# AN ABSTRACT OF THE THESIS OF

James M. Lewis for the degree of Master of Science in Electrical and Computer Engineering presented on December 7, 2007.

Title: Power Reduction of MPEG Video Decoding for Mobile Multimedia Systems

Abstract approved:

________________________________________________

Ben Lee

The purpose of this thesis is to explore methods which can reduce the power dissipation of a mobile system while decoding MPEG video. MPEG decoding is a microprocessor intensive process that makes heavy use of both the L1 and L2 caches as well as main memory. The heavy load placed on the system during the MPEG decoding process results in large dynamic power losses caused by both the execution of instructions and the flow of data into and out of the caches and main memory.

To reduce the power dissipation of the system during MPEG decoding, multiple techniques were applied to control the flow of data and make the decoding process more efficient. The system was simulated with different L2 cache sizes to determine which sizes resulted in the best power improvements while maintaining acceptable performance levels. A fast IDCT algorithm was implemented to improve the efficiency of the decoder during the computationally heavy IDCT phases. Finally, selective caching was introduced to the system to further reduce the traffic between the caches and main memory. These techniques were simulated on the Sim-Panalyzer simulator using a similar system configuration to one found in a typical mobile media device. These methods coupled with proper L2 cache sizing produced power reductions of 50-60% over the baseline system.

# Power Reduction of MPEG Video Decoding for Mobile Multimedia Systems

by
James M. Lewis

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented December 7, 2007
Commencement June 2008

## ACKNOWLEDGEMENTS

I would like to express my sincere appreciation and gratitude to those people at Oregon State University who assisted me in this journey.  First, and most importantly, I would like to thank Dr. Ben Lee for his guidance, insight, and patience throughout this process.  I would also like to thank my committee members for taking their time to help me in this endeavor.  I also must thank all the professors and instructors throughout the years who have influenced me in countless ways.  Finally, I would like to thank my wife and family for their patience and unceasing support.

# TABLE OF CONTENTS

Page

# LIST OF FIGURES

# LIST OF TABLES

Power Reduction of MPEG Video Decoding for Mobile Multimedia Systems

## 1. Introduction

Technology has allowed today's populations to become more mobile than ever, providing information and communication anytime and anywhere. The increasing demand for mobility has lead to small, portable devices, which have in turn created a need for smaller, faster, and lower power systems. Currently, portable devices can easily handle voice communication and text messages, with some devices offering wireless internet access and video playback. Even more advanced devices offer video-based mail, phone calls, and even video-on-demand (VOD). This shift in focus requires powerful hardware and software that consume low power.

Video compression algorithms such as MPEG are very efficient in compressing high quality video to a more easily transmittable size. However, most video compression standards are not designed with power efficiency in mind. The performance of these algorithms has been thoroughly studied in many environments and applications [1, 2], but only recently has attention turned to the impact and reduction of power consumed during the execution of these techniques. Many power reduction techniques focus on reducing power in the microprocessor through techniques like Dynamic Voltage Scaling (DVS) [11, 10] or through specialized hardware such as power-aware Field Programmable Gate Arrays (FPGA) and Systems-on-a-Chip (SOC) [9, 5]. The limiting factor of these techniques is that they do not approach the problem by looking at the interactions between multiple components.

Current mobile multimedia systems contain a variety of components that must all interact to perform the video decoding process. While reducing the power consumed by each component is important, a system-wide optimization is required to improve the power efficiency of all of the components by addressing the interactions among components.

This paper explores these system level power issues and presents methods to reduce power usage of the system while maintaining an acceptable level of quality for video playback. Past research has generally focused on studying the effects of one technique on one component in the system. This paper applies a group of techniques to a set of tightly integrated components, namely the microprocessor, RAM, and memory bus, with the purpose of increasing power efficiency through a combination of decoder performance optimizations and improvements in the flow of data between components. Methods such as Dynamic Voltage Scaling (DVS) and efficient caching techniques have only been applied to the processor and cache, respectively, without analysis of the effect on the surrounding system. Instead, this research looks at the effect of a few techniques on the whole processor/memory system as well as the effects of multiple techniques working simultaneously.

Controlling system cache sizes and configurations are typically limited by the specifications supported by the processor and manufacturing processes. However, looking at the interactions between the cache and system memory for differing sizes, specifically in the realm of power dissipation, provides more opportunities to improve system power characteristics. Furthermore, the Fast-IDCT was developed at the advent of modern mobile devices and therefore has not been applied as a power saving tool in mobile multimedia devices, but instead as a performance booster to aid the video playback of slower mobile processors. Finally, techniques similar to the selective cache approach used in this study have been employed in desktop and server systems by a handful or architectures to aid performance and efficiency, but have not been applied or supported by standard mobile microprocessors.

The organization of this paper is as follows. Section 2 discusses research related to the methods covered in this thesis. Section 3 introduces the proposed methods and their implementations. Section 4 presents the simulation environment and discusses the results. Finally, Section 5 provides a conclusion and discusses future work.

# 2. Related Work

## *2.1 Microprocessor Power Reduction*

To compare the power consumption with the speed of a particular hardware architecture, understanding what comprises the power consumption of a given system is important. There are two basic forms of power usage in any architecture: static power and dynamic power. In this study of systems, the base power requirements that need to be met in order for the system to function are no concern, but instead turn our attention to the unwanted forms of power usage, namely, power losses.

The frequency that a chip operates at is directly proportional to its supply voltage and its switching or threshold voltage. This leads to the following equation to model power consumption in a system [22]:

$$P = ACV^2 f + VI_{leak} \qquad (1)$$

This equation (1) can be broken into two parts: the first term represents the dynamic power usage and the second represents the static power losses due to leakage current. *A* represents the activity factor or fraction of gates that are switching at any one time, *C* is the total capacitance load of all the gates in the system, *V* is the operating voltage, *f* is the frequency, and *Ileak* is the leakage current. Static power losses are heavily dependent on the process technology used in the manufacturing of the microprocessor. Dynamic power losses are the focus of the related work discussed here. In particular, power reduction techniques focus on reducing both the frequency and voltage of a system through scheduling techniques, and/or the activity factor through data shaping and flow control.

Dynamic Voltage Scaling (DVS) is one of the most widely researched and implemented dynamic power management techniques in today's low-power systems. A voltage scheduler is employed to dynamically alter the operating voltage of a system, which in turn alters the operating frequency of the system. Typical system loads consist of bursts of activity separated by periods of idle time. The goal of DVS is to stretch the processing times to cover the idle times by running at lower frequency and voltage. This allows for a more efficient use of energy when a system is not performing under heavy load.

DVS techniques can be split into two basic categories: interval-based algorithms and deadline-based algorithms. Interval-based DVS methods, proposed in [12], [13], and [16], divides running time into equal units and attempts to assign a task to the minimum number of time blocks. To do this, the algorithms analyze the system usage of past blocks and attempt to set the appropriate voltage level for future blocks. The benefits of these methods are that they can be added to most systems that support multiple voltage levels and are less complex than their deadline-based counterparts. The downside is miss-prediction of future processing needs of a system.

Deadline-based algorithms attempt to correct the quality of service (QoS) issues present in interval-based systems by scheduling the voltage/frequency levels of a system while maintaining a guaranteed deadline for task completion. Designs proposed in [14] and [15] work by determining and managing an optimal voltage/frequency and attempting to stretch tasks to fill run-time gaps, or NOPs, thereby allowing the system to run slower by eliminating idle times. These designs tend to produce more efficient results, but suffer from higher complexity and difficult implementations. Unfortunately, these techniques only focus on reducing power dissipation of the microprocessor and cannot be used to optimize the entire system.

Another interesting power reduction technique is the implementation of adaptive storage structures, such as those suggested by S. Dropsho, *et al.* [43]. The first structures they develop are selective cache ways. This structure breaks a cache down into multiple, equally sized “ways”. These ways are separated into two groups, A and B. On a cache access, group A is accessed first and each way in group A is searched. If the data is not found in group A, then the ways in group B are searched. If the data is found in group B, then it is read as well as moved into group A, where it replaces the least recently used block. The goal is to have the most often accessed data in group A for as much time as possible. This allows the cache to adapt to changing conditions on the fly.

Their research also suggests the use of adaptive buffers through the microprocessor. These buffers use limited histograms to determine how they should be resized. This resizing is accomplished by partitioning each memory structure through bit-line segmentation, which isolates regions of the memory electrically and reduces dynamic power losses. This results in power saving of up to 70% on individual memory structures and 30% on the overall processor. However, this comes at the cost of an estimated 2.1% performance degradation due to the added overhead of these structures. Unfortunately, the configuration tested does not closely resemble a mobile system nor do the researchers implement a more complete system, which would include main memory, a network interface, and a hard drive.

## *2.2 Power Reduction in Video Decoding*

DVS has shown promise in enhancing the power efficiency of mobile video devices. Son *et al.* proposed two DVS algorithms specifically designed for MPEG4 video decoding [17]. DVS-DM is an interval-based algorithm that schedules voltage based on previous workload. DVS-PD is an algorithm that schedules voltage/frequency by both looking at previous workload and predicting decoding times for future video frames. They found that DVS-DM, which only predicts using past results, had difficulty with streams possessing high fluctuations in processing needs. DVS-PD proved to be very effective, reducing the power losses by 13%-56% depending on the stream. However, this study was done in a processor simulator that does not take into account external memory devices and their connecting busses.

Lee *et al.* presented a DVS algorithm and compared it to other algorithms, which were all analyzed in terms of decoding time prediction as well as a study on optimum processor voltage/frequency granularity [10]. They present a new algorithm, Frame-data Computation Aware Dynamic Approach (FDCA). This algorithm uses data contained in each frame to aid in efficient decoding. The authors found that their FDCA algorithm proved to be the best when dealing with both stored and real-time video. Their techniques provided a 68% power savings over a standard decoder.

Mesarina and Turner presented a scheduling algorithm for MPEG playback, which is a more algorithm-centric solution [18]. The decoding process is divided into tasks, and their scheduling algorithm sets out to assign a voltage and frequency to each task so that it requires the minimum amount of energy. This algorithm was then tested on a Pentium II and Pentium III processor with different voltage/frequency levels. The results showed that savings of 19% were achievable for high quality playback and up to 50% for slightly reduced quality. These savings are difficult to compare to other DVS algorithms because they were realized on a desktop system (Pentium II and Pentium III) and based on a server/client scheduling algorithm. Also, this algorithm is not ideal for an embedded real-time video system because of the complex task scheduling algorithm and preprocessing needed.

Choi *et al.* presented a method that breaks a frame into a frame-dependent (FD) and frame-independent (FI) portion [19]. The FD part contains the parsing, IDCT, and reconstruction steps. The FI part contains the dithering step. Voltage/frequency scaling is first performed on the FD portion of the frame. If a misprediction is made during the FD portion of the algorithm, the FI portion must take appropriate steps to insure the process will meet the deadline. Since it was found that the energy consumption of the FI portion was nearly constant, it is possible to shrink or compress the time allowed for the FI portion to compensate for under or over predictions. Using this method, savings of approximately 53% were estimated. However, this method opens up the possibility of errors propagating to other frames.

## *2.3 Power Reduction in Mobile Devices*

Mohapatra *et al.* explored various techniques for reducing power consumed by mobile handheld devices, which range from architectural changes to an adaptive middleware framework [6]. They begin by identifying which system elements provide the means to control power usage—network interface and CPU/memory. The display is the third major component in a handheld system, but this component was not considered in their study. They then discuss some techniques for minimizing power dissipated in the CPU/memory portion of the device. One method is cache reconfiguration that alters the structure of the cache to improve the spatial locality of video data, which in turn reduces the high number of cache-to-memory transfers. They found that this can save 10-20% on the cache power usage. Another method is to take advantage of a processor's DVS capabilities, which was discussed earlier in this paper.

One of the middleware techniques presented is Energy-Aware Admission Control and Stream Transformations. [6] This method uses feedback from a mobile device to monitor the "utility factor" (performance vs. power usage) of the mobile system. A decision is then made as to the highest quality video that should be streamed while meeting the device's energy budget.

Another method explored is Network Traffic Regulation, where the system dynamically adapts to changing network and device conditions. This is accomplished by considering access point capabilities and the underlying network protocol being used. Then data is buffered by a proxy and transmitted in bursts of video data which contain control information for the devices [6]. Doing this with an optimized stream allows the network interface card (NIC) to be put into sleep mode to help save more power than if the device is merely in idle mode.

They showed that using both the low-level and middleware techniques through the introduction of "knobs"—small communication interfaces between the architecture and middleware—allow for significant device power savings. These techniques produced energy gains as high as 57.5% in the CPU and memory portions of the system.

## *2.4 Cache/Memory Behavior of MPEG-2*

Soderquist and Leeser explore the basic cache and memory behaviors of the MPEG-2 decoding process [2]. They begin by analyzing the different types of data used in the decoding process, which consist of the following [2]:

- Input: The compressed MPEG-2 data, which is read from a fixed-size buffer and refreshed through system calls.
- Output: The uncompressed video data, which is stored in a video image buffer. This data is write-only and is transferred to the video buffer through system calls.
- Tabular: The static read-only data used during the decoding process. This data includes all of the various look-up tables utilized by the algorithms.
- Reference: This type contains the current frame and previously decoded frames used to decode the current frame. This can include up to two for decoding B-frames.
- Block: The data that contains the DCT coefficients and pixel values for an individual macroblock.
- State: This data contains values needed by the decoding process but not part of the image data

One interesting point they note is that due to its large size, the reference data used to reconstruct the B- and P- frames tend to expel other data from the cache unless the system possesses a very large cache. It is also interesting to note the Block data—DCT and pixel values for a macroblock—are referenced the most. This is to be expected since this data is at the heart of the calculations required to uncompress and reconstruct the image.

They examined how three characteristics of cache, namely cache size, associativity, and line size affect the amount of traffic between the cache and the main memory. Their studies showed that most cache sizes between 32 KB and 512 KB with associativity of two or higher performed equally well. However, the real improvements were shown to occur when the cache size meets or exceeds 1MB. In terms of associativity, they found that 2-way set-associative can reduce cache-memory traffic by 50% and increasing to 4-way set-associative reduces traffic by 60%. Moreover, associativity greater than four do not yield increased performance on a level that would make them a viable choice.

Although Soderquist and Leeser do not discuss the effects of these enhancements on power dissipation, it is clear that each would have some effect on the amount of power lost during the decoding process. Reducing the cache-memory traffic of the system would clearly reduce the number of memory accesses that cause switching and internal power dissipation. It is important to note that increasing the cache size reduces the traffic and the miss rate; however, the larger cache will dissipate more power through larger switching, internal, and leakage capacitances.

This paper takes their research a step further by studying the relationship between the caches and main memory in terms of power dissipation. Furthermore, a main memory power simulation is added to provide more detailed feedback as to how cache-to-memory traffic affects the power losses in both the processor and the RAM. Finally, unlike findings presented in [2], which were simulated on a SuperSPARC processor, this study is done on a platform which includes a mobile processor as well as the attached main memory and memory bus.

## 2.5 Fast-IDCT

One of the most time consuming operations of MPEG decoding is the Inverse Discrete Cosine Transform (IDCT). However, there has been a large amount of mathematical research focused on improving the computational performance of the IDCT, which has resulted in numerous “fast” IDCT algorithms. The Fast-IDCT algorithm used for this study is the Chen-Wang IDCT algorithm [3, 4], which strives to improve the performance of the IDCT by eliminating most of the high latency instructions such as multiplication and division. The Chen-Wang algorithm is one of the best and most popular IDCT algorithms for increasing IDCT performance.

The standard DCT and IDCT follow the following form:

$$DCT(i,j) = \frac{1}{4}C(i)C(j)\sum_{x=0}^{7}\sum_{y=0}^{7} pixel(x,y)\cos\frac{(2x+1)i\pi}{16}\cos\frac{(2y+1)j\pi}{16} \qquad (2)$$

$$pixel(x,y) = \frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} C(i)C(j)DCT(i,j)\cos\frac{(2x+1)i\pi}{16}\cos\frac{(2y+1)j\pi}{16} \qquad (3)$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} \text{ if } x = 0 \\ 1 \text{ if } x > 0 \end{cases} \qquad (4)$$

In these equations, *pixel*(*x*, *y*) is decoded by multiplying the encoded *DCT*(*i*, *j*) with two cosine values that contain divisions and two constant factors *C*(*i*) and *C*(*j*).

On the other hand, the Chen-Wang algorithm breaks this calculation into the following four stages for each row:

```
/* first stage */
x8 = W7*(x4+x5);
x4 = x8 + (W1-W7)*x4;
x5 = x8 - (W1+W7)*x5;
x8 = W3*(x6+x7);
x6 = x8 - (W3-W5)*x6;
x7 = x8 - (W3+W5)*x7;

/* second stage */
x8 = x0 + x1;
x0 -= x1;
x1 = W6*(x3+x2);
x2 = x1 - (W2+W6)*x2;
x3 = x1 + (W2-W6)*x3;
x1 = x4 + x6;
x4 -= x6;
x6 = x5 + x7;
x5 -= x7;
```

```
/* third stage */
x7 = x8 + x3;
x8 -= x3;
x3 = x0 + x2;
x0 -= x2;
x2 = (181*(x4+x5)+128)>>8;
x4 = (181*(x4-x5)+128)>>8;

/* fourth stage */
blk[0] = (x7+x1)>>8;
blk[1] = (x3+x2)>>8;
blk[2] = (x0+x4)>>8;
blk[3] = (x8+x6)>>8;
blk[4] = (x8-x6)>>8;
blk[5] = (x0-x4)>>8;
blk[6] = (x3-x2)>>8;
blk[7] = (x7-x1)>>8;
```

This is followed by a similar sequence for each column. Each one of these functions is performed eight times per 8×8 coefficient block. The Chen-Wang algorithm utilizes bit-wise shifting and the use of constants (W1, W2, …), which have been empirically determined, to reduce the latency of the IDCT calculations and increase performance.

This paper presents research which builds upon the ideas presented in this Fast-IDCT algorithm by looking for ways to further reduce the number of high-latency instructions found in the MPEG-2 decoder. It will also be shown that cycle reduction resulting from the use of the Fast-IDCT improves the power dissipation of the system. Due to the fact that improved IDCT algorithms were developed to increase decoding speed and reduce the number of cycles required, research and analysis pertaining to the effects of different IDCT techniques on system power is lacking.

## *2.6 Selective Caching*

Selective Caching techniques allow data to be transferred directly from registers in the processor to main memory, thereby bypassing the cache [26]. This allows data that is no longer needed to be removed from the cache hierarchy and thus reduce the amount of traffic between the processor and the cache both for cache writes and cache replacements.

For conventional caches, every reference is forced through the cache and every miss results in a new block of data being brought into the cache. This new block can be a piece of rarely used data and it can replace a piece of heavily used data. This can result in excess caches misses, increased traffic on the system memory bus, and lower performance.  Selective caching improves performance by eliminating many of these unnecessary block replacements and therefore preventing rarely used data from polluting the caches.  By prohibiting scarcely used information from traversing through the cache hierarchy caches are able to maintain better temporal and spatial locality.  Furthermore, cache miss rates are reduced which limits unnecessary traffic on the system memory bus and reduces the number of long multi-cycle memory accesses to the L2 cache and main memory [26].

Simulation studies done in [23] showed that eliminating the video output data from the cache resulted in a 25-50% decrease in the cache-memory traffic.  Miss rates were also improved by an average of 60%.

Selective caching techniques include the UltraSPARC architecture, which has block load and store instructions that are able to move data blocks directly between the registers and main-memory [20], and the PA-RISC architecture whose load and store instructions can contain cache-control hints that can instruct the processor to take the data meant to be stored in the cache and instead place it in main memory [21].  Both of these architectures allow for more efficient use of the bandwidth between the processor, the cache, and main memory.  Another technique described in [26] excludes known rarely used code and data from the caches, resulting in a 12-41% speedup of execution.  Unfortunately, this method requires that the processor has some knowledge about a large portion of future instructions and the context in which they are performed.  While effective, this technique is not feasible for most of today's applications because it would either require a large amount of pre-processing in order to trace through the instruction order before execution, or it would limit the processor to only be able to perform these actions on a small number of specific situations which the processor could be programmed to recognize.

However, none of these processors are designed for mobile devices, nor have the implications of Selective Caching on the power dissipation of a system been explored. Therefore, this paper looks at a mobile system architecture based on ARM processor and implements a selective caching technique to study its effect on the power consumption of key system components as well as the system as a whole.

## 2.7 *Power Dissipation of a Software Decoder*

Before any optimizations were performed, the standard MPEG decoder and system were run for the three video clips. Figure 1 shows a chart of the power dissipation of the five major system components, namely, the Processor, L1 cache, L2 cache, RAM, and memory bus. This data is averaged for the three movie clips using an L2 cache size of 128KB. Although the processor and cache can be analyzed as one component (CPU) they are separated here to show the contributions of the L1 cache.

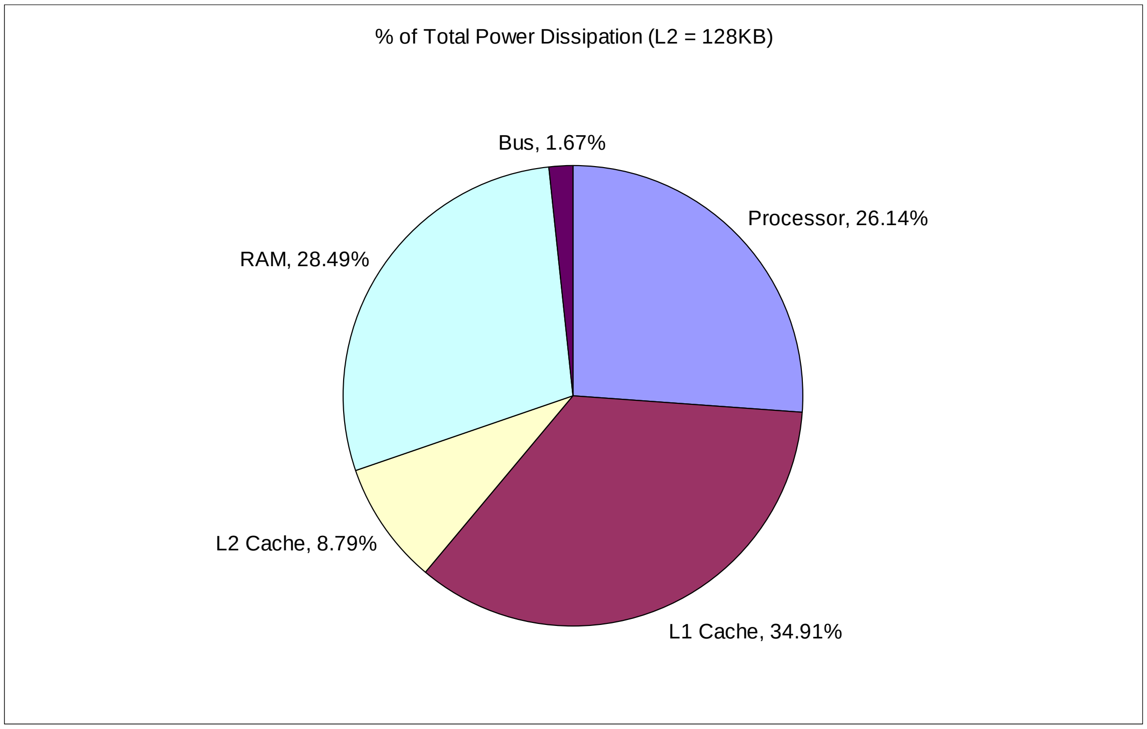


**Figure 1: Power dissipation of system components**

A majority of the power dissipation in the CPU is found in the L1 caches. This is due to the large number of accesses on the L1 caches compared to the L2 cache and RAM. The L2 cache is accessed more than the RAM but dissipates less power due to its smaller size. The large size of main memory causes its leakage losses to be much greater than those of the L2 cache which offset its reduced number of accesses. Finally, the memory bus contributes a small amount of power losses which are tempered by the fact that a large majority of memory accesses are to the L1 caches.

# 3. Proposed Methods

This section discusses a set of proposed techniques that reduces the power dissipation of a mobile system. These methods consist of *variable (configurable) L2 cache size and configuration*, *Fast-IDCT*, and *Selective Caching.* The goal is also to use as much of the available hardware without adding large amounts of extra devices or introducing extra latency. This is important because mobile systems must remain small, contain hardware flexible enough to perform a variety of functions, and MPEG decoding is a time-sensitive process.

## *3.1. Effect of L2 Cache Size*

The critical path through the system during video decoding is the processor and memory sub-systems, which include the L1 and L2 caches and the main memory. L2 cache is becoming more common-place in today's mobile processors with Intel, AMD, Freescale, and a host of processors based on ARM architectures offering 128 KB – 512 KB of L2 cache for media processing applications. These components are some of the largest contributors to the power dissipation of the system due to their constant use and large size compared to other components on the die or throughout the system.

The characteristic of the cache that is easily configurable is the cache size. Numerous studies have shown that increasing the cache size will reduce the number of cache misses, therefore reducing the necessary number of main memory transfers, which in turn reduces the dynamic power losses introduced by the main memory. This has been shown to be an effective method of reducing cache-memory transfers. [36, 37] However, when this technique is examined in the power domain its benefits are more tempered.

Unfortunately, increasing the size of the cache increases its power dissipation, increasing the total power dissipation of the system. However, because main memory is typically much larger than the cache, the increased power dissipation of the cache can possibly be offset by a reduction in the dynamic power dissipation of the main memory. In fact, if the reduction in memory accesses is large enough, it should be possible to produce a net reduction in system power dissipation if the power reduced in the memory is larger than the power lost in the larger cache.

To test the trade-offs between cache sizes and their effect on system power, the memory size simulated will remain at a constant 32 MB of SDRAM and the cache size will be varied from 32 KB to 1 MB of unified L2 cache. The goal is to determine if there is a "sweet-spot" in the system where the total power reduction benefits between cache size and system power are maximized.

## *3.2. Fast-IDCT Power vs. Performance*

As was discussed in Section 2, the Fast-IDCT presented in [3, 4] increases the speed and efficiency of the IDCT by breaking the calculations into smaller, simpler, and more efficient portions. Instead of performing many multiplications for each pixel, it breaks the calculation down into four stages, each containing a portion of the total calculation. This allows it to implement the IDCT using more additions and bit-wise shifts instead of multiplication or division in order to increase performance.

This technique was chosen because it is very effective at reducing the number of cycles required to perform the IDCT. This in turn results in a reduction in the total amount of time required to decode each frame of video, which can allow the processor to sit idle for more time, where idling requires far fewer switching transistors than during execution. Since a large portion of the power dissipation of a system is expended through switching or dynamic losses, it follows that increasing the idle cycles will reduce the switching losses.

Another benefit of the Fast-IDCT is a reduction of memory traffic among the microprocessor, the cache, and RAM. The increased number of idle cycles allows the busses between devices to remain constant for a greater percentage of the processing time, which helps reduce switching losses on the busses. Switching losses on the memory devices is also reduced because of the reduced number of accesses during the idle time.

Finally, the structure of the Fast-IDCT lends itself well to increasing the power efficiency by reducing number of cache misses generated during the IDCT portion of execution. The Fast-IDCT uses very few distinct instructions (add, multiply, shift, and subtract) so the number of I-cache misses should be greatly reduced. The data used during the IDCT should be able to be stored within a few blocks of memory so that the number of D-cache misses will also be reduced.

Overall, the expected benefit from using the Fast-IDCT algorithm is a large power savings over the standard IDCT algorithm. By both increasing idle time through the reduction the decoding time and by reducing the number of cache misses, the Fast-IDCT can reduce the dynamic power dissipation of the microprocessor, the memory busses between the processor, cache, and RAM, and the RAM itself.

While a performance gain and a reduction in the power dissipated by the system are expected, some applications choose to implement standard IDCT algorithms. Fast-IDCT algorithms have shown great performance gains but also introduce degraded image quality. [24] Liang and Tran show that Fast-IDCT algorithms reproduce images which are very close to the same quality as the original when the original image has a lower Quality Factor. As the Quality Factor is increased the quality losses introduced by the Fast-IDCT algorithms also increase. This is due to the fact that the standard IDCT normalizes intermediate results to produce more accurate results. Most Fast-IDCT algorithms bypass this normalization step to increase performance, but they produce less accurate images. A lossless IDCT algorithm could be used, but it will not produce the same performance and power gains. It is for this reason that many commercial codecs support Fast-IDCT algorithms but do not use them as the default decoding method. It is also the case that as wireless bandwidth increases and mobile device displays improve, consumers are demanding higher quality video.

## *3.3. Selective Caching to Reduce Cache Write Backs*

Data that has no future use but remains in the cache steals important space from future data and causes unnecessary cache misses, data write backs, and block replacement. A prime example of this are fully decoded macroblocks, often referred to as output data, which are write-only, meaning they are written by the processor, but never read back for any purpose. This differs from reference data, which is kept in the cache for use in decoding other frames.

In the standard MPEG decoder, output data is uncompressed into a YUV or YCrCb format for use by the video controller or display driver. During the decoding process, each completed macroblock is stored in the cache until the entire frame is completed. Once the entire frame is decoded, a system call is issued to write the frame into the frame buffer.

Examining the assembly code generated by this function shows that the decoded block must be copied from the L1 data cache (assuming the data was not replaced by another block) to a 32-bit register where the system call is then made to write the data to the frame buffer. This results in a lot of overhead on the cache from initially writing back output data and then reading output data which, once decoded, is no longer needed by the decoder.

There are a few techniques that aim to reduce the number of times data must be written back to the caches after the blocks have been decoded. The goal is to create a method for the processor to move data directly from its registers to main memory, completely bypassing the caches. Unfortunately, this is a technology that must be supported in hardware through the use of specific instructions that perform the register-to-memory transfers. Architectures which support selective caching include the UltraSPARC architecture [20] and the PA-RISC architecture [21] which were discussed in Section 2.6. However, a majority of current processors do not support such instructions, but it is possible to simulate the effects of these instructions through the strategic use of system calls from the codec to the simulator.

Unlike the standard MPEG-2 decoder, the new algorithm will make a special system call, which will write the newly decoded block directly from the register to main memory, thus bypassing the unneeded cache reads and writes for the output data. This system call will attempt to mirror an instruction that would perform the same task if it were implemented in the ARM technology.

To accomplish this, a new instruction is implemented which allows the microprocessor to transfer data directly from its registers to a location in main memory which is set aside to act as a frame buffer. This was simulated by using special system calls which allowed the simulation environment to perform the function without the need to understand the instruction at the machine level. This instruction moves the data in a 32-bit register to a location in main memory pointed to by an address.

This is simulated by making a system call immediately before the decoder writes it out as “output” data. This ensures that data that will not be used again will bypass the cache. This system call determines how many register-to-memory copies would be required and applies the appropriate amount of traffic to the memory bus and updates the number of RAM accesses. While this is not a perfect implementation of the technique, the goal is to observe any benefits that can be reaped from this technique in the areas of power and performance.

## 4. Simulation and Results

### *4.1. Simulation Environment*

The simulation environment used to study a mobile multimedia system consisted of a microprocessor simulator, memory bus and RAM simulator, and a selection of MPEG-2 encoded video clips chosen to represent a broad range of possible media inputs.

Sim-Panalyzer is based on the SimpleScalar architecture and simulates an Intel StrongARM microprocessor [7] based on the 32-bit ARM7TDMI processor core architecture [8]. This processor runs at 233 MHz at a supply voltage of 2.0 V, and contains a 16 Kbyte L1 instruction cache and a 16 Kbyte L1 data cache, as well as 128 KB of unified L2 cache. External I/O uses a supply voltage of 3.3 V. Sim-Panalyzer also provides a method to simulate the power usage of the memory bus. This is accomplished by dividing the I/O into four subcomponents: the buffer chain, I/O pad, microstrip, and external load. The I/O pad information was extracted from the TSMC 0.18um Artisan Cell Libraries. The microstrip capacitance was simulated for a typical interconnect material using an impedance calculator. The wire length and external load are fully configurable through the simulator. When an external memory access occurs during execution, the access information is placed in a queue which calculates I/O power estimation. Sim-Panalyzer then evaluates the I/O power estimation for each cycle.

A black-box methodology was used to simulate the power requirements of the main memory. During program execution, a memory trace is generated as well as vital statistics about memory usage, such as percentage of clock cycles spent on main memory reads and writes. After the program execution, the trace and statistics are used to calculate the average power usage of the main memory. These calculations are based on the SDRAM Power Calculator created by Micron for their family of SDRAMs [25]. While this method does not provide results that are as accurate as a trace-driven simulation—which often requires long simulation times—it is more accurate than a DIMM-level estimate, which is typically provided by the manufacturer. This method provides a nice balance of accuracy and simulation performance.

The main memory implemented for this study is a 32 MB SDRAM module operating at a frequency of 133 MHz and a voltage level of 3.3 V. This is the same configuration used to study integrated power management in [6].

To simulate a processor that supports selective caching a system call was added to the simulator which implements a register-to-main memory instruction, which is similar to instructions included in the architectures of some processors. Support for this was added to the MPEG decoder and an adequate cycle penalty was added to the simulation for this new instruction.

Three MPEG clips were used for simulation purposes. These clips are the same clips used in DVS research [10] except that they have been re-encoded using an MPEG-2 encoder, and are representative of three different types of video—low-motion, high-motion, and animation. The low-motion clip is a public service message about children and possesses frames whose decode times only slightly vary from each other. The high-motion clip is from the action movie *Under Siege*, and has much more varying frame decode times due to its more complex motion-estimation requirements. However, it also contains large sections of unchanging black pixels. Finally, an animated clip from *Red's Nightmare* has periods of low-motion interspersed with high-motion scenes that take up the entire frame. These clips represent a broad range of possible video structures and provide results representing real-world performance.

## *4.2. Simulation Results*

Figure 2 shows the relative improvement of applying the collective power reduction techniques to the system as compared to the baseline system as function L2 cache size. It shows that the total power required by the system is 50% or less of the baseline system and decoder for all three clips and for all L2 cache sizes. The *Reds* clip, which contains the most full frame motion estimation, showed the most improvement, while *Under* showed the least improvement. This is due to the *Under* possessing large portions of unchanging black pixels surrounding the high-motion portion of the frame. These sections require little work to encode or decode so there is less improvement from the techniques used in this study. The low-motion clip produced the most constant improvement only requiring 42% of the baseline power for all L2 cache sizes.

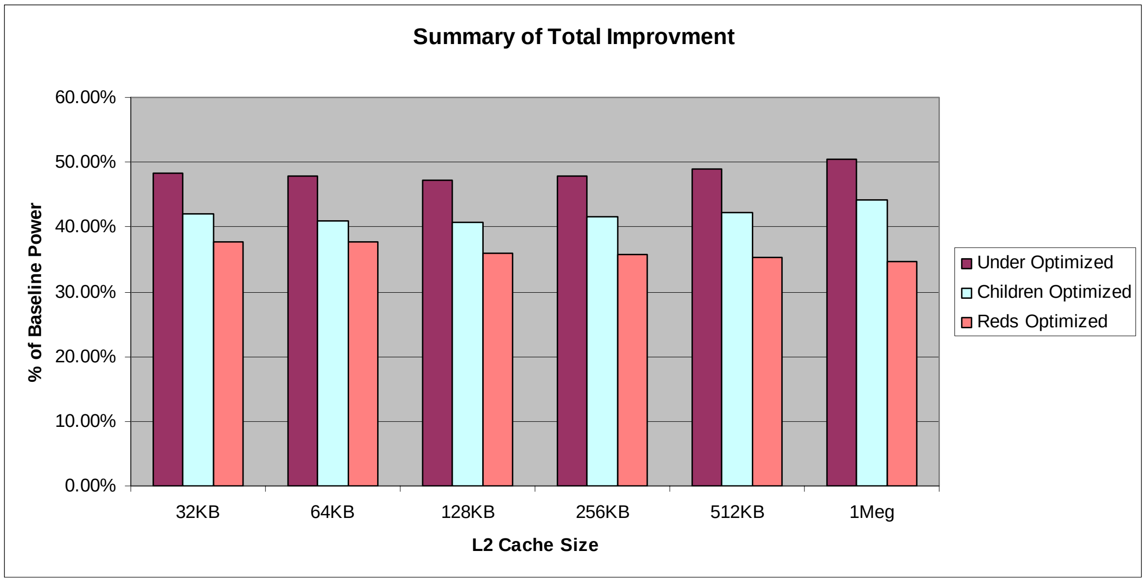


**Figure 2: Total power reduction per L2 cache size**

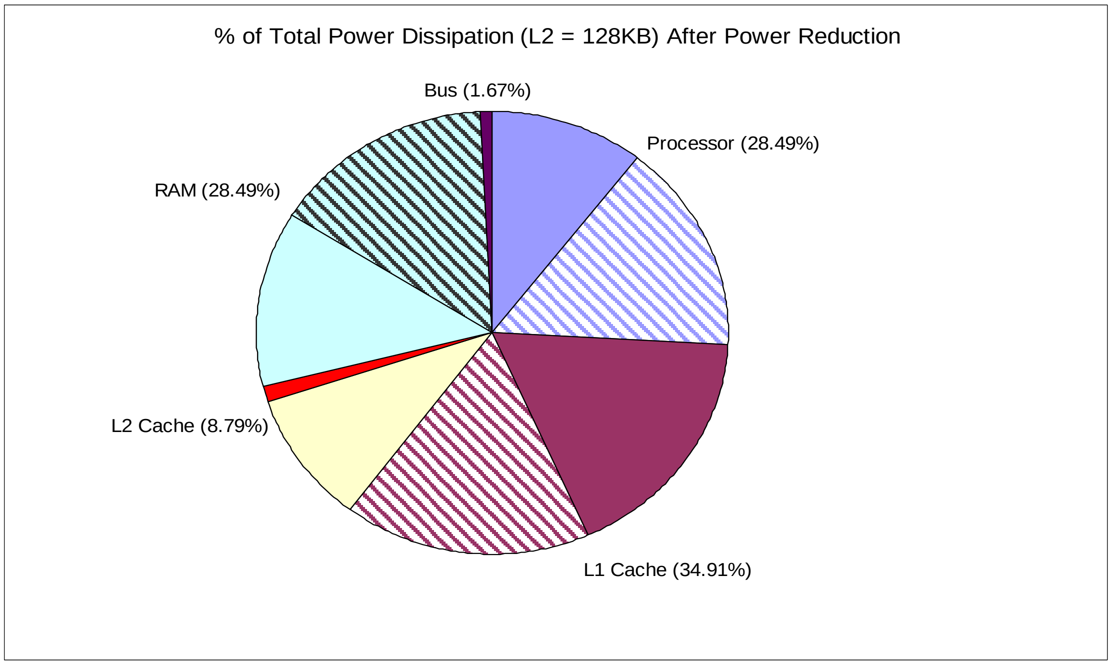


**Figure 3: Power reduction of individual system components**

Figure 3 shows the power reduction of individual components for 128KB of L2 cache. This figure mirrors Figure 1 except that the power reduction is indicated by the shaded out portion. The original percent contributions are listed next to the component names. Both the processor and L1 cache were reduced by approximately 50% and the RAM was reduced by almost 60%. The power dissipated by the L2 cache actually increased slightly which is indicated by the red area of the chart.

Figure 4 summarizes the total power savings based on frame type (I-Frame, P-Frame, or B-Frame). To obtain this data, the total power required to decode each frame of each type was averaged together. Although some frames can require more computation to decode due to larger amounts of motion estimation or less compression due to the picture complexity (similar colored pixels, etc.), these computational variations do not vary greatly enough to severely skew the results of these averages. Also, because only like-frames are being compared, the variation is further minimized.

For the most part, the power savings remain relatively constant across the different frame types. When frames are decoded using the optimized techniques, all three types run at between 30-35% of the baseline power. However, it is interesting to note that the P-frames tended to improve the most, followed by the I-frames, and then the B-frames. This is due to the fact that the P-frames hold less information than the I-frames, but also

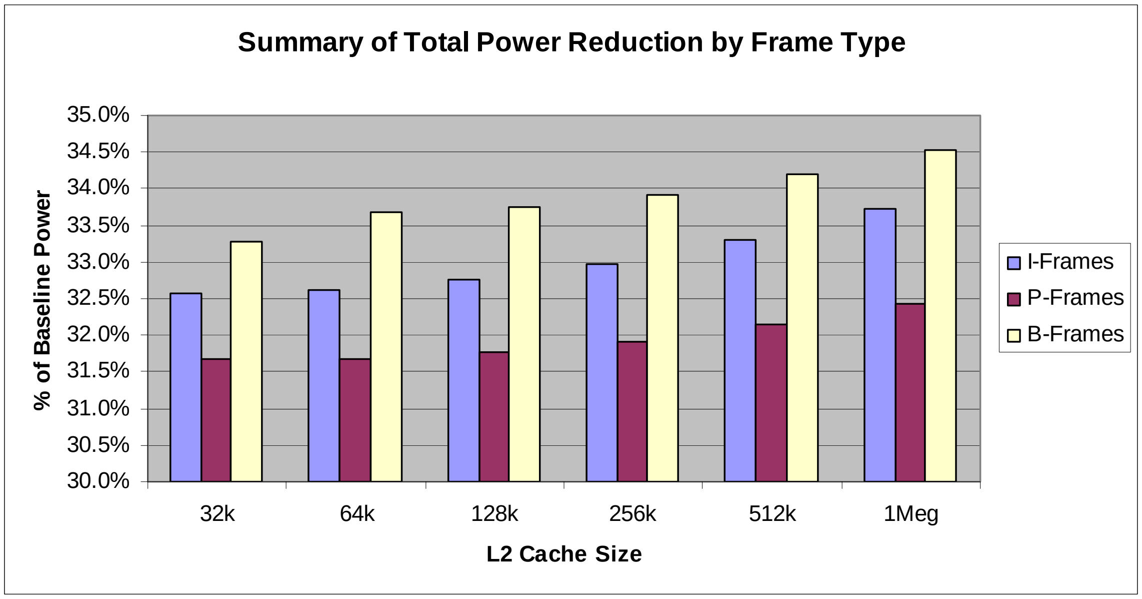
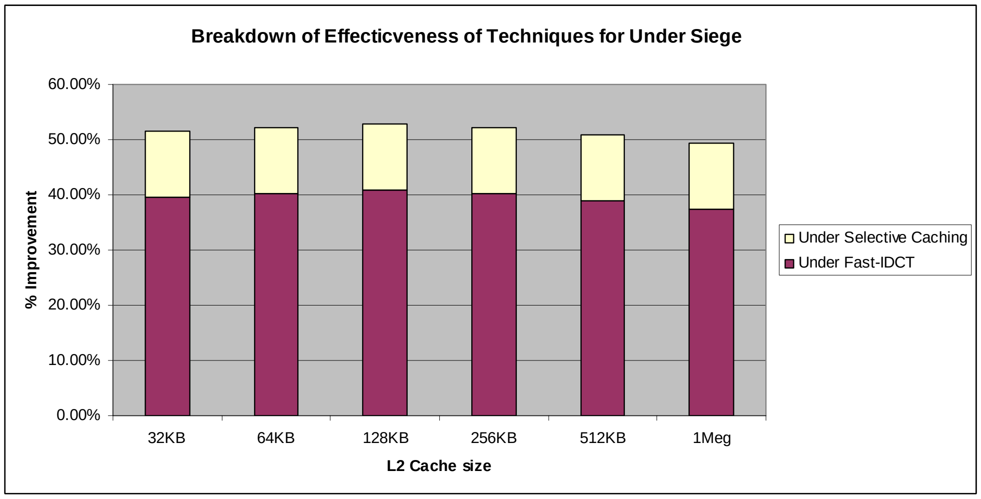


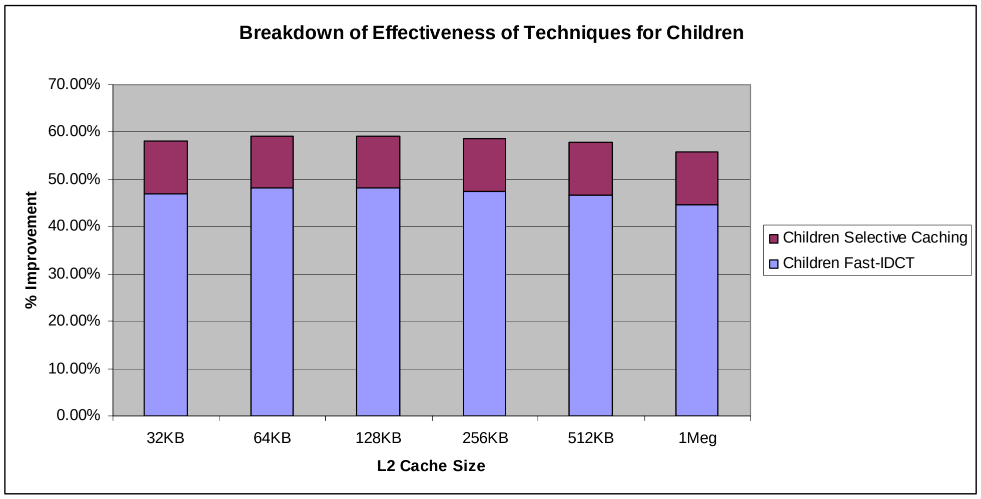
**Figure 4: Power reduction by frame type**

require less motion estimation than the B-frames since they utilize unidirectional prediction. Since only the IDCT was optimized, which has been shown to be the major contributor to the power dissipation improvements, the frames whose cycles are used more for VLD and Motion Estimation would benefit less from the Fast-IDCT implementation.

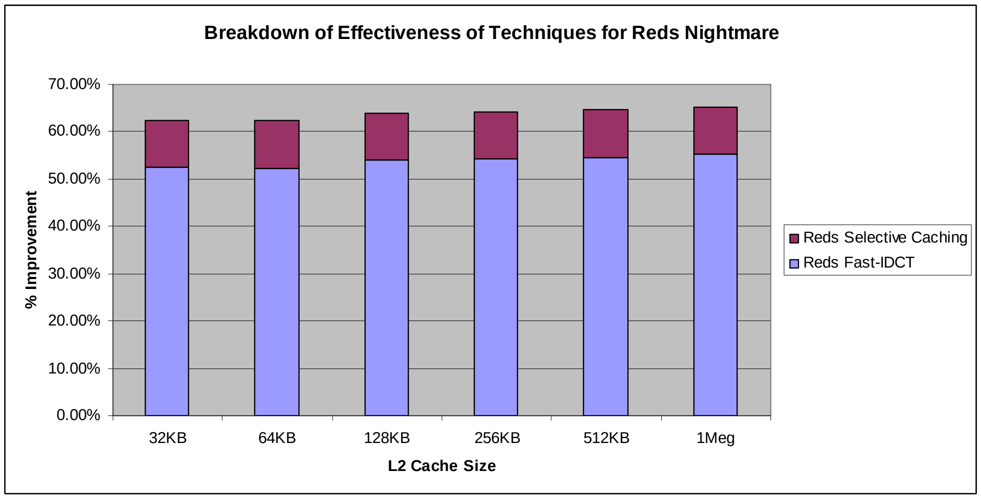
Figure 5 elaborates on the improvements shown in Figure 2 by breaking down the contribution of each technique as a sum-of-parts. Of the 50-62% total improvement over the baseline system, the Selective Caching technique provided a savings of 10-12%. The Fast-IDCT provided an additional savings of 40-50%. Clearly, the Fast-IDCT contributed a much larger amount of power savings to the total improvement. In fact, the Fast-IDCT appears to be about four times more effective than the Selective Caching (10-12% vs. 40-50%) for all of the video clips and cache configurations. It should also be noted that these results are shown as percentages and that the actual magnitudes of the power greatly vary as the L2 cache sizes increase. This will be discussed further below. This means that a 50% power reduction of the system with 512 KB of L2 cache can still consume more power than the baseline system using 32 KB of L2 cache.



(a) *Under* results



(b) *Children* results



(c) *Reds Nightmare* results

**Figure 5: Breakdown of improvement by method and L2 cache size**

## A. Effects of L2 Cache Size

Increasing L2 cache size results in less cache misses which implies fewer RAM accesses will be required. These accesses generate the dynamic power losses for the main memory, which makes up a bulk of the power it dissipates.

Figure 6 shows the total power dissipation of the system for each clip normalized to the 128KB L2 cache size results. It is clear that as the cache size is increased, the total power dissipation of the system also increases.

Looking at the change in power dissipation for all the components, only the following three changed dramatically due to the change in L2 cache size: the L2 cache, the main memory, and the clock distribution tree. As expected, the L2 cache dissipates more power as its size increases due to both static and dynamic power losses. However, the larger cache size resulted in fewer cache misses, which in turn lead to fewer RAM accesses. This reduced the dynamic power dissipation of the RAM, resulting in a 20% savings for cache sizes greater than 64 KB. This is shown in Figure 7. However, these savings are quickly wiped out by the tremendous increase in power dissipation of the clock distribution tree. This occurs because the capacitance of the clock tree is dependent on the die area.

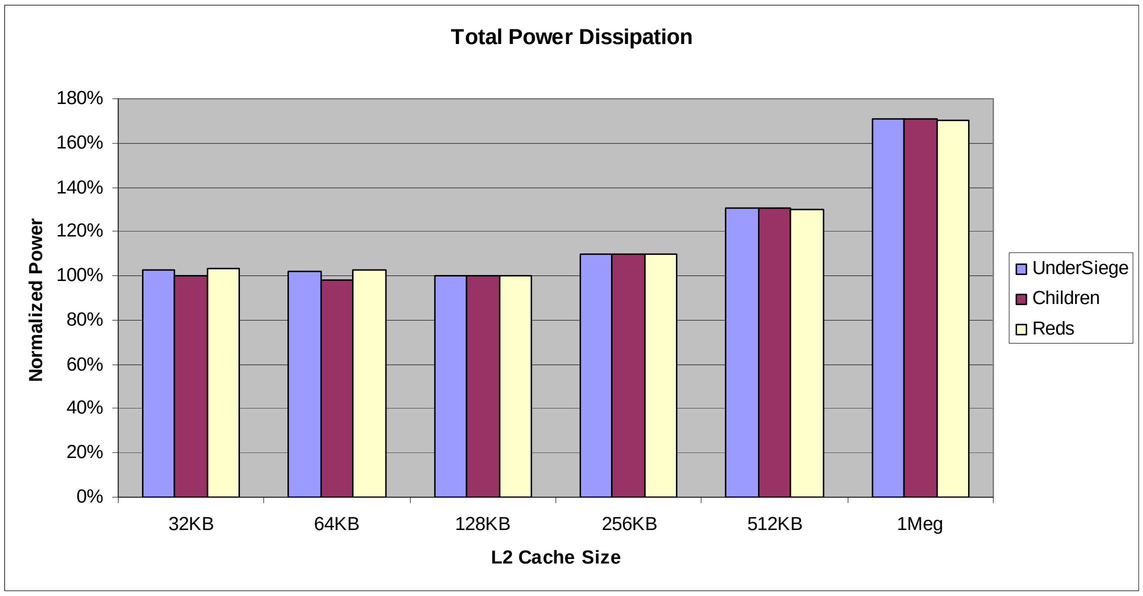
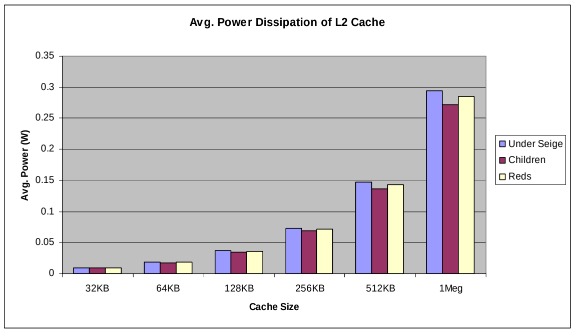


**Figure 6: Normalized system power dissipation for different L2 cache sizes**



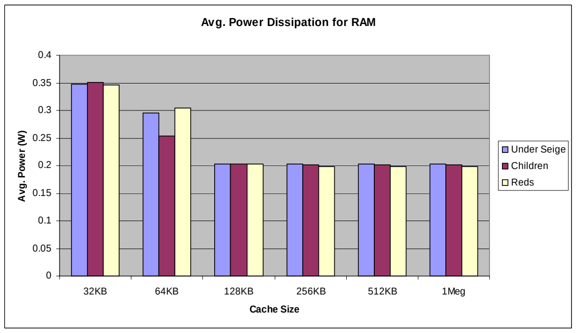


**Figure 7: Power dissipation of L2 cache and RAM**

Larger L2 cache sizes require larger clock distribution trees, which overwhelm the power savings resulting from fewer cache misses. Conversely, smaller L2 cache sizes require less power, but the misses introduced by the small cache size result in higher latency, which would be detrimental to the long-term performance of the system. This means that there is a minimum point where the power generated by the cache misses and the power dissipated by the clock tree are at their lowest. From this research, this point was found to be an L2 cache size of 128 KB.

However, looking at the effects of L2 cache size from a performance point of view offers a slightly different outcome. Table 1 shows the instructions per cycle (IPC) results for each L2 cache size. It is clear that as the L2 cache size increases so does the IPC during MPEG decoding. It must be noted that the IPC grows asymptotically as it approaches the larger cache sizes with the greatest increase occurring between 64KB and 128KB of L2 cache. So clearly a larger L2 cache produces better IPC performance, but as was stated earlier, this comes at the great cost of power dissipation.

Comparing the IPC results to the power dissipation results produces an optimal L2 cache size between 64KB and 128KB. This gives the best power dissipation result while sacrificing a very small amount of performance.

**Table 1: IPC results for different cache sizes**

| | 32KB | 64KB | 128KB | 256KB | 512KB | 1Meg |
|---|---|---|---|---|---|---|
| Under IPC | 1.617 | 1.636 | 1.670 | 1.670 | 1.670 | 1.670 |
| Children IPC | 1.667 | 1.656 | 1.667 | 1.667 | 1.667 | 1.669 |
| Reds IPC | 1.635 | 1.651 | 1.691 | 1.692 | 1.692 | 1.692 |

The conclusion taken from this is that mobile system designers must balance the performance increases of providing a large cache with the power losses coupled to the increased size. This study shows that is possible to find an optimal configuration for a given system and application.

## B. Effect of Fast-IDCT

After determining that more than 128 KB of L2 cache had a negative effect on the overall system power dissipation during MPEG-2 decoding, attention was turned to optimizations that could be made to the decoder to improve performance and power usage. The unmodified decoder uses the standard IDCT algorithm (which is more accurate than the Fast-IDCT algorithm), which resulted in approximately 52,000 divisions per second of video. The modified codec reduced the number of divisions by 69% to approximately 16,500 per second of video. Table 2 shows the decreased cycle count from the reduced number of divisions.

**Table 2: Cycle reduction for Fast-IDCT**

| | Nominal | Fast-IDCT | Cycle Reduction |
|---|---|---|---|
| Under Siege | 690732613 | 297319717 | 57% |
| Children | 643585528 | 234111450 | 64% |
| Reds | 821457401 | 333516402 | 59% |

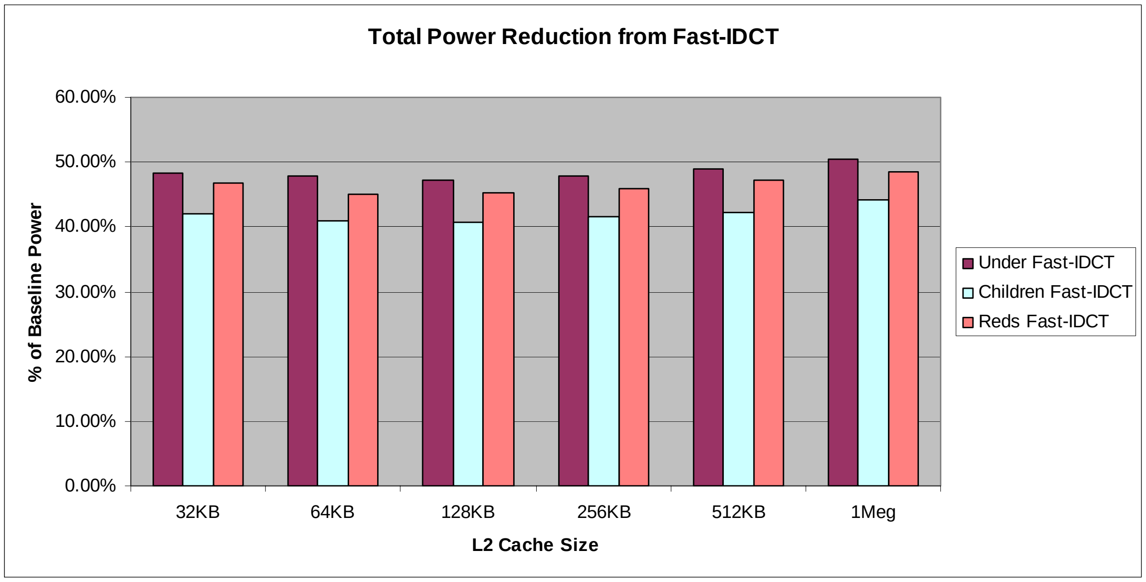


**Figure 8: Comparison of IDCT results**

Figure 8 compares the total power dissipation of the video decoding for the standard codec and the modified codec with Fast-IDCT.

The Fast-IDCT-based decoder requires less than 50% of the nominal cycles to decode each frame so the total power required to decode the video is smaller. This level of improvement occurs because the IDCT requires the most processing time during decode, so improving the IDCT performance will have the greatest impact on overall system power. Table 3 shows the breakdown of decoding time for one second of video. Clearly the IDCT and motion compensation consume a majority of the processing time and therefore it makes sense to focus on improving these two components which compose more then 79% of the decoding time.

**Table 3: Decode time per second of video**

| | % of Decode Time |
|---|---:|
| IDCT | 47.84% |
| Motion Comp | 31.27% |
| Inverse Quant. | 6.69% |
| VLD | 3.74% |
| Other | 10.45% |

Looking deeper into this power improvement confirms that the large reduction in cycles is the greatest contributor to the effectiveness of Fast-IDCT. This is because dynamic power losses are a function of the switching activity of the gates. Fewer cycles implies fewer gates switching, which leads to reduced dynamic power losses. In addition, a system could take advantage of increased idle time through the application of DVS or similar processor-based power management techniques to further reduce power [12, 13, 15].

Further exploring the results of the Fast-IDCT simulation showed some interesting findings. Figure 9 shows the average power dissipation of the different types of caches for the two decoders running with a 128 KB L2 cache. The first point of interest is that the power consumption for L1 I-cache decreases with the Fast-IDCT. These results show that Fast-IDCT performed 69% less L1 I-cache accesses due to the way the algorithm is structured to perform large groups of additions, multiplications, and shifts in close succession.

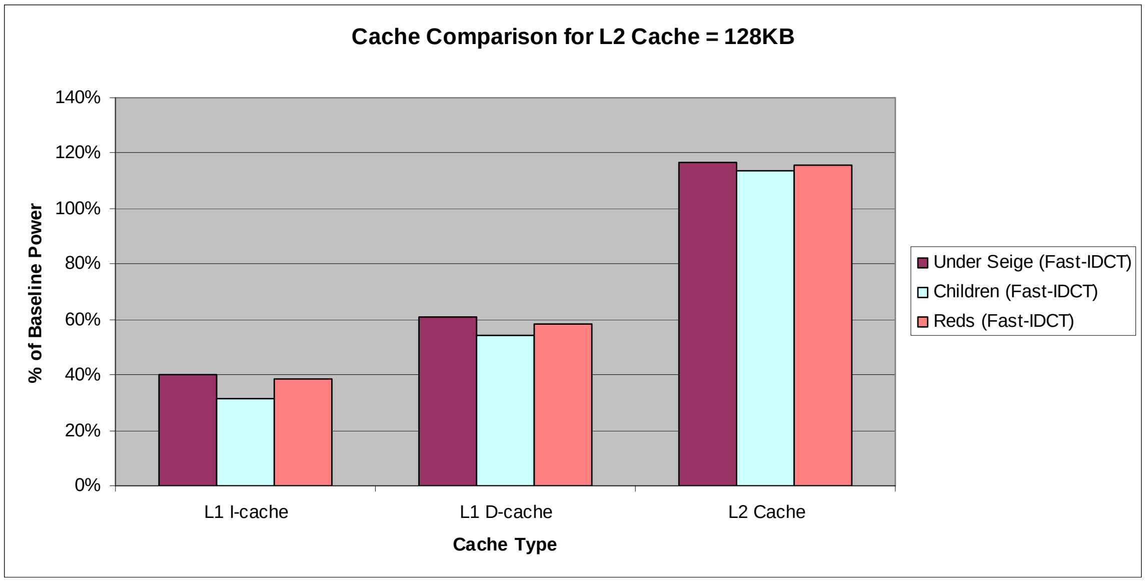


**Figure 9: Effects of Fast-IDCT on different caches**

The total power dissipation by the L1 D-cache decreased as well, where the Fast-IDCT required 46% less L1 D-cache accesses. This resulted in a 46% reduction in total power dissipated in the D-cache. The power dissipation in the L2 cache increased by approximately 17% due to an increase in the number of L2 cache accesses stemming from an increase in both the L1 I-cache and D-cache miss rate.

Table 4 summarizes the average number of cache accesses, misses, and the miss rate per second of video for the different types of cache. Of particular interest is the great reduction in the number of L1 cache accesses generated by the Fast-IDCT algorithm. The number of I-cache accesses was reduced by approximately 66%, while the D-cache accesses were down by nearly one half. This clearly shows that the Fast-IDCT is able to decode at a much more efficient rate than the standard IDCT. However, it is important to note that the I-cache miss rate increased considerably. This is due to the Fast-IDCT using more unique instructions and the instructions are not as tightly bundled as they are in the standard IDCT algorithm. This results in more instructions needing to be retrieved from the L2 cache.

The number of D-cache misses remained fairly constant, which resulted in a higher miss rate relative to the standard IDCT. This is not a concern since the decoder must read in the same amount of encoded information regardless of IDCT algorithm.

Finally, the L2 cache accesses increased due to the increased L1 I-cache misses. Since dynamic power dissipation is a function of switching, the increased L2 accesses increased the power dissipation. However, the number of L2 cache misses did not increase much relative to the increased accesses. This results in only a slight increase in RAM accesses.

**Table 4: Cache statistics per second of video**

|  | Avg. Accesses | Avg. Misses | Avg. Miss Rate |
|---|---|---|---|
| L1 I-cache | 2354435293 | 1600297 | 0.07% |
| L1 I-cache (fast) | 729641489 | 5375616 | 0.74% |
| L1 D-cache | 727236920 | 401636 | 0.06% |
| L1 D-cache (fast) | 389533913 | 416815 | 0.11% |
| L2 Cache | 2233549 | 76303 | 3.42% |
| L2 Cache (fast) | 6033562 | 87311 | 1.45% |

Clearly, due to the large discrepancy in the number of cycles required for each type of decoder, it is difficult to compare the results when looking at the power per cycle. Instead, a broader view of the results in terms of power per frame shows that the Fast-IDCT is effective in reducing the power of the system. This is in part due to the fact that a portion of the dynamic power losses is composed of switching losses. Since the information is being decoded at a much faster rate, this results in more idle time for the system. This means that switching losses in the microprocessor will be greatly reduced as well as the number of accesses to the caches and RAM during these idle periods, resulting in reduced switching losses in these devices as well. The busses between the microprocessor and the memory devices will experience diminished traffic, also reducing the power dissipation. In summary, the improved performance of the Fast-IDCT results in power savings by allowing the system to spend more cycles in an idle or reduced power state, reducing the switching power losses, and therefore reducing the power required to decode each frame.

## C. Selective Caching Results

To simulate the effects of selective caching, the MPEG decoder was first modified to support a special storage algorithm that stores each block directly into the frame buffer as it is decoded rather than storing the finished picture after the entire frame is decoded. This eliminates the need to store the finished output data into the cache, which reduces the amount of traffic between the processor and the cache. The second modification was to the processor simulator itself. A system call was implemented to move the completed output data directly from the processor registers to the frame buffer. This simulates a register-to-main memory instruction, which is similar to instructions included in the architectures of some processors, as was discussed in Section 3.3.

Selective caching is performed on each macroblock of the current video frame as it is completed. This means that instead of storing the complete frame in the L1 D-cache until the entire frame is ready to be transferred to the output, the macroblocks are immediately transferred from the registers to main memory or the frame buffer for displaying. Since the decoded data is typically the only output of the video decoder, this is the only type of data that selective caching was applied to. Due to the fact that other data types used in the decoding process are not written back to the cache they do not make good candidates for selective caching in the decoder. However, selective caching can be applied to all the different output types of the decoder which include progressive and interlaced video which are output as either a complete frame or two fields (as is the case with interlaced video). It can also be applied to the different output formats such as YUV, SIF, and TGA.

To explore the effects of the selective caching design on an unmodified setup, each media clip was run through the simulation on a system configured with 128 KB of unified L2 cache because this cache size has been shown to be power efficient.

Figure 10 shows the relative power dissipation of the clips, and the power dissipation improvement of key components. The power dissipation of the L1 D-cache was reduced by 7% due to the reduced traffic resulting from the direct register-to-main memory storage instruction and the removal of the entire picture storage step. The L1 I-cache power was slightly reduced due to the reduction in instructions needed to transfer. Reduced L1 D-cache misses and L2 cache accessed lead to a 3-4% reduction in the L2 cache power losses.

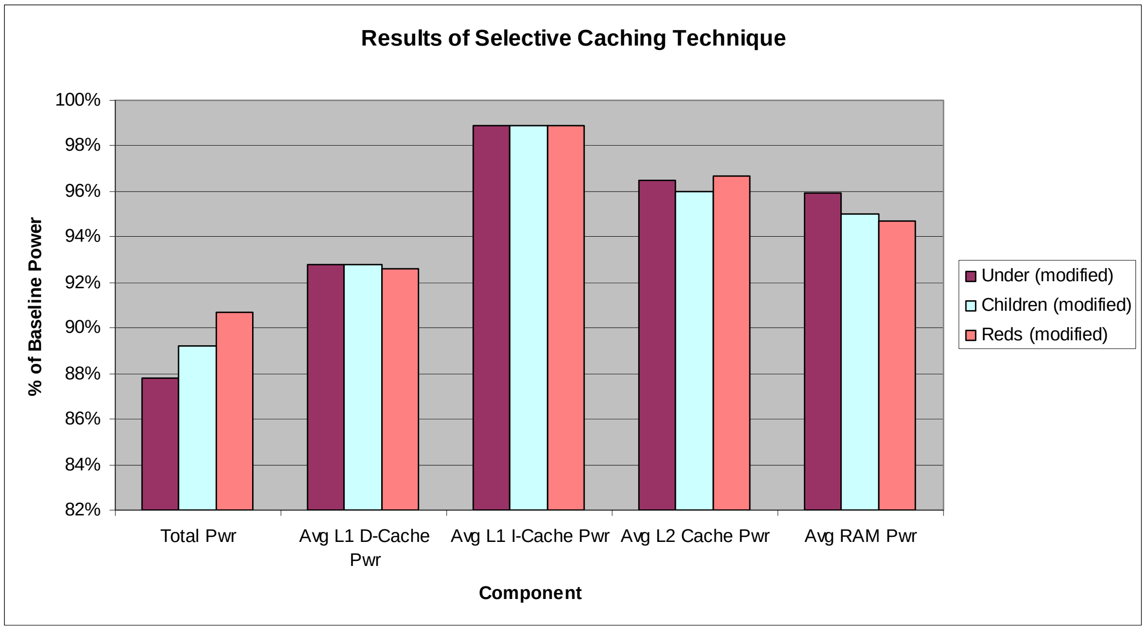


**Figure 10: Results of selective caching**

The power dissipation of main memory was reduced by 4-6% due to a 15% reduction in the number of main memory accesses per frame. This results from fewer L2 cache misses due to the reduced activity on the L2 cache.

These results point at the possibilities of more substantial power savings. Given a more detailed cycle accurate trace of the traffic flow between the processor and main-memory, there may be more types of data or data flows, which can be selectively cached in a similar way. This will open up more free space in the caches and further reduce the number of cache reads and writes which can ultimately lead to main memory accesses.

Soderquist and Leeser raise the possibility of applying selective caching techniques to reference data as well as output data [23]. Recall that reference data consists of the current frame and previously decoded ones used for reconstruction. Unlike output data, reference data experiences periods of read and write accesses as well as write-only phases. This makes a complete selective caching solution like the one applied to output data unattractive because the read and write phases are more efficient when standard caching is used. However, during the write only phases, selective caching can provide some traffic reduction. Instead, they proposed a technique called Selective Write Allocation which bypasses the cache for reference data writes and brings the data back in to the cache for reads. This eliminates the register-to-cache writes for the reference data and reduced the cache-memory traffic by 25% when applied to both output and reference data.

Further detailed examination of the behavior of the different MPEG data types may reveal that power dissipation can be reduced through the application of caching techniques throughout the MPEG decoding process. However, the selective caching technique used in this study was intended only for use on write only data. More advanced techniques would need to be used on data types who have read and write characteristics.

## 5. Conclusion

Controlling the flows of data between the different levels of cache and the main memory is an important factor in reducing the power dissipation during MPEG decoding on a mobile system. Specifically, the interaction between the data caches and the main memory are large contributors to the power loss of the system due to the inherent dynamic nature of video data. The contribution of the instruction cache and its interactions with main memory also show improvement from these techniques, although the gains are not as great as those produced by the Fast-IDCT. However, a reduction in the number of high latency instructions can improve performance and allow for the application of DVS and other performance-based power reduction techniques.

From the results presented in Section 4, it is clear that there are gains to be made in reducing power losses during MPEG decoding. Cache size has a large influence on the power of the overall system, but mostly due to increased passive power dissipation in the cache and the clock tree. This showed that keeping the cache sized between 64KB and 128KB was important in a system where power reduction is critical. The Fast-IDCT showed strong promise through its great reduction in cycles required to decode each frame, making it a strong candidate for inclusion in one of the many dynamic power management methods. Finally, selective caching of the output data proved to be effective in eliminating unnecessary data storage and traffic, reducing power of both the data caches and the main memory.

For future work, an improved simulation environment with the ability to simulate main memory more accurately as well as a network interface and corresponding bus would allow for more detailed analysis. In addition, a more state-of-the-art video codec such as H.264 may provide more flexibility for manipulating the flow of data which has been shown to be effective in reducing power dissipation. Furthermore, selective caching of the output data was successful in reducing the total power of the system and a deeper examination of caching techniques could uncover more effective caching methods.

## Bibliography

[1] P. Soderquist, and M. Leeser, “Memory Traffic and Data Cache Behavior of an MPEG-2 Software Decoder”, *Proceedings ICCD ’97: International Conference on Computer Design*

[2] P. Soderquist, and M. Leeser, “Optimizing the Data Cache Performance of a Software MPEG-2 Decoder”, *ACM Multimedia ’97*, November 1997

[3] W. Chen, C. Harrison, and S. Fralick, A fast computational algorithm for the discrete cosine transform, IEEE Trans. Com., Vol. COM-25 (9), pp. 1004-1011, Sept. 1977.

[4] Z. Wang, Fast algorithm for the discrete W transform and for the discrete Fourier transform, IEEE Trans. ASSP, Vol. ASSP-32 (4), pp. 803-816, Aug. 1984.

[5] J. Liang, S. Swaminathan, and R. Tessier, “aSOC: A Scalable, Single-Chip Communications Architecture”, *Proceedings of the IEEE International Conference on Parallel Architectures and Compilation Techniques*, Oct. 2000.

[6] *Integrated Power Management for Video Streaming to Mobile Handheld Devices*, S.Mohapatra et al.

[7] Intel StrongARM 1110 Datasheet

[8] ARM7TDMI Datasheet

[9] A. H. Farrahi, and M. Sarrafzadeh, “FPGA Technology Mapping for Power Minimization”, *4th International Workshop on Field Programmable Logic and Applications*, pp. 66-77, Sep. 1994.

[10] B. Lee, E. Nurvitadhi, R. Dixit, C. Yu, and M. Kim, "Dynamic Voltage Scaling Techniques for Power Efficient Video Decoding", *Journal of Systems Architecture,* Vol. 41, Issues 10-11, Oct.-Nov. 2005, pp. 633-652.

[11] D. Son, C. Yu, and H. Kim, "Dynamic Voltage Scaling on MPEG Decoding", I*nternational Conference of Parallel and Distributed System (ICPADS)*, Jun. 2001.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy", *First Symposium on Operating Systems Design and Implementation*, pp. 13-23, Nov. 1994.

[13] T. Pering, T. Burd, and R. Broderson, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms", *International Symposium on Low Power Electronics and Design*, pp. 76-81, 1998.

[14] Y. Shin, K. Choi, and T. Sakurai. "Power Optimization of Real-time Embedded Systems on Variable Speed Processors", In *Proceedings of International Conference on Computer Aided Design*, pp. 265-268, 2000.

[15] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors", Proc. *Int'l Symp. on Low Power Electronics and Design*, pp. 197-202, 1999.

[16] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU", *First International Conference on Mobile Computing and Networking*, Nov. 1995.

[17] D. Son, C. Yu, and H. Kim, "Dynamic Voltage Scaling on MPEG Decoding", I*nternational Conference of Parallel and Distributed System (ICPADS)*, Jun. 2001.

[18] M. Mesarina and Y. Turner, "Reduced Energy Decoding of MPEG Streams", *ACM/SPIE Multimedia Computing and Networking 2002* (MMCN '02), Jan. 2002.

[19] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, “Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder”, *Proceedings of the International Conference on Computer-Aided Design*, 2002.

[20] Ultra SPARC VIS Instruction Set Architecture

[21] PA-RISC 2.0 Instruction Set Architecture

[22] *Leakage Current: Moore's Law Meets Static Power*, Computer Magazine

[23] P. Soderquist and M. Leeser, “Increasing Data Cache Efficiency for MPEG-2 Video Decoding”

[24] J. Liang and T. D. Tran, “Fast Multiplierless Approximations of the DCT With the Lifting Scheme”, *IEEE Transactions on Signal Processing, Vol 49, No. 12*, 2001

[25] Micron Technologies MT48LC32M8A2 SDRAM Datasheet

[26] L. John and R. Radhakrishnan, “A Selective Caching Technique”