconfigure. This section mainly discusses about these two issues for TACC.

4.1. Reconfiguration Flow

From the cache reconfiguration point of view different temporal granularities are available such as: i) for each application (e.g. [1] and [6]), ii) for each execution of application, and iii) at each phase of application [25]

In our case, the cache is configured for each execution of the application considering the temperature at that time. Therefore, the configuration is fixed for the whole execution of the application. Because the target of TACC is embedded systems, the execution time is most often not very long. Moreover, the change in the temperature is not very fast. Consequently, we believe that this temporal granularity is reasonably enough, since as we will show it does not impose much hardware overhead and performance penalty.

The reconfiguration flow shown in Fig. 1 is used for reconfiguring the cache. The flow consists of two phases: evaluation phase (which is done offline) and reconfiguration phase (which is online). In the evaluation phase, the information of static and dynamic energy for different configurations of the cache in different temperatures, and the results of running the target application(s) for valid cache configurations (cache miss, cache hit, execution time), which can be obtained via running the application on an instruction set simulator (ISS), are used and then the lowest energy valid cache configuration for different temperatures are determined.

The target temperatures and their corresponding proper cache configurations are recorded in a software table (similar to Table 2 and 3, explained in Section 5) in the operating system (OS). Then, in the reconfiguration phase, before executing the application, first the temperature is measured. According to the temperature, the suitable configuration is read from the software table and loaded into the reconfigurable cache, and then the application is executed.
4.2. Reconfiguration Flow
Various configurable cache architectures have been proposed in [1], [5], [6], [7], [8], [9]. The degrees of freedom for reconfiguring the cache can be size, number of ways, line-size and replacement policies. Since many configurable cache architectures have already been proposed, we do not focus on the details of the configurable cache microarchitecture; instead, we focus on how to convert an available configurable cache to a TACC. We select [1] as the base configurable cache in which the size, number of ways and line-size of the cache are configurable and unlike [9] we assume replacement policy is fixed and is LRU (least recently used). In [1] the following configuration changes are applicable: i) changing the size of cache is done by way shutdown where cache ways can be selectively turned off, ii) changing the number of ways is done via a technique called way concatenation that concatenates every two ways into a single way (halves the number of ways), iii) changing line size is done by treating multiple physical cache lines as a single one via the cache controller.

Therefore, for a base configurable cache with size \( N \) (in KB) and 4-way, the valid configurations are \( N \) (4-, 2-, and 1-way), \( N/2 \) (2-, and 1-way), \( N/4 \) (1-way) while for each of these six valid configurations the line size can be three different values (e.g. 32-, 16-, and 8-byte). Therefore, totally there are 18 valid configurations.

To convert this configurable cache to a TACC, we only need to stick a thermal sensor on the top of the package of the embedded processor chip. Moreover, an accessible port should be added to the system so that the processor can read the sensor output (e.g. a register with a specified address).

It is noteworthy that due to the junction-to-ambient thermal resistance (\( \theta_{JA} \)) of the chip package, the temperatures of the package surface and inside of the chip are not equal. The inside temperature can be calculated using the following equation:

\[
T_j = T_a + \theta_{JA} \cdot P \quad (8)
\]

where \( T_j \) is junction temperature, \( T_a \) is ambient temperature, and \( P \) is power consumption. For each embedded processor and package, the \( \theta_{JA} \) and \( P \) are known. \( T_a \) is measured using the thermal sensor. Therefore, \( T_j \) can be calculated. According to our experiments (Section 5), 20°C ~ 40°C are suitable temperature resolutions for reconfiguring the TACCs in 70nm technology node. Therefore, accurate temperature detection is not needed. Moreover, our study on values of thermal resistance for different packages shows that \( \theta_{JA} \) ranges between 7°C/W ~ 35°C/W [23]. Table 1 shows the power consumption for two widely used embedded microprocessors of ARM [24].

<table>
<thead>
<tr>
<th></th>
<th>ARM7TDMI</th>
<th>ARM966E-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>130nm</td>
<td>7.98 mW</td>
<td>62.5 mW</td>
</tr>
<tr>
<td>Frequency</td>
<td>236 MHz</td>
<td>250 MHz</td>
</tr>
<tr>
<td>90nm</td>
<td>7.08 mW</td>
<td>51.7 mW</td>
</tr>
<tr>
<td>Frequency</td>
<td>236 MHz</td>
<td>470 MHz</td>
</tr>
</tbody>
</table>

5. Evaluation results
We use applications from Mibench [22] benchmark suite. As mentioned, SimpleScalar and CACTI (version 4.1) are used as our simulation tool and power modeling tool, respectively. The cache hit is assumed to take one clock cycle and cache miss 100 cycles. The clock frequency of the single-issue base processor is assumed to be 200 MHz. Our experiments are done for six different temperatures: 0°C, 20°C, 40°C, 60°C, 80°C and 100°C, and the target technology is 70nm where \( V_{dd} \) is 0.9V. We assume that the size of the BCC is 16KB, number of ways is 4 and line sizes are 8-, 16-, and 32-bytes, hence the following configurations are valid: 16KB (4-, 2-, 1-way), 8KB (2-, and 1-way), and 4KB (1-way). The line size for each of the configurations can be 8-, 16-, or 32-byte.
In these experiments, for each application we run Simplescalar for all the 18 configurations. Then we extract the data for dynamic energy and static energy for all the 18 cache configurations for six different temperatures and 70nm technology using CACTI (18×6 = 108 cases). Using these data and equations of Section 3, for each application we find the cache configurations that minimize Equation 2 for different temperatures. Because the same configurable cache is used for BCC and TACC, the overhead of configurability is not applicable.

Table 2 and 3 respectively show the configurations for data and instruction caches which result in minimum energy for six different temperatures. Since BCC corresponds to the best configuration at 100°C, last line of each table also shows the BCC configuration. To study the effect of using TACC individually on data and instruction caches, the instruction cache configuration is assumed to be fixed when data cache is a TACC and vice versa. In the triple used for presenting the cache configuration in the tables, the first number shows the cache size in kilo bytes, the second number shows the line size in bytes and the last one specifies the number of ways. This table is the same lookup table that is used for reconfiguring the cache at the reconfiguration phase (Section 4.1). According to the results, as the temperature is increasing (larger leakage power) the optimal cache is moving to smaller sizes. For some applications (e.g. qsort and sha) the data cache changes among three different cache sizes and also the instruction cache of gsm needs 4 different configurations when temperatures changes from 0°C to 100°C.

Fig 2 shows the energy saving obtained by using a TACC compared to a BCC for data cache for three different temperatures (0°C, 20°C, 60°C). For example, for qsort, by reconfiguring the data cache from (4KB, 32, 1) (i.e., the configuration with minimum energy at 100°C or the configuration of BCC, Table 2) to (16KB, 32, 2) (i.e., the configuration with minimum energy at 0°C, Table 2) 49% energy is saved, when the temperature is 0°C. The maximum energy saving is for adpcm (76.5%@0°C). The average-DC in Fig. 2 shows the average energy saving for data cache (35.88%@0°C). The max-IC (60.87%@0°C), and average-IC (16.33%@0°C) are maximum and average energy savings for instruction cache, respectively.

The results show that the TACC is more effective for data cache compared to instruction cache. This is because, the number of accesses to data cache is less than instruction cache, therefore the leakage power has more contribution in the total energy of the data cache.

As the temperature decreases, the contribution of static energy in the total energy decreases and dynamic energy becomes more effective. Therefore, in low temperatures, larger cache sizes are selected which cause less number of cache misses and off-chip memory accesses and at the same time reduce the execution time and improve the performance. Fig. 3 reports the performance enhancement obtained by TACC compared to BCC for three different temperatures. Although the maximum performance improvement for instruction cache (28%@0°C) is more than data cache (17%@0°C), the average of data cache (8.1%@0°C) is more than instruction cache (5%@0°C).

6. Conclusions

We proposed a Temperature-Aware Configurable Cache, TACC, in which the cache configuration is selected according to the temperature before each execution of the application. The suitable configurations for different temperatures are stored in a look up table in an offline phase and are used to reconfigure the cache before executing each application.

Our studies show that: i) as the leakage power is increasing in future technologies, using a temperature-aware configurable cache will be very important for reducing energy in low energy embedded systems. Our results show that up to 61% (17% on average) energy consumption can be saved in 70nm technology for instruction cache when the temperature changes from 0°C to 100°C, ii) since the smaller caches are more suitable
for low energy systems in finer technologies and higher temperatures, finding an optimal cache configuration that simultaneously optimizes performance and energy is increasingly more difficult in future, specially at high temperatures, iii) since the accesses to data cache are less than the accesses to instruction cache, the data cache is more easily affected by temperature than instruction cache. By using a configurable data cache, up to 77% (36% on average) energy can be saved in 70nm technology, iv) the TACC not only can save the energy but also improves the performance. The performance improvement for instruction cache is up to 28% (5% in average) and for data cache, it is up to 17%.

7. Acknowledgment
This research was supported in part by Grant-in-Aid for Encouragement of Young Scientists (A) 17680005. Also it is supported by Institute of Systems & Information Technologies/KYUSHU, VDEC, The University of Tokyo with the collaboration of STARC, Panasonic, NEC Electronics, Renesas Technology, Toshiba, Core Research for Evolutional Science and Technology project of Japan Science and Technology Corporation.

8. References

Table 2: Data cache configurations (cache size, line-size in bytes, and the number of ways) with minimum energy.

<table>
<thead>
<tr>
<th>Temperature</th>
<th>qsort</th>
<th>djpeg</th>
<th>lame</th>
<th>dijkstra</th>
<th>patricia</th>
<th>sha</th>
<th>adpcm</th>
<th>Crc</th>
<th>fft</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°C</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 4</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 4</td>
</tr>
<tr>
<td>20°C</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 4</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 4</td>
<td></td>
</tr>
<tr>
<td>40°C</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 4</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 2</td>
<td>4K, 32, 1</td>
<td>8K, 32, 2</td>
<td>16K, 32, 4</td>
</tr>
<tr>
<td>60°C</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 4</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>4K, 32, 1</td>
<td>4K, 16, 1</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
</tr>
<tr>
<td>80°C</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>4K, 32, 1</td>
<td>4K, 16, 1</td>
<td>4K, 32, 1</td>
<td>8K, 32, 2</td>
</tr>
<tr>
<td>100°C</td>
<td>4K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>4K, 32, 1</td>
<td>4K, 32, 1</td>
<td>8K, 32, 2</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Instruction cache configurations with minimum energy

<table>
<thead>
<tr>
<th>Temperature</th>
<th>qsort</th>
<th>djpeg</th>
<th>lame</th>
<th>Dijkstra</th>
<th>blowfish</th>
<th>rijndael</th>
<th>gsm</th>
<th>fft</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°C</td>
<td>16K, 8, 4</td>
<td>16K, 8, 4</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
<tr>
<td>20°C</td>
<td>16K, 16, 4</td>
<td>16K, 16, 4</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
<tr>
<td>40°C</td>
<td>16K, 16, 4</td>
<td>16K, 16, 4</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
<tr>
<td>60°C</td>
<td>16K, 16, 4</td>
<td>16K, 16, 4</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
<tr>
<td>80°C</td>
<td>16K, 32, 4</td>
<td>16K, 32, 4</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
<tr>
<td>100°C</td>
<td>16K, 32, 4</td>
<td>16K, 32, 4</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>8K, 32, 2</td>
<td>16K, 32, 2</td>
<td>16K, 32, 2</td>
<td>8K, 32, 1</td>
</tr>
</tbody>
</table>