The 41st IEEE/ACM International Symposium on Microarchitecture. 2008. Lake Como, Italy. pp. 459-470.

Reconfigurable Energy Efficient Near Threshold Cache Architectures

Ronald G. Dreslinski, Gregory K. Chen,
Trevor Mudge, David Blaauw, Dennis Sylvester
University of Michigan - Ann Arbor, MI
{rdreslin,grgkchen,tnm,blaauw,dennis}@eecs.umich.edu

Krisztian Flautner
ARM, Ltd.
Cambridge, United Kingdom
krisztian.flautner@arm.com

Abstract

Battery life is an important concern for modern embedded processors. Supply voltage scaling techniques can provide an order of magnitude reduction in energy. Current commercial memory technologies have been limited in the degree of supply voltage scaling that can be performed if they are to meet yield and reliability constraints. This has limited designers from exploring the near threshold operating regions for embedded processors.

Summarizing prior work we show how proper sizing of memory cells can guarantee that the memory cell reliability in the near threshold supply voltage region matches that of a standard memory cell. However, this robustness comes with a significant area cost. We show how to employ these cells to build cache architectures that greatly reduce energy consumption. We propose an embedded processor based on these new cache architectures that operates in a low power mode, with minimal impact on full performance runtime. The proposed cache uses near threshold tolerant cache ways to reduce access energy combined with traditional cache ways to maintain performance. The access policy of the cache ways is then dynamically reconfigured to obtain energy efficient performance while minimally impacting the high performance mode runtime. Using near threshold cache architectures we show an energy reduction of 53% over a traditional filter cache. For the MIBench embedded benchmarks we show on average an 86% (7.3x) reduction in energy while in low power (10MHz) mode with only an average 2% increase in runtime in high performance (400MHz) mode. And for SpecInt applications we show a 77% (4.4x) reduction in energy in low power mode with only an average 4.8% increase in runtime for high performance mode. In addition we show that these trends hold from 130nm to 45nm technology nodes.

1. Introduction

Power has become a first class design constraint, particularly in processors for embedded applications. New mobile devices are demanding more processing capabilities, but battery lifetime still remains a critical concern. That creates a need for energy efficient embedded processors capable of handling a range of application or task performance demands. The goal of our study is then to design a core that has a high performance mode that can complete time critical or compute intensive tasks quickly, but also provides a low power mode that runs slowly and can complete non-critical tasks in an energy efficient manner.

The use of near threshold and subthreshold supply voltage scaling has been the focus of many different studies. Zhai et al. [26] proposed a subthreshold sensor network processor and Wang et al. [21] designed a subthreshold chip for FFT applications. Most of these applications are small and require little cache/memory space. If these techniques are going to be used for larger embedded applications that benefit from significant amounts of cache, then several issues need to be addressed. Most importantly, because the core and memory exhibit very different requirements for voltage scaling, they will need to be handled differently. First logic tends to have a higher activity factor than memory, leading to a different optimal supply voltage for a given frequency target. This justifies the need for a separate supply voltage domain in order to achieve the optimal energy efficiency [9, 24]. Second logic is more robust under supply voltage scaling, needing very little additional sizing to maintain yield and function reliably. Memory cells on the other hand require substantial resizing in order to maintain reliable operation at low voltages [25]. Chen et al. [8] describes a method of determining the cell sizes necessary to maintain the same reliability levels for several different SRAM cell topologies.

Our study is concerned with the near threshold operating region, which is just above 400mV for the 130nm technology node of our study (We also present results for 90nm, 65nm and 45nm). We do not target subthreshold designs. The energy savings in the subthreshold region is limited by the fact that devices slow down exponentially with decrease in threshold voltage, Figure 2. This leads to extremely slow operating frequencies that show diminishing returns in energy

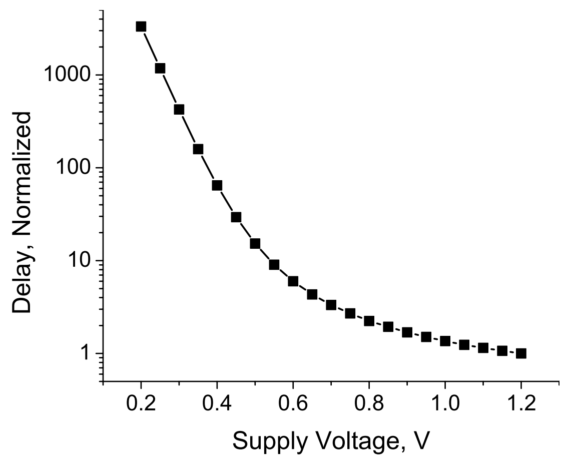


Figure 1. Impact of supply voltage scaling on delay.

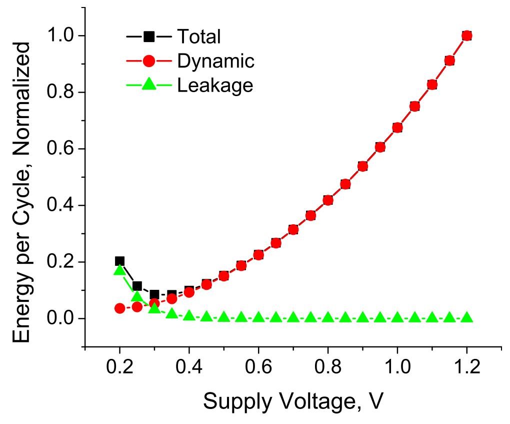


Figure 2. Impact of supply voltage scaling on energy.

savings. As the voltage is scaled even deeper into the subthreshold region the total energy to complete the task becomes larger, Figure 1. This happens when the leakage energy of the circuit begins to dominate the total energy consumption resulting in a net increase in energy for all lower supply voltages [6, 22]. In addition to this, the required cell sizes for reliable SRAM operation become extremely large allowing only small memories on chip.

Using the work done by Chen [8] as a starting point, we explore memory hierarchies in which some or all of the memory cells in the system are designed to scale their supply voltages into the near threshold operating region. This will allow for increased energy savings in low power mode. We then revisit the idea of a filter cache [17] in the context of near threshold operation and supply voltage scaling. We show that a near threshold filter cache reduces the energy of a traditional filter cache by 36%. However, the filter cache has the potential to impact the overall performance of the system in the high performance mode by increasing the runtime by nearly 20%.

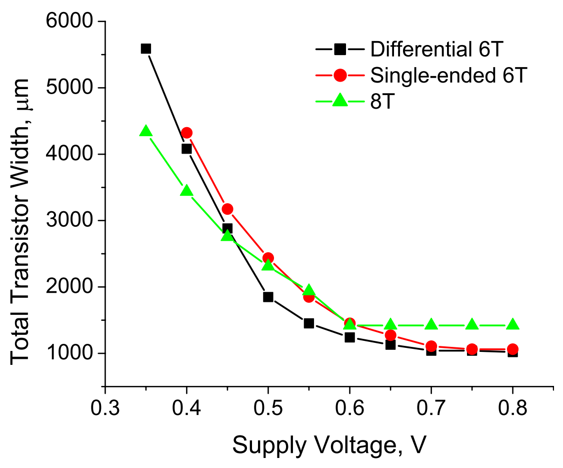


Figure 3. SRAM Cell area (measured in total transistor width) at different supply voltages to maintain ISO-Robustness.

To overcome this we propose a new cache architecture, the Reconfigurable Energy-efficient Near Threshold (RENT) cache, that reduces the runtime overhead in high performance mode while maintaining considerable energy savings in low power mode. The cache is composed of one near threshold tolerant cache way and several standard SRAM cache ways. The near threshold tolerant cache way is accessed in the first cycle and only on a miss are the other cache ways accessed. In this manner the near threshold tolerant cache way acts as a shield to the other cache ways. If the miss rate in the near threshold tolerant cache way exceeds a threshold, then the cache is dynamically reconfigured to access all the cache ways in parallel, providing a single cycle access to all of the cache space. This will help to almost eliminate any runtime increase in high performance mode. By using this technique combined with some advanced access policies the RENT cache shows a 53% reduction in energy over a traditional filter cache. This results in a system that provides a 86% (7.3x) energy savings in low power mode with only an average 2% increase in runtime in high performance mode.

The rest of the paper is organized as follows. First, Section 2 will summarize the essential points necessary to characterize near threshold tolerant memories. Section 3 will present the proposed architectural solutions that employ these near threshold tolerant techniques. Section 4 will describe the methodology, and Section 5 will present the results. We then finish the paper with a brief discussion of related work in Section 6, and provide concluding remarks in Section 7.

2. Low Voltage Tolerant SRAM cell design

Supply voltage scaling has been shown to be an effective way to handle lowering the energy consumption of logic circuits. There have been several cores built that show a quadratic savings in dynamic energy consumption with a delay degradation [21, 26]. These designs employ CMOS circuitry, which is quite resilient to supply voltage scaling. SRAM memory cells, on the other hand, do not voltage scale into the near and subthreshold regions as easily. Several ultra-low energy designs have been implemented [7, 16, 20, 23] but require increased cell size. This increased cell size reduces the energy gain from supply voltage scaling, because the larger cells consume more power than the smaller cells. In addition, when the die size is held constant, the increased cell size reduces the total overall size of the memory structure. This increased cell size is done in order to maintain reliability levels, but until recently there was no formal method to guarantee reliable operation in near threshold memory structures.

By modeling process variation, especially random dopant fluctuations (RDF), and employing a modified Monte Carlo analysis, Chen et al. showed how to calculate the robustness of an SRAM cell design at near and subthreshold supply voltages [8]. Because SRAM cells occur in large arrays, failure rates must be kept very low to ensure a reliable cache. For example, for 99% of 8kB caches to be reliable it would require a bitcell failure rate of less than 1.57 x 10^{-7}.

Among other things, the analysis of [8] calculates the necessary cell size to maintain the same failure rate as a standard SRAM cell, termed iso-robustness. In determining the iso-robust point the transistors are scaled to avoid read, write, and read-upset failures. The technology node for our study is 130nm where standard SRAM cells operate with acceptable reliably down to 800mV. Later we present results for 90nm, 65nm, and 45nm. For our robustness analysis cells that operate at supply voltages below 800mV are required to be sized to match the failure rate of a standard SRAM cell operating at 800mV. A plot of several different SRAM topologies is shown in Figure 3. The graph presents the necessary cell-area in terms of total transistor width to maintain iso-robustness for a given supply voltage. The three lines in the plot represent a standard 6T SRAM cell, a single-ended 6T SRAM cell [23] and a 8T SRAM cell [20]. It is possible to not only pick the proper supply voltage to meet the delay requirements, but to also pick the SRAM topology that provides the most area efficient solution.

This analysis has several ramifications for the design of a low-power embedded microprocessor. The SRAM in these systems must be designed for reliable operation. Using this analysis we can determine what size to make an SRAM cell in order to have it operate reliably down to a given supply voltage. The increased cell size will affect performance of a cache in two main ways. First at full supply voltage the SRAM cell will consume more energy and be slightly slower due to the increased size. Body-biasing techniques can be employed to regain speed, but at the cost of additional energy consumption. Secondly, there will be less total cache in the same die space. If we are given a design frequency to target,

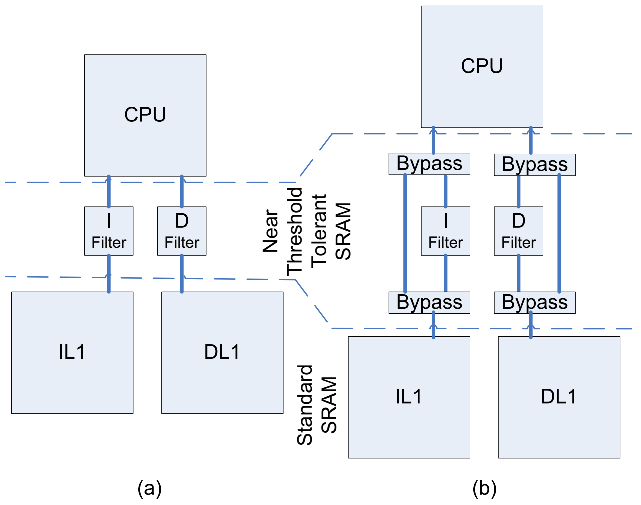


Figure 4. Near threshold filter cache architectures (a) with and (b) without bypass networks.

it is critical that we perform the proper analysis to avoid oversizing the SRAM cell and reduce the performance of the chip in non-power saving mode. Conversely it is important not to undersize the cell and increase reliability problems in power savings mode.

3. Energy Efficient Cache Architectures

The ability to voltage and frequency scale cores has provided an order of magnitude savings in energy [21, 25, 26]. However, supply voltage scaling of the memory system has been limited by the tolerance of the SRAM cell. If we use the SRAM cells discussed in Section 2, in combination with a voltage and frequency scaled core, even more energy savings could be possible. The goal of our design is to create a chip that can operate in two distinct modes. The first is a high performance mode where the system is fully powered and the core runs at a high frequency. The second mode is a power savings mode. In this mode the chip is frequency scaled to a slower performance target, and the supply voltage can then be dropped to reduce energy consumption.

Due to the differing activity factors for logic and memory, the core and caches need to be operated in different voltage domains [9, 24]. In addition, providing two memory voltage domains will allow more complex architectures that involve both near threshold tolerant SRAM, and standard SRAM. The near threshold tolerant SRAM will be used to create energy efficiency, while the standard SRAM will help to provide larger capacity on chip since they will not need to have their size increased to maintain reliability. This requires the addition of at most two additional supply voltage domains to the chip and associated voltage level converters. A diagram of the regions in which a chip might be partitioned is shown in Figure 4(a and b). In the figure, the core is operated in one supply voltage domain, the filter cache in a separate near threshold tolerant SRAM cell domain, and the L1 in a third domain. Level converters are used to cross the dotted lines separating the voltage domains.

These changes to the chip and SRAM cell design provide opportunities to explore more energy efficient cache hierarchies. Simply using near threshold tolerant cells in place of current cells will allow the caches in a system to scale to much lower supply voltages, creating a dynamic energy savings at low frequencies. However the increase in the cell size will drastically reduce the total amount of cache available when the die size is held constant. This will negatively impact performance by increasing the number of off-chip accesses, which require large amounts of energy and have a long latency. This increase in latency will in turn slow the completion of the program, and prolong the leakage energy. At high frequencies, when the cache is operating at full supply voltage the energy per access of the cache will also be increased due to the larger cell sizes required to meet the iso-failure conditions set forth in Section 2 and will need to be operated at a higher supply voltage or body bias in order to operate at the same speed. In the following sub-sections three techniques will be discussed for better use of the near threshold tolerant SRAM cells to reduce energy consumption.

3.1. Near Threshold Filter Cache

Filter caches, a technique proposed in [17, 19], is a method to reduce cache energy. The idea behind filter caches is to place a small, low energy per access, cache in between the processor and the L1 cache. This cache then filters access to the larger, more energy hungry, L1 cache. Employing near threshold tolerant SRAM cells to design a filter cache would further reduce the energy even more. A diagram of what a near threshold filter cache architecture would look like is shown in Figure 4(a).

The main drawback to using filter caches is that if the memory access pattern creates a high miss rate in the filter cache, then the overall system energy goes back up due to the additional miss that causes an access to the L1 cache. The system performance is also degraded because all cache accesses that miss in the filter cache but hit in the L1 cache take two cycles instead of one. To overcome these drawbacks a simple bypass network could be employed to bypass the filter cache when the miss rate in the filter cache is too high. Figure 4(b) provides an illustration of what a bypass filter cache would look like. This bypass filter cache also presents some drawbacks. First, in order to save energy the filter cache needs to operate with a writeback policy instead of a writethroough. This means when the filter cache is bypassed, the entire filter cache must be flushed and written back. In addition, it is difficult to track when the bypass network

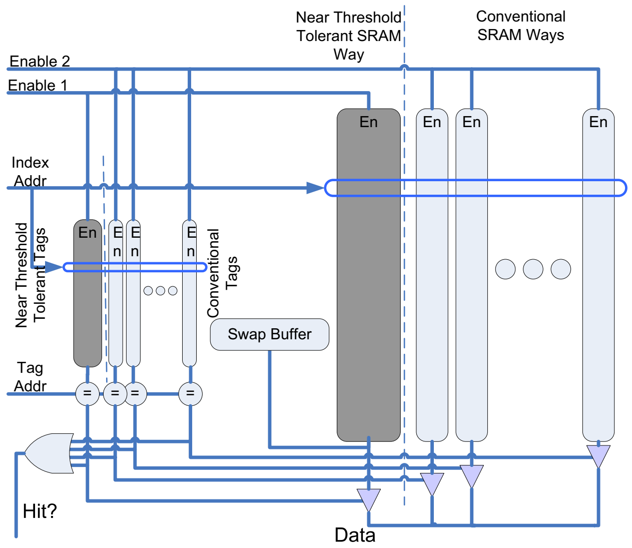


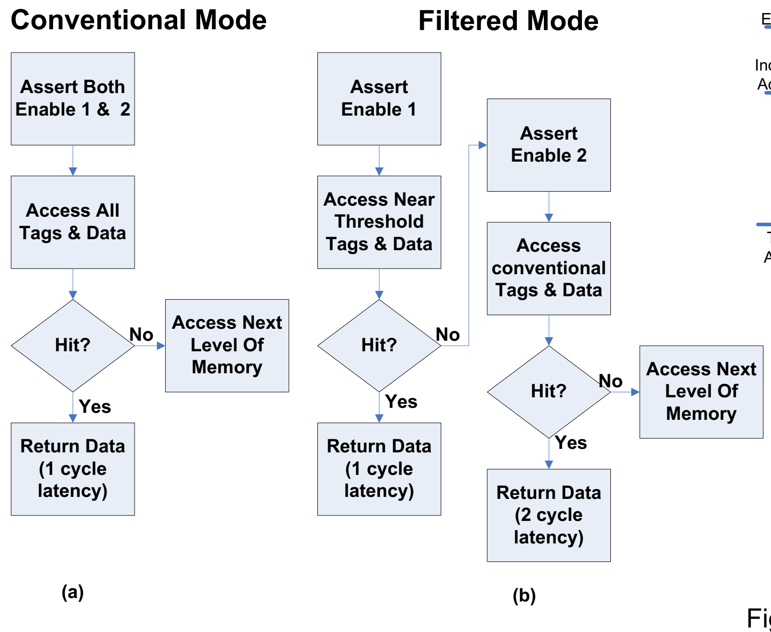
Figure 5. Cache architecture of the RENT cache.

should be turned off, since it is difficult to approximate performance of the filter cache. These two complications prevent more aggressive dynamic adjustment of the use of the bypass network.

3.2. Reconfigurable Energy-Efficient Near Threshold (RENT) Caches

Although naively employing a near threshold filter cache provides great energy savings, there are new architectures that can provide even further savings. An additional drawback of the bypass filter cache design is that when the filter cache is being bypassed, a portion of the cache space is not being used. That means there is effectively less total cache space on the chip, leading to a larger number of off-chip accesses. These off-chip accesses require large amounts of energy and have a long latency. These longer latencies lead to an increase in runtime for the program, prolonging leakage energy and degrading performance. In order to minimize the amount of cache space lost and the performance degradation that occurs from using near threshold tolerant SRAM cells, while still maintaining the energy efficiency of the filter cache, a new cache architecture is proposed based on the work done in [1, 2, 14, 18].

The proposed architecture is called the Reconfigurable Energy-efficient Near Threshold (RENT) cache. A diagram of the cache structure is presented in Figure 5. The basic premise behind the cache is as follows: There is one way of the cache and tags that is implemented with near threshold tolerant SRAM cells. The other ways of the cache and tags are implemented with standard SRAM cells. The cache will be operated in two distinct modes. We call these modes conventional and filtered. In the conventional mode the cache



Conventional Mode
Assert Both Enable 1 & 2
Access All Tags & Data
Hit?
No
Access Next Level Of Memory
Yes
Return Data (1 cycle latency)
(a)
Filtered Mode
Assert Enable 1
Access Near Threshold Tags & Data
Hit?
No
Yes
Return Data (1 cycle latency)
Assert Enable 2
Access conventional Tags & Data
Hit?
No
Access Next Level Of Memory
Yes
Return Data (2 cycle latency)
(b)

Figure 6. RENT cache access policy flow charts for (a) conventional mode and (b) filtered mode.

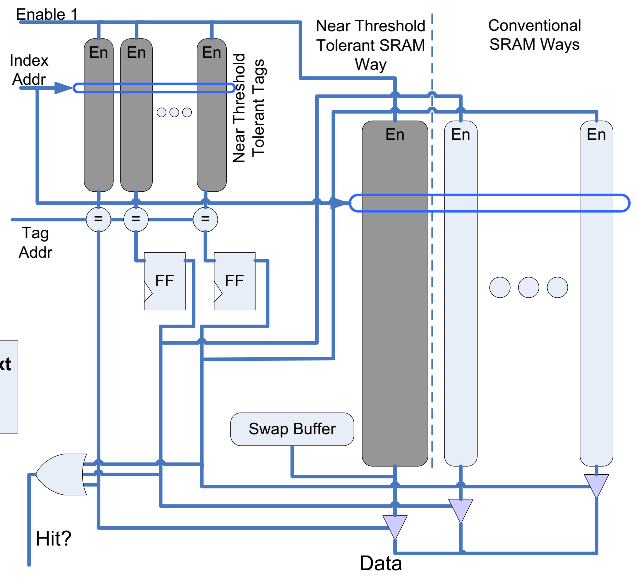


Figure 7. Architecture of the alternative RENT cache.

is accessed in the regular manner. That is, the index portion of the address reads out all the cache tags and data from each way in parallel. The tags are then matched with the incoming address and the appropriate way of the cache is enabled to the output data bus. The new mode of operation, filtered mode, is designed to act like a filter cache. When filtered mode is in use only the first way, the near threshold tolerant way, of the cache and tags are accessed on the first cycle, via the enable 1 signal. If there is a hit, the cache will return the data and energy will be saved by only accessing this one way of the cache. If there is a miss on this first cycle access, then the data and tag from the near threshold tolerant way is stored in a swap buffer. The enable 2 signal is then used and the rest of the cache is checked for a hit. If a hit is found it is both gated onto the data bus and written to the near threshold tolerant way along with its tag. On a subsequent cycle the data and tag stored in the swap buffer is then written into the cache way where the hit occurred. This action essentially swaps the value in the near threshold tolerant way and the way in which there was a hit. This ensures that the most recently used (MRU) block is in the near threshold tolerant cache way. This swapping action is similar to the writeback buffer in the filter cache, and the design is no more complicated than creating an exclusive filter cache. Figure 6 shows a flowchart of the two access modes. This addresses two of the drawbacks of the bypass filter cache. First, it utilizes the low voltage tolerant SRAM cells, even when in high performance mode leading to more on-chip cache. Second, it prevents the flushing action required when switching modes, since the data in the cache ways is still available in high performance mode, unlike the filter cache when it is bypassed. It does however present to trade-offs. First, the near-threshold tolerant cache ways will consume more energy at full voltage because they are larger with more capacitance and need to be operated with body-biasing to achieve the same speed. Second, the swap mechanism requires an additional cycle to complete some cache accesses, prolonging execution time in filtered mode.

Changing from conventional mode to filtered mode could be identified explicitly by the programmer, but greater savings could be achieved if the cache was allowed to choose operating modes dynamically based on performance. This would allow the cache to adapt to different program phases. It is relatively easy to decide when to switch from filtered mode to conventional mode. A counter can be used to monitor the hit rate of the near threshold tolerant way, when the hit rate drops below a predefined threshold the cache changes modes. It is harder to decide when to transition back to the filtered mode of operation. In order to determine when to switch, we make use of some information from the replacement policy. If the cache is using a pseudo least recently used (LRU) replacement policy we can easily determine the MRU block. This is due to the fact that the pseudo-LRU policy always identifies the MRU block in the cache tags. If we track the number of times the cache hits on an MRU block and compare it to the number of times the cache hits on any non-MRU block we can calculate what the miss rate would have been for the near threshold tolerant cache way. This follows because the MRU block would be in the near threshold tolerant cache way in filtered mode. When this hit rate exceeds some threshold, we can switch back to filtered operation. In both cases, after a switch has occurred a suitable number of accesses must be completed to allow the cache to reach a steady state before changing modes again. These methods incur very little overhead, just two counters to track accesses and MRU/filter hits. A similar approach can be found in the work done by Dropsho et al. [10] This addresses the final drawback of the bypass filter cache, by allowing more aggressive mode transitions.

3.3. Alternative RENT Caches

Further possible improvements can be made to the RENT cache to reduce energy consumption. Notice that in filtered mode we access the additional ways of the cache in the second cycle. If instead we had used the first cycle to check not only the first set of tags, but all the other cache tags in the set as well, we could know by the second cycle the way in which the data we seek resides. The modified architecture is presented in Figure 7. In this case we make all the tags using near threshold tolerant SRAM cells and access them all on the first cycle. In parallel with the tag check we access the near threshold tolerant way of the data cache. If there is a hit in the near threshold tolerant way the data can be provided in a single cycle, as before. If there is a miss, the tag check will provide which conventional way of the cache we should access on the second cycle. This reduces the energy consumed because on the second cycle only one way of the conventional cache is accessed. The flip-flops shown in Figure 7 are used to delay the enable of the conventional ways until the next cycle. There is also the added benefit of knowing if the cache will miss after the first cycle. In this case an access to the next level of memory can be initiated one cycle earlier, reducing the off-chip access time by one cycle. Of course we are trading off the energy reduction of accessing all the additional ways of the cache with the increase in energy for the tag look up. If the conventional ways of the cache are rarely accessed then the system may consume more energy with this design.

4. Methodology

A system with two different operating modes was created. The first was a 400 MHz full power mode. The frequency was determined by the ARM9 CPU model used for energy analysis, the details of which were obtained from consultation with the chip designers. The second mode, a low power mode, was chosen to operate at a 10 MHz clock frequency. This operating frequency was picked to provide a significant energy savings while still being fast enough to handle a wide range of simpler applications. A summary of the resultant design parameters for the system is in Table 1. For this study we use a system with a split instruction and data cache, but to limit the analysis space we kept the same sizes and types of caches for both the instruction and data memory hierarchies. For all comparisons the die size was held constant.

4.1. Simulation Environment

For simulation we use a modified version of the M5 simulator [5]. The simulator was modified to incorporate the dynamic and leakage energy modeling necessary to determine the energy of the overall system.

4.2. Benchmarks

For the simulation we use the MiBench benchmarks [12]. These benchmarks are small embedded system benchmarks and represent a range of applications, from automotive to encryption. A subset of benchmarks that compiled to the target architecture were run to completion for all test cases using the reduced input sets. Further analysis was done using the complete SpecInt 2000 benchmarks [13]. This analysis shows similar results as the MiBench studies and is briefly presented at the end of the results section.

4.3. Energy Models

We create accurate energy consumption models for the system using SPICE analysis coupled with iso-robustness analysis. We account for CPU, cache, bus, level converter, and off-chip access energy in the analysis. All data presented is using a commercial 130nm process, some additional analysis was done to verify that the results we show still hold for 90nm, 65nm, and 45nm. The results of that comparison are presented in Section 5.6.

4.3.1. Cache Model. We use the iso-robustness analysis method discussed in Section 2, assuming that standard SRAM cells can operate reliably down to 800mV, to determine the appropriate cell size to meet the delay of the core and iso-robustness reliability constraint for any near threshold SRAM cells. We then use SPICE modeling on the SRAM cell to determine the dynamic and leakage energy consumption of the cell at different operating points. We account for the energy in the data and tag arrays of the cache as well as any decoders associated with the cache. However we do not model the cache controller energy. This will result in our energy figures being slightly lower than a real system. We have simulated and accounted for the energy and additional cycle latency of the swap buffer. The use of body-bias techniques is assumed in our experiments to reduce the cache leakage energy, or to meet timing constraints.

4.3.2. CPU Model. For our CPU model, we base the core energy consumption numbers from a cacheless 400MHz ARM9 core. We use SPICE simulation of some representative smaller CMOS circuitry in order to determine the impact of voltage and frequency scaling on the core for both dynamic and leakage energy. The core used is extremely aggressive

Simulation Parameters

	Baseline		Traditional Filter Cache		Near Threshold Filter Cache	
	10Mhz	400MHz	10MHz	400MHz	10MHz	400MHz
Core Voltage	450mV	1.2V	450mV	1.2V	450mV	1.2V
Filter Cache Voltage	N/A		800mV	1.2V*	500mV	1.2V*
Filter Cache Size	N/A		1 kB		512 B	
L1 Cache Voltage	800mV	1.1V	800mV	1.1V	800mV	1.1V
L1 Cache Size	8 kB		7 kB		7 kB	
L1 Cache Ways	8		7		7	

Table 1. Simulated System Parameters. *Body-Biasing used to meet timing constraints.

with many structures removed to reduce energy. Amdahl's law prohibits the energy savings seen if the core energy is higher, but even in a baseline system with a core that consumes 60% of the total power our techniques still show significant improvements.

4.3.3. Off-Chip Energy. The off-chip latency was derived for a memory system composed of SRAM for data, and ROM for instructions. This is a typical setup for an embedded system, and the latency was 20 cycles at 400 MHz. The energy consumed in the package and a reasonable amount of off-chip routing is accounted for in all the measurements. However, the energy of the off-chip memory itself is not accounted for in the simulation as the size of this could vary widely depending on the application. A designer could use our figures as part of their total power/performance calculations.

5. Results

5.1. Filter Cache

The first analysis to be performed was on the simple near threshold filter cache. The first step was to determine the L1 size for the baseline system without a filter cache that provided the lowest energy solution. A sweep of L1 sizes and associativities yielded an optimal size of an 8kB, 8-way cache for the MiBench benchmarks. Then, while holding the die size constant the lowest energy system with a near threshold filter cache size was determined. The analysis was done keeping in mind that the size of the near threshold filter cache SRAM cells are larger in size than the standard ones used in the L1 cache. Across the benchmarks the optimal size was a filter cache of either 512 B or 1 kB. For our studies we chose a 512 B filter cache and a 7kB, 7-way L1 cache. For comparison we also evaluate a filter cache designed with standard SRAM cells, which do not support voltage scaling below 800mV. We term this configuration a traditional filter cache. It has a 1 kB filter cache and a 7kB, 7-way L1 cache. A larger filter cache is possible in this case because the SRAM cells do not have to be sized to operate at lower supply voltages. A summary of the baseline and filter cache systems can be found in Table 1. For the initial results we do not use the bypass method described in Section 3.1, the impact of using a bypass on the filter cache will be evaluated in Section 5.1.2.

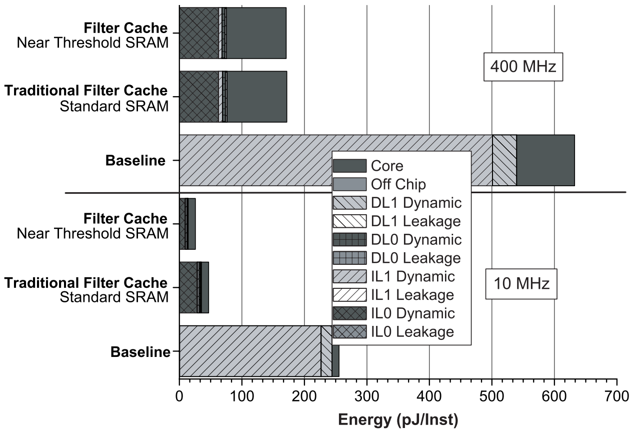


Figure 8. Filter cache energy breakdown for BitCount benchmark.

5.1.1. Without Bypass. The results of the analysis are presented in Figure 8. In the figure there are 6 bars, the top 3 bars are for the system at 400 MHz and the bottom 3 are for the system at 10 MHz. The bars present the total energy of the system divided by the number of instructions completed. We illustrate the analysis with the BitCount benchmark. The first thing to note is that by simply using voltage scaling on the baseline system we can reduce the energy consumption to complete this benchmark by 60% (3rd bar vs. 6th bar). It can also be seen that, because we are able to more aggressively voltage scale the core, the cache quickly becomes the dominant energy source at low frequencies. From the figure it can be seen that the IL1 consumes most of the energy in the memory system for BitCount. Using a filter cache dramatically reduces this IL1 dynamic energy by shielding

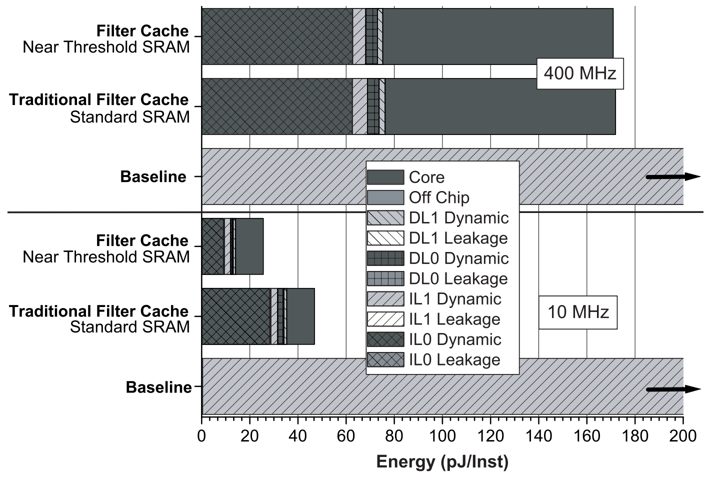


Figure 9. Filter cache energy for BitCount benchmark (Zoomed In).

the IL1 with a small filter cache. This results in a 73% (2nd and 3rd bars) reduction in energy at 400MHz and a 82% reduction in energy at 10 MHz (5th and 6th bars) over their equivalent speed baseline counterparts.

Figure 9 presents a zoomed in portion of Figure 8 to help better see the shorter bars. The addition of near threshold supply voltage scaling capabilities on the filter cache does two things. First, it reduces the energy consumption at 10 MHz a further 45% over the traditional filter cache. Second, due to the larger cell sizes, the energy at 400 MHz is increased by around 1%. This increase is mitigated by the fact that although the near threshold filter cache has larger cell sizes, there are only half as many cells as the traditional filter cache (see Table 1).

Figure 10 shows additional benchmarks from the MiBench suite and the performance of the near threshold filter cache. Figure 11 shows how the near threshold filter cache's energy consumption compares to the traditional filter cache and the baseline at 10 MHz. On average the near threshold voltage scaled filter cache shows an 84% reduction in energy over the baseline at 10 MHz, and a 36% reduction over a traditional filter cache.

5.1.2. With Bypass. The filter cache does present some drawbacks. As mentioned in Section 3.1 the existence of the filter cache can degrade performance when the miss rate in the filter cache is high. Figure 12 presents the increase in runtime that occurs over the baseline system when a filter cache is used. Note that for this analysis we used a filter cache size of 1kB and a 6kB, 6-way L1 so that we can compare equal die size RENT caches in Section 5.2. On average a 17% increase in runtime occurs with a standard deviation of 9%. The worst case was a 29% increase in runtime for the Patricia benchmark. In order to reduce the runtime increase a bypass network can be employed which

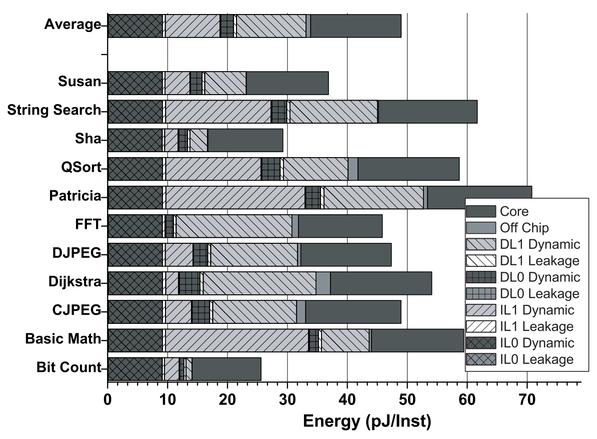


Figure 10. Near threshold filter cache energy breakdowns for all Benchmarks at 10MHz.

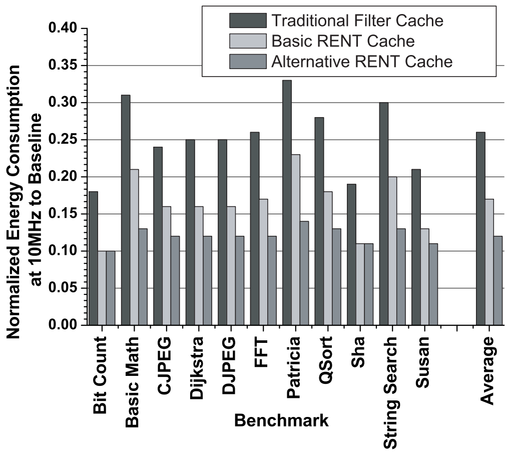


Figure 11. Cache energy comparisons for all benchmarks normalized to the baseline at 10MHz.

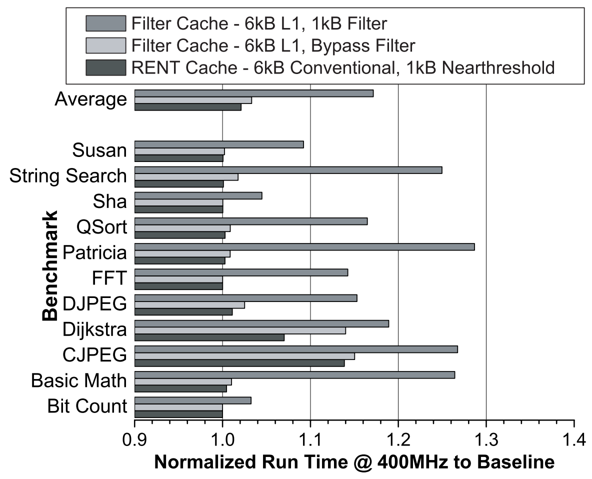


Figure 12. Normalized runtimes at full voltage to baseline machine.

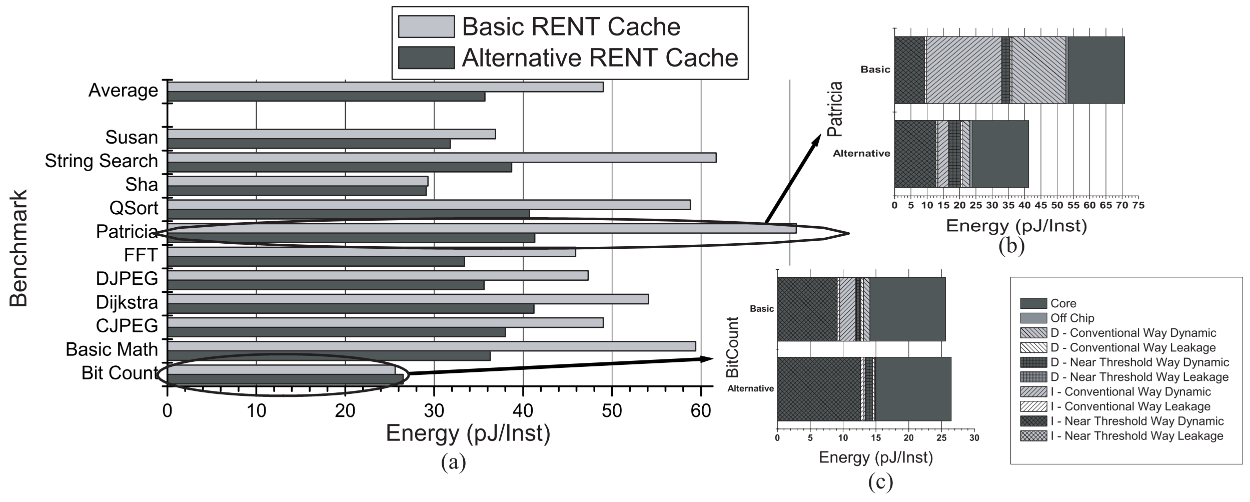


Figure 13. Comparison of energy breakdowns for RENT cache policies (a) for all benchmarks at 10MHz. With a breakdown of energy for (b) Bitcount and (c) Patricia

is enabled when the miss rate is high. The operation of the bypass filter cache is described in Section 3.1. Figure 12 also shows the resultant runtime when the bypass network is used. Notice that now the average runtime increase is only 3.3% with a 5.6% standard deviation. The benchmarks that still have a significant increase in runtime happen as a result of having a working set that is larger than the L1 cache. This leads to additional off-chip accesses which are long latency. Those benchmarks would benefit from being able to utilize the cache space that is being disabled by the bypass network.

5.2. RENT Cache

Using the RENT cache we are able to increase the cache capacity on chip when running in the conventional mode. The simulation configuration is slightly different because all ways in the cache should be the same size. The resultant cache sizes are presented in Table 2, notice that the near threshold tolerant cache way requires about the same space as two conventional cache ways and so the total cache size is decreased. This is an improvement over the bypass cache because it allows the benchmark to utilize more of the cache on chip in conventional mode-7kB. In addition the cache is also able to adapt dynamically to different phases of program behavior. The third bar in Figure 12 shows that with the use of the RENT cache we reduce the runtime overhead even further. The average increase is now just 2.1% on average with a 4.4% standard deviation. The use of the RENT cache has also kept the energy consumption at the same level in the low power mode as the bypass filter cache.

	Baseline	RENT Cache
# of Near Threshold Ways	0	1 x 1kB
# of Conventional Ways	8 x 1kB	6 x 1kB

Table 2. Simulated System Parameters for RENT Cache

5.3. Alternative RENT Cache

Using the alternative RENT cache the energy savings can be even greater. The die size is slightly larger to accommodate the near threshold tags, but was not significant enough to justify the removal of another full data cache way. In Figure 13(a) the basic and alternative version of the RENT cache are compared across the MiBench benchmarks. For almost all the cases there is a decrease in total energy. A blown-up detailed view of the Patricia benchmark is presented in Figure 13(c). It clearly shows a reduction in the energy of the conventional ways of the cache. The dynamic energy to the near threshold portion of the cache is increased due to the additional tag checks. In this case the decrease in the conventional access energy outweighed the increase in the tags and the alternative method was better. In Figure 13(b) we present a case in which the basic RENT cache outperforms the alternative version, the BitCount benchmark. In this case there was already very little conventional way dynamic energy and the increase in the filter way dynamic energy from the additional tag accesses was too large for the alternative method to be more energy efficient. Note that in all cases there is a slight reduction in runtime and therefore a small reduction in the leakage energy for all the alternative RENT cache methods. This reduction in runtime comes from being able to issue off-chip requests after the first cycle instead of the second cycle, thus reducing the total runtime by one

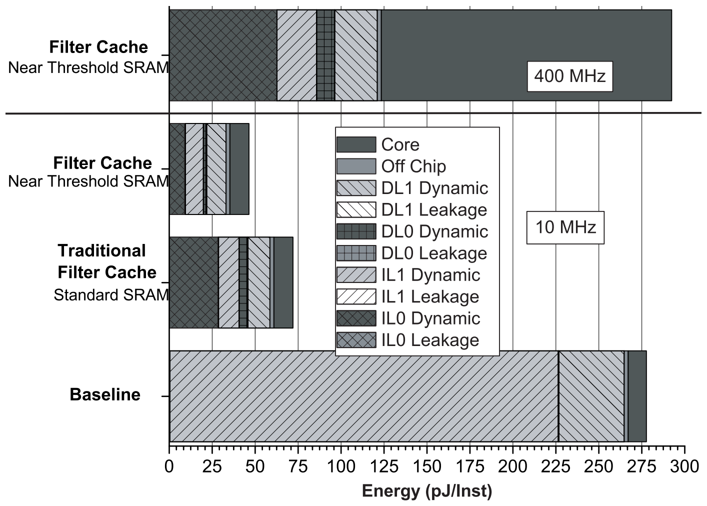


Figure 14. Energy breakdowns for the GZIP benchmark.

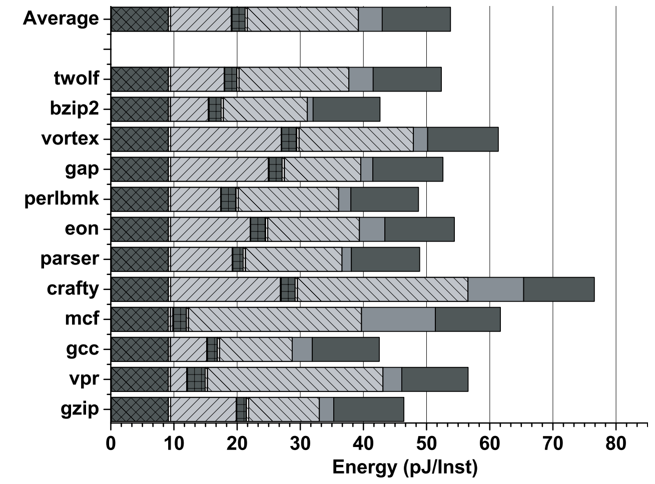


Figure 15. Energy breakdown for SpecInt 2000 benchmarks with the basic RENT cache at 10MHz.

cycle per off-chip access. It is hard to see this decrease in the graph, but the data confirms this expected result. From Figure 11, on average the alternative RENT cache policy provides an additional 27% average energy reduction over the basic RENT cache, 54% over the traditional filter cache, and an 88% over the baseline at 10MHz.

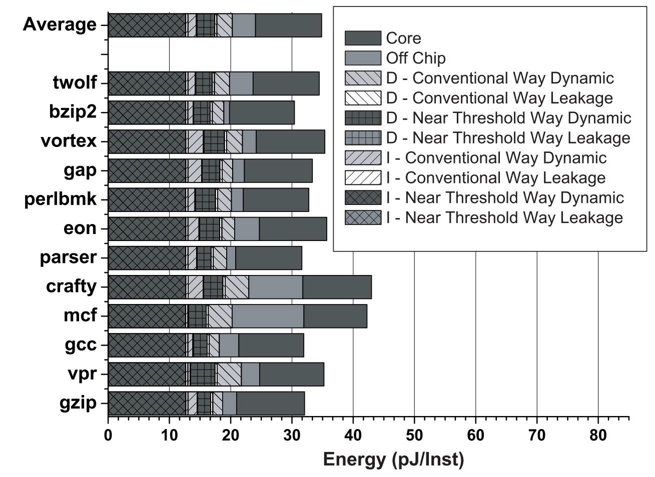


Figure 16. Energy breakdown for SpecInt 2000 benchmarks with the alternative RENT cache at 10MHz.

5.4. Spec2000 Analysis

Analysis was also done using the SPECINT benchmarks. Figure 14 shows the energy breakdown of the GZIP benchmark for the traditional and near threshold filter caches. Notice that even for these larger benchmarks we are still seeing a 34% reduction in energy using the near threshold SRAM cells. Figure 15 and Figure 16 show the energy breakdowns of the 10MHz basic and alternative RENT caches. We still show significant energy reductions using the alternative policy for the SpecInt benchmarks. Overall we see a 57% reduction on average using the alternative RENT cache over a traditional filter cache at 10MHz.

5.5. Power Savings Mode

The power savings figures are calculated by taking the energy consumption of the alternative rent cache in the low power mode, 10MHz, and comparing that to the high performance mode, 400MHz. For the MiBench benchmarks on average we show an 86% (7.3x) reduction in energy when operating in low power mode, with only a 2% increase in runtime in high performance mode. For the SpecInt benchmarks we show on average a 77% (4.4x) reduction in energy in low power mode with only an average 4.8% increase in runtime while in the high performance mode.

5.6. Technology Node Comparison

Additionally we verified that the trends still hold in smaller technology nodes. Figure 17 shows the average MIBench performance at different technology nodes for both the RENT and alternative RENT cache. The bars are normalized to the performance of the baseline in the low power mode at the same technology node (i.e. 45nm RENT cache is normalized to 45nm Baseline). From the graphs you can see that we are still seeing a 88-90% reduction in cache energy for the basic RENT cache across all nodes. The breakdown of where the energy goes shifts to leakage at smaller nodes, but the same overall trends hold.

Each technology node has a different optimal voltage and frequency for both the low power mode and the high performance mode, space limits us from presenting them.

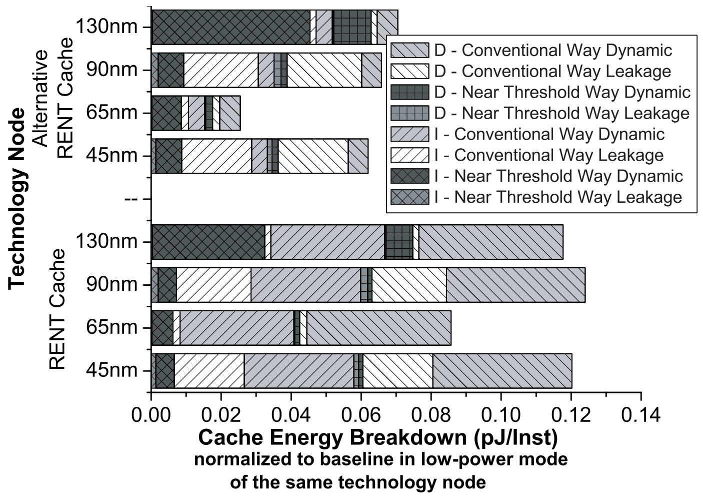


Figure 17. Normalized Cache Energy for Average MIBench Performance at Different Technology Nodes.

5.7. Additional Analysis

A number of additional experiments were run, but again space limits what we can show. These experiments included sensitivity analysis for the off-chip latency and energy consumption numbers, drowsy cache techniques [15], and 1MHz subthreshold designs.

6. Related Work

There has been a significant amount of work done in the area of energy efficient cache architectures, particularly for embedded applications. Our work differs from the previous work in that we specifically use a near threshold tolerant SRAM design to explore cache architectures that can scale into the near threshold operating region. This results in significant energy savings of 36% over a traditional filter cache. The original work on filter caches was presented by Kin et al. [17], and Tang et al. [19] expanded on that work by creating a prediction network that would allow the instruction filter cache to be bypassed.

Further work in the reduction of cache energy was done by Albonesi [1]. That work proposes a cache that reconfigures itself to be optimal in size. This is achieved by enabling and disabling ways of the cache. This work is orthogonal to the work we present, and could be used to further improve energy performance by disabling cache ways for applications with small working sets. Inoue et al. investigated the idea of way prediction [14] and Powell et al. [18] expanded on it. In their work only the predicted way is accessed on the first cycle, after which the other ways are accessed. This is similar to the RENT cache in that we are only accessing one way of the cache on the first cycle. Our work differs because we always start with the near threshold tolerant way of the cache and swap the MRU block into that way of the cache. This helps to leverage the low energy of accessing this way of the cache. Zhang [27] proposes a multi-column cache to reduce energy, this work expands on the idea of way-prediction focused mainly on the MRU way of the cache. Zhang performs swapping of the MRU block into the next predicted way, although this is not guaranteed to be the first way. Thus, preventing it from being used with the near-threshold SRAM cells. Zhu's [29] work most closely resembles our work in that it focuses on alternating access patterns to the cache based on performance to achieve energy optimality. However Zhu's work builds on that of Zhang and, again, because the predicted way is not always in the same place near-threshold techniques do not work. Lastly, Zheng [28] proposes a cache that uses way concatenation to help reconfigure the cache into the optimal energy configuration.

In addition there have been several studies investigating the trade-offs of swapping cache locations. The work done by Balasubramonian [2] and Dropsho [10] are similar to the work proposed in the RENT and alternative RENT cache design. These ideas however suffered from the fact that the swap mechanism was energy hungry and the ideas were passed over for more complicated designs. Batson [4] did an investigation of such techniques and found the energy performed from swapping outweighed the gains of hitting on a prediction. However this pertained to standard SRAM cells, if the energy savings of the prediction is increased using near-threshold SRAM, this trade-off favors the use of swapping as was shown by our results. This coupled with the fact that more recent techniques can not be done using near-threshold techniques efficiently reinstates these previous ideas in the new context of near-threshold computing.

Additional work was done looking at caches with cache ways of different latencies. Fujii [11] and Balasurbramonian[3] both look at techniques where low energy cache ways are used and operated at slower speeds. And faster, high energy cache ways are accessed first or by hot addresses streams. Their focus is on much larger caches and the reduction of leakage energy. Their work differs in that they focus on using low energy cache ways second after the access to the high energy way, or place hot frequent accesses in the high energy way and cold infrequent accesses in the low energy way. Whereas we perform the lookup in the opposite order.

There has also been a large number of studies on sub-threshold and near-threshold systems [9, 21, 24, 26]. These studies, however, focus on small performance targets or chip multiprocessing, unlike our system which targets a single general purpose core that can operate in both low power mode and high performance mode. There has also been substantial work on subthreshold and near threshold SRAM design [7, 16, 20, 23], but none of these considers potential cache architectures for embedded applications.

7. Conclusion

Embedded processors, particularly for mobile applications, are requiring ever increasing performance but still have battery lifetime as a critical design parameter. In this work we propose an embedded processor with a high performance mode to handle time sensitive and compute intensive work, and a low power mode which can complete non-critical tasks in an energy efficient manner. To achieve this we investigate near threshold tolerant memory structures coupled with voltage and frequency scaling.

We propose the RENT cache to provide both a significant reduction in energy in low power mode without penalizing runtime in high performance mode. The cache is designed with one near threshold voltage tolerant cache way to filter accesses to the rest of the cache. This cache way is accessed first, and only on a miss are the other cache ways accessed. If the miss rate in the near threshold cache way becomes to large and degrades performance the cache is dynamically reconfigured to act like a conventional cache, where all the cache ways are accessed in parallel. This changes the cache to have a uniform, single cycle hit latency. Using this technique we show a 53% reduction in energy over a traditional filter cache. This leads to a system that provides a 86% (7.3x) reduction in energy while in low power mode with only a 2% increase in runtime in high performance mode.

References

[1] D. Albonesi. Selective cache ways: On-demand cache resource allocation. In *31st Ann. Int'l Symp. on Microarchitecture*, 1999.

[2] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *33rd Ann. Int'l Symp. on Microarchitecture*, 2000.

[3] R. Balasubramonian, V. Srinivasan, S. Dwarkadas, and A. Buyuktosunoglu. Hot-and-cold: Using criticality in the design of energy-efficient caches. In *LNCS: Power - Aware Computer Systems*, 2005.

[4] B. Batson and T. Vijaykumar. Reactive-associative caches. In *10th International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*, 2001.

[5] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, Jul/Aug 2006.

[6] B. Calhoun and A. Chandrakasan. Characterizing and modeling minimum energy operation for subthreshold circuits. In *IEEE/ACM ISLPED*, 2004.

[7] B. Calhoun and A. Chandrakasan. A 256kb sub-threshold sram in 65nm cmos. In *IEEE ISSCC*, 2006.

[8] G. Chen, D. Blaauw, N. S. Kim, T. Mudge, and D. Sylvester. Yield-driven near-threshold sram design. In *Proc. 2007Int'l Conf.on Computer Aided Design*, 2007.

[9] R. Dreslinski, B. Zhai, T. Mudge, D. Blaauw, and D. Sylvester. An energy efficient parallel architecture using near threshold operation. In *Proc. 16th Ann. Int'l Conf. on Parallel Architectures and Compilation Techniques*, 2007.

[10] S. Dropsho, A. Buyuktosunoglu, R. Balasubramonian, D. Albonesi, S. Dwarkadas, G. Semeraro, G. Magklis, and M. Scott. Integrating adaptive on-chip storage structures for reduced dynamic power. In *Proc. 11th Ann. Int'l Conf. on Parallel Architectures and Compilation Techniques*, Sept. 2002.

[11] S. Fujii and T. Sato. Non-uniform set-associative caches for power-aware embedded processors. In *LNCS: Embedded and Ubiquitous Computing*, 2004.

[12] M. Guthaus, J. Ringenberg, T. Austin, T. Mudge, and R. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *4th Annual Workshop on Workload Characterization*, Dec. 2001.

[13] J. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *IEEE Computer*, 33(7):28–35, July 2000.

[14] K. Inoue, T. Ishihara, and K. Murakari. Way-predicting set-associative cache for high perfromance and low energy consumption. In *Proc. 1999 Int'l Symp. on Low-Power Electronics and Design*, 1999.

[15] N. S. Kim, K. Flautner, D. Blaauw, and T. Mudge. Single-vdd and single-vt super-drowsy techniques for low-leakage high-performance instruction caches. In *IEEE/ACM ISLPED*, 2004.

[16] T.-H. Kim, J. Liu, J. Keane, and C. H. Kim. A high-density subthreshold sram with data-independent bitline leakage and virtual-ground replica scheme. In *IEEE ISSCC*, 2007.

[17] J. Kin, M. Gupta, and B. Mangione-Smith. The filter cache: An energy efficient memory structure. In *27th Ann. Int'l Symp. on Microarchitecture*, 1997.

[18] M. Powell, A. Agrawal, T. N. Vijaykumar, B. Falsafi, and K. Roy. Reducing set-associative cache energy via way-predicition and selective direct mapping. In *34th Ann. Int'l Symp. on Microarchitecture*, 2002.

[19] W. Tang, R. Gupta, and A. Nicolau. Design of a predictive filter cache for energy savings in high performance processor architectures. In *Proceedings of the 2001 International Conference on Computer Design (ICCD)*, 2001.

[20] N. Verma and A. Chandrakasan. A 65nm 8t sub-vt sram employing sense-amplifier redundancy. In *IEEE ISSCC*, 2007.

[21] A. Wang and A. Chandrakasan. A 180mv fft processor using subthreshold circuits techniques. In *IEEE ISSCC*, 2004.

[22] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *DAC*, 2004.

[23] B. Zhai, D. Blaauw, D. Sylvester, and S. Hanson. A sub-200mv 6t sram in 0.13um cmos. In *IEEE ISSCC*, 2007.

[24] B. Zhai, R. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester. Energy efficient near threshold chip multiprocessing. In *Proc. 2007 Int'l Symp. on Low-Power Electronics and Design*, 2007.

[25] B. Zhai, S. Hanson, D. Blaauw, and D. Sylvester. Analysis and mitigation of variability in subthreshold design. In *Proc. 2005 Int'l Symp. on Low-Power Electronics and Design*, 2005.

[26] B. Zhai, L. Nazhandali, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, D. Blaauw, and T. Austin. A 2.60pj/inst subthreshold sensor processor for optimal energy efficiency. *IEEE VLSI Technology and Circuits*, 2006.

[27] C. Zhang, X. Zhang, and Y. Yan. Two fast and high-associativity cache schemes. In *IEEE Micro*, volume 17, pages 40–49, Sep/Oct 1997.

[28] C. Zheng, F. Vahid, and W. Najjar. A highly configurable cache architecture for embedded systems. In *Proc. 30th Ann. Int'l Symp. on Computer Architecture*, 2003.

[29] Z. Zhu and X. Zhang. Access-mode predictions for low-power cache design. In *IEEE Micro*, volume 22, pages 58–71, Mar/Apr 2002.