Memory Access and Computational Behavior of MP3 Encoding

by

Michael Lance Karm, B.S.E.

Report

Presented to the Faculty of the Graduate School

of The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Masters of Science in Engineering

The University of Texas at Austin

December 2003

Memory Access and Computational Behavior of MP3 Encoding

APPROVED BY

SUPERVISING COMMITTEE:

_____

_____

Memory Access and Computational Behavior

of MP3 Encoding

by

Michael Lance Karm, M.S.E

The University of Texas at Austin, 2003

SUPERVISOR: Lizy Kurian John

Multimedia applications are a substantial workload for modern computing platforms. However, most processors lack architectural refinements that would enable ideal levels of computational efficiency. This report provides an understanding of the execution characteristics of audio compression applications in order to understand the bottlenecks that bound performance. These results could be used to design optimal multimedia processor architectures.

Characteristics of scientific and multimedia applications are typically represented by a set of kernels contained in a benchmark suite. Entire applications are seldom profiled to obtain a complete appraisal of the actual workload imposed on the system. This report contains a more complete analysis of an MP3 encoder application and presents the significant aspects of its behavior. Contributions include instruction profiling of the application and its major routines, memory access characterization, and a measurement of computational demand.

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

Multimedia applications have become a significant workload for General-Purpose Processor (GPP) and Digital Signal Processor (DSP) platforms [2][17]. The term "multimedia" generally applies to programs that process image, video, and/or music media. A multimedia application performs one or more of the following tasks: compression, decompression, editing, and/or encryption. Although GPPs are commonly used to process multimedia data, it is well accepted that most of today's processors are not ideal for these applications [7][4][35][13]. It has been determined that increasing complexity in multimedia algorithms drives the need for high-performance media-capable processors that are sensitive to power and cost constraints [16][17][30][10][24]. The widespread use of multimedia applications has motivated workload analysis projects that strive to provide information which will assist in the design of more efficient architectures.

A typical personal computer user invokes a variety of media encoders and decoders in everyday activities; however, audio processing remains the dominant activity. One aspect of this is demonstrated by recent statistics that attribute an average of more than 3 billion downloads per month to music files in 2001 [6]. *Fortune* magazine predicts that music commerce will be drastically affected by the capabilities of the Internet and personal computers. Clearly, this level of demand validates attention to processors that more effectively handle audio compression and decompression applications.

In order to design a better processor, the workload must be fully understood. Several studies have evaluated the key aspects of multimedia programs by analyzing benchmarks with signal processing kernels and multimedia application segments, but few have completely explored an entire application. Although the small segments of a program represented by a benchmark typically represent a large percentage of its behavior, many properties of an application remain overlooked.

1.1 Objective

This report will analyze the execution of an MP3 encoder program on an UltraSPARC superscalar general-purpose architecture. The MP3 encoding algorithm was chosen because it exhibits several properties common to media processing applications, and it is frequently used both privately and commercially. General purpose instruction set architectures provide a good framework for application characterization research. The wide variety of instructions enables the compiler to choose instructions that more closely represent the high-level algorithm. The results obtained from characterization on a limited instruction set machine or custom architecture restricts the relevancy of the conclusions to that specific implementation.

Rather than study benchmarks or code segments, this report evaluates the complete behavior of a real media application. Utilizing benchmarks narrows the

focus to the code that would ideally dominate the dynamic instruction stream for a desired algorithm. Unfortunately, this disregards many overhead operations and interactive second-order effects caused by routines that are required for program flow and high-level formatting rather than true data processing. Benchmark analysis also limits the understanding of how each kernel contributes to the overall execution statistics of the complete application.

The analysis will describe the key attributes of the MP3 application, and then identify and characterize the functions that dominate the dynamic instruction stream. The Shade simulator for SPARC micro-architecture is used to capture and review the dynamic instruction stream of the MP3 encoder application. These results are used to determine the significant functions and factors that affect execution performance. An emphasis will be placed on understanding the load/store bandwidth requirements and instruction mix of the function calls which require the majority of instruction cycles.

1.2 Contributions

The central focus of this report is to provide an understanding of the execution characteristics of audio compression applications. This information could be used to optimize custom processor architectures for multimedia applications. In addition to providing an overview of the MP3 encoder algorithm, this report makes the following contributions: a routine-level profiling of the

encoder execution, instruction mix analysis for the application as a whole and for each of the seven encoding stages, memory access characterization, and computational workload analysis. Properties that pose bottlenecks to the application execution are identified throughout the analysis.

1.3 Organization

An introduction in Chapter 1 will preclude related work on the topic of workload characterization and media processing analysis found in Chapter 2. A description of MP3 audio encoding and a specific implementation of this algorithm are described in Chapter 3. The fourth chapter describes the methodology and analysis tools used to profile the MP3 encoding application. In Chapter 5, results of the simulations are presented along with an explanation of the more significant qualities. A conclusion in Chapter 6 will summarize the results and recommend aspects to consider when designing more efficient media processing hardware.

2 RELATED WORK

Without a sufficient understanding of typical processor workloads, it is difficult to make architectural enhancements that improve processor performance. Often benchmarks are used to analyze the characteristics of an application or performance of a processor [26][11][20]. Based on these results and educated guesses, several studies have proposed architectural enhancements to general purpose processors (GPPs) that reduce execution time of multimedia workloads. Although the operation of a complete multimedia application is typically not considered, significant speedup for many algorithms has been achieved with the addition of SIMD extension technologies to GPPs, such as Motorola's AltiVec, Intel's MMX, Sun VIS, and HP's MAX2, that enable parallel processing of data [7][33][35]. Additional performance gains are possible with the aid of comprehensive application analysis.

2.1 Memory Access Analysis

John et al. conducted a comprehensive study of the memory aspects of scientific workloads [15]. Several floating point benchmarks were analyzed for their memory access characteristics. A metric called program balance is introduced that describes the inordinate contribution of overhead loads and stores to the overall instruction stream of a program on a RISC GPP. John et al. found that more than 66% of the dynamic instruction stream is devoted to memory access instructions or

operations related to memory address calculations. It was concluded that the superscalar hardware features are underutilized as a result of this memory activity.

Thus the load/store demands coupled with a standard external memory hierarchy becomes the primary bottleneck and inhibitor to potential speedup that could be obtained from adding additional arithmetic units or increasing processor core frequency [15][10][16][13][7]. In addition to overall memory bandwidth analysis, studies often include the effects of cache performance of media kernels [28]. Comparisons between the cache utilization characteristics of SPECint95 benchmarks and multimedia applications attempt to understand the multimedia processing potential of traditional GPP architectures [29]. Sohoni et al. compare amount of data references per instruction of multimedia benchmarks with Specint95 in order to obtain a relative measure of memory system demand [29]. It was determined that in many cases multimedia applications place a lower demand on memory than typical integer applications. The memory access characteristics for the SPEC2000 benchmarks executing on the Itanium architecture is presented by Serrano et al. [26]. This analysis includes information on memory access patterns induced by looping structures and a detailed description of how execution performance is affected by pipeline stall cycles.

2.2 Multimedia Application Execution Analysis

The study of memory bandwidth is extended by Lee and John to include multimedia algorithms and other characteristics of the program code [19]. This study revealed that 90% of the memory accesses could be considered overhead transactions that do not contribute to the true computation of the algorithm. Execution time is noted to be directly proportional to percentage of overhead memory accesses. Another contribution from Lee and John is an instruction mix analysis for a variety of multimedia application kernels which demonstrates the fact that a significant percentage of instructions are ALU operations. A final analysis presented the data-level parallelism present in typical multimedia kernels.

Benchmarks were the basis of a performance analysis of Pentium-II and DSP processors found in [31] and [32]. In these papers, the effectiveness of the processor architectures is measured by cycles per instruction, cache performance, utilization of hardware acceleration units, and overall speedup obtained by taking advantage of advanced architectural features. A profile of the multimedia applications determined that these programs have large amounts of data parallelism, but the branch characteristics inhibits the significant speedup anticipated by the VLIW and SIMD architectures.

2.3 Multimedia Acceleration Hardware

Proposals have also explored ways to improve GPPs with dedicated hardware enhancements beyond standard SIMD extensions have also been explored. The addition of a specialized Huffman coding unit can dramatically enhance many multimedia applications [35]. Moravie et al. recommend a memory address co-processor which would eliminate several problems common to GPPs [22]. The MediaBreeze enhancement proposed by Talla et al. addresses shortcomings, including limitations in available GPP memory bandwidth and addressing capabilities, that restrict the supply of sufficient amounts of data to computational elements [30]. Additional ALUs will not be efficiently utilized if a system exhibits inherent data flow bottlenecks. Significant speedup is observed with this new architecture tailored to the memory addressing characteristics and computational requirements of these programs.

3 MP3 AUDIO COMPRESSION

The Motion Picture Experts Group (MPEG) working group was formed in 1988 to define standards for video and audio compression. Published in 1993 by the International Standards Organization/International Electrotechnical Commission (ISO/IEC), the MPEG-1, ISO/IEC 11172 standard includes specifications for 1-2 Mbps video compression and three layers of audio compression of media [1][14]. The term "MP3" is commonly used in reference to the MPEG-1, Layer 3 specification for audio coding [3]. The MP3 standard defines the decoding process, bitstream format, and encoding strategy to establish a framework for an optimal balance between the final bitrate of the compressed material and perceived audio quality of the reconstructed signal. The core algorithms and theory were primarily developed by the Fraunhofer Institute, which holds several patents on this encoding method, and later adopted by the MPEG and ISO/IEC committees [8].

3.1 Features of MP3 Compression

The MPEG standard for audio compression can accept audio sources recorded at 32 kHz, 44.1 kHz, and 48 kHz sampling rates. A standard digital Compact Disk (CD) contains two channels of uncompressed 16-bit linear Pulse-Code Modulation (PCM) data sampled at 44.1 kHz. In this format, each sample represents an analog voltage at one point in time. The resulting bitstream requires a

data throughput of 1.411 Mbps to convey this audio information. A well-encoded MP3 file can achieve near CD-quality audio reproduction at datarates as low as 128 Kbps – a compression ratio exceeding 10:1.

An MP3 bitstream is a compressed format that contains only the critical information required to represent the aspects of the source material that humans can readily perceive. The bulk of MP3 processing entails the classification and removal of the imperceptible information in the original source. As recommended in the MPEG standard, most MP3 encoders contain algorithms designed around the models of the human auditory system to determine "irrelevant" details of a source that can be removed without adding excessive levels of audible distortion [25]. This type of lossy algorithm is classified as a perceptual encoder [21].

3.1.1 Lossy Compression

Compression algorithms are typically classified as either lossless or lossy [34]. The former preserves the original exactly upon decompression, whereas the latter removes information to achieve a higher compression ratio. Typical examples of lossless audio compression formats include AudioPaK, LTAC, MUSICompress, OggSquish, Philips, Shorten, Wonarc, and WA [12]. These algorithms are instrumental to the distribution of high-fidelity media, but lack widespread acceptance due to their relatively low compression of no more than three-to-one. The popular alternative is lossy compression algorithms, such as MP3, WMA, RA, and AAC.

The goal of media compression algorithms is to encode the input audio data such that the resulting file is drastically smaller than the original without undue loss of fidelity. The compressed data thus requires fewer resources to store audio files and less network bandwidth to transfer this data. Data compression can also reduce the cost of network infrastructure required to satisfy the huge consumer demand for media [9]. Successful lossy algorithms accomplish this goal without a significant or noticeable sacrifice in image or audio quality. This type of audio encoder attempts to intelligently remove information from the source that is not necessary to reproduce the acoustic experience. To achieve this result, compression applications typically contain a complex multi-step process of input data transformations and analysis that funnel into one or more compression techniques. Many advanced applications utilize signal processing routines including filterbanks, Fourier transforms, and/or discrete cosine transforms.

Due to its effectiveness as lossy algorithm for audio compression, MP3 is one of the most pervasive multimedia formats used in by today's personal computers. The overwhelming acceptance is due to several factors including convenience, high quality results, and a significant reduction in file size as compared to the original PCM format [3].

3.1.2 Critical Bands

Perceptual encoders can typically achieve a higher quality result for a given bitrate compared to PCM coding techniques by removing unnecessary information

that is undetectable in the time domain. An understanding of the human perception of audio signals is crucial to the success of a perceptual codec. The MP3 encoding process converts PCM domain samples to the frequency domain to identify spectral redundancy in the source and take advantage of the psychoacoustic properties of human hearing [34]. Once in the frequency domain, the encoder evaluates several aspects of the signal.

The range of frequencies detected by humans is internally divided into several mutually exclusive frequency regions. The width of these regions, determined by experimentation, is a function of frequency. It was discovered that the frequency bandwidth of the regions centered at lower frequencies bands is smaller than those at high frequencies [1]. It is important to note that theperception of acoustic energy at a frequency within one of these "critical bands" can be impacted by other signals that fall within this same frequency region. This property, also revealed through human experimentation, illustrates the importance of identifying these critical bands [23][25]. In order to properly understand the interaction of a full-spectrum signal, the perceptual algorithm must contain knowledge of the critical band boundaries and the attributes that allow various frequencies to interfere with each other.

3.1.3 Masking

Several properties of a signal inhibit a human's perception of the entire audio experience. These factors relate to the signal amplitude as a function of time

and/or frequency. The first deals with the ability to hear in a quiet environment. Although hearing is often considered to detect signals from 15 Hz to 20 kHz, it has been determined that the human ear is less sensitive to energies at the lower and higher ends of this spectrum. An initial perceptual analysis compares a signal's spectral content against thresholds of hearing determined by human auditory perception models. If the energy in any frequency is below that which can be detected, it is deemed less important for original source reproduction [21]. Derived from descriptions of the psychoacoustic modeling described by Noll in [23] and Ambikairajah in [1], Figure 3.1 depicts the absolute threshold of hearing as a function of frequency.

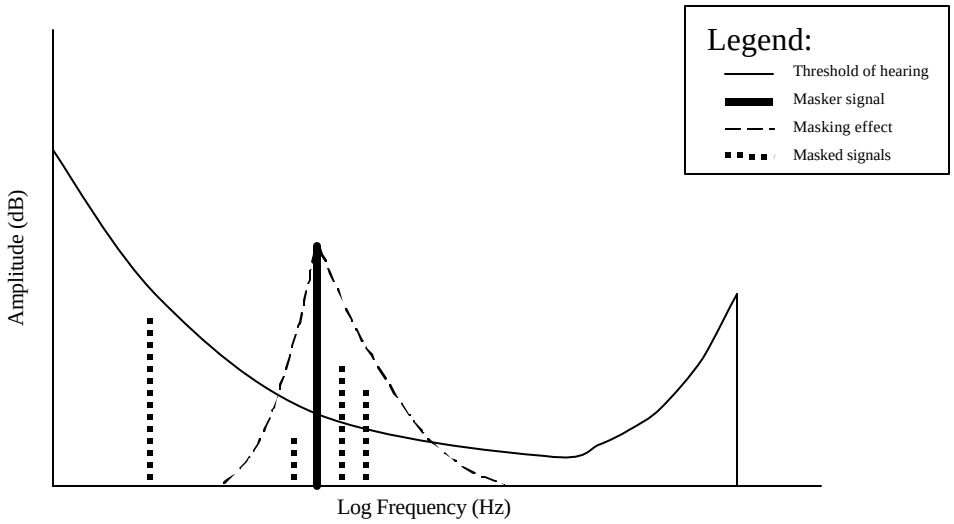

Figure 3.1: Psychoacoustic Masking Effects

Beyond the static limitations of hearing, a typical signal contains information that dynamically impedes the ability to perceive all of the information inherent to that signal. This phenomenon is known as masking. The 16-bit, 22 kHz signal contained on a CD includes vast amounts of information that a human cannot detect due to signal masking effects. As determined by human auditory modeling, significant energy in a frequency band diminishes the ability to perceive energy in nearby frequencies [23]. Figure 3.1 shows how several signals are masked by a dominant "masker" signal, and the threshold of hearing.

A predictable, or otherwise tonal, signal induces different masking behavior than a non-tonal signal (something perceived as noise) [1]. In addition to measuring the intensity of potential masking signals, the psychoacoustic modeling must also determine the amount of tonality present in each critical band to accurately resolve the masking function. This tonality classification requires a linear prediction based on data from the last two frames. The masking threshold algorithm relies on this information when deciding how energy at one frequency spreads a masking shadow across its critical band [25].

Shapes of the instantaneous masking effects are calculated at run-time to adjust the absolute threshold of hearing to a new dynamic threshold based on current stimulus. The spectral masking effect also exhibits a temporal component that inhibits the ability to perceive signals in a critical band after the presence of a

strong masker signal for some time into the future [34][23]. The frequency domain representation of the original signal is analyzed for spectral and temporal masking effects to identify additional irrelevant information.

3.1.4 Window Size

The transform from time to frequency domain translates approximately 26 ms of audio into the corresponding frequency representation. This sample size was determined by the MPEG organization to allow for tradeoffs of reasonable frequency and time resolution. However, transient affects such as an instantaneous change from a relatively small signal to large signal at one or more frequencies causes a processing effect known as "pre-echo". This is a situation where audible quantization errors occur in the reproduced signal before the instantaneous event [23]. To address this issue, MP3 encoders can temporarily switch to a smaller window size with a higher time resolution, at the sacrifice of frequency resolution [1]. In this case, the quantization error is focused in a smaller region of time and thus is less perceptible.

3.1.5 Bit Allocation

The final product of a psychoacoustic modeling process is a recommendation for bit allocation to the quantization stage. The most significant reduction in bitrate is accomplished by reducing the number of bits used to represent energy amplitude in as many frequency bands as possible. However,

representing a signal with fewer bits introduces a deterministic amount of quantization noise into the reconstructed signal. Therefore, this process must be carefully implemented so as not to destroy the original signal or add annoying artifacts. A signal-to-mask ratio (SMR) is determined for each of the subbands to quantitatively convey psychoacoustic information to the quantization stage. Frequency bands with a high SMR are allocated more bits than the more heavily masked (low SMR) regions. Higher quantization noise, induced by utilizing fewer bits of resolution, is less perceptible in regions with a low SMR [34].

3.2 MP3 Encoder Operation

Specified at the time of invocation, the encoder is restricted by a limit on the final number of bits available to represent the amplitude of each frequency component of an audio recording. When compressing a PCM source, the user selects the maximum bitrate of the compressed file, from 96 Kbps to 320 Kbps. The encoder application reads a set of samples that constitute a frame (typically 1152, assuming that the original is sampled at 44.1 kHz) and converts this time-domain signal to a frequency-domain representation divided across subbands of frequency.

The compression occurs in the next stage where a quantization algorithm distributes the available output bits, determined from the user-specified bitrate and frame size, across the frequency bands. Based on SMR input from the

psychoacoustic analysis of this set of data, samples in the frequency domain are allocated bits and quantized – represented as a digital value with a minimum number of binary bits – according to their relevancy to the original audio signal. Subbands determined to be more significant during this sample period are allocated more bits to represent energy at that subband and will therefore more accurately represent the true amplitude of that spectral content. Less significant frequencies will receive fewer bits and which results in higher quantization noise. This noise is the difference between the true value, and the value represented by an insufficient number of digital bits [34].

After quantization, the algorithm iterates to ensure an optimal level of quantization noise in the significant subbands while not creating excessive noise in other regions of the spectrum. The control flow of the iteration loop routines rely on input from the psychoacoustic modeling and calculations that assess the quantization-induced noise in each band. The final encoded data is a frequency-domain representation of the original analog signal that maintains satisfactory levels of perceived audio quality while adhering to the bitrate restriction of the output data stream. This data is stored in a ".mp3" file along with sideband information and appropriate headers.

3.3 LAME Implementation of MP3 Encoding

For this study, the LAME ("LAME Ain't an MP3 Encoder") source code version 3.93, released in December 2002, was selected [18]. The LAME project is an open-source MP3 encoding application that evolved around reference code published by the ISO as a starting place for MP3 encoder algorithms. As described by its name, LAME is solely distributed in source code form as a research project for the study and enhancement of MPEG audio algorithms. LAME is continually updated through contributions from the open-source community continually in order to increase performance, both in runtime and audio quality. The distribution files for this project contain all of the necessary code and Makefiles to target a variety of architectures, including many UNIX environments.

Although the ISO MP3 sample code is a functional implementation of an MP3 encoder, it is not optimized for execution on a general-purpose processor with cache memory and other performance-enhancement features in mind. By contrast, aspects of the LAME encoder include cache and algorithm optimizations [18]. An analysis of the LAME encoder thus provides information that can be used to understand realistic multimedia workloads executed on standard PC environments. In addition to enhancements on the basic encoding framework provided by the ISO source, the LAME encoder incorporates the GPSYCHO GPL psychoacoustic model for noise shaping. GPSYCHO is another open-source initiative whose goal is to provide the best modeling that will drive the highest quality MP3 encoding.

Figure 3.2 shows a graphical representation of the encoding process. The implementation of the MP3 encoding stages is described in the following sections. MP3 encoders typically adhere to these stages when compressing a PCM audio source:

I. Windowing and Polyphase Filtering

II. Modified Discrete Cosine Transform (MDCT)

III. Fast Fourier Transform (FFT)

IV. Psychoacoustic Modeling

V. Quantization

VI. Huffman Entropy Encoding

VII. Bitstream Formatting

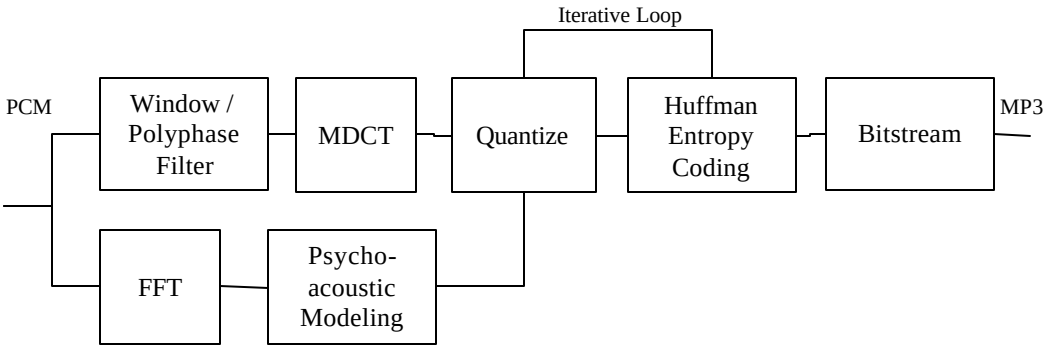

Figure 3.2: MP3 encoding stages

3.3.1 Windowing and Polyphase Filtering

The first two stages use an overlapping window function and a polyphase filterbank to integrate a set of 1152 PCM samples into the analysis window and divide this time-domain signal into 32 equally-spaced frequency subbands. These subbands are not the most accurate representation of the human's critical bands, but it is an acceptable compromise based on resulting quality and algorithm complexity tradeoffs [25]. Figure 3.3 contains the equation for the polyphase and windowing filtering. These operations add an acceptable level of nonlinear artifacts to the input signal. The minimal degradation attributed to the polyphase filtering is easily overshadowed by its fast runtime performance when compared to alternate filterbank implementations.

$$s_t[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] \times \left(C[k + 64j] \times x[k + 64j] \right)$$

$$M[i][k] = \cos\left[\frac{(2 \times i + 1) \times (k - 16) \times p}{64} \right]$$

Figure 3.3: Windowing and polyphase filterbank equations

3.3.2 Modified Discrete Cosine Transform

Immediately following the windowing and polyphase operations, an MDCT converts the windowed samples into an easily quantized set of 576 frequency domain samples. Depending on window size, each MDCT will evaluate 6 or 18 points for each of the 32 subbands. The MDCT equation is shown in Figure 3.4. This digital signal processing algorithm exhibits predictable memory access patterns to access the input data and coefficient tables. Computationally, this algorithm requires extensive multiply/accumulate (MAC) operations.

The software implementation of these algorithms is optimized to take advantage of cache organization, but cannot avoid the substantial memory accesses required to read tables, transfer the intermediate data, and store the final results. A proposed replacement for the MDCT is a Fast Harley Transform that has been proven to reduce memory transactions for the inverse MDCT operation which is the most significant part of the decoding process [27].

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos\left[\left(n + \frac{M+1}{2}\right)\left(k + \frac{1}{2}\right)\left(\frac{p}{M}\right)\right]$$

Figure 3.4: MDCT equation

3.3.3 Fast Fourier Transform

The third and forth stages, depicted as FFT and Psychoacoustic Modeling in Figure 3.2, can occur simultaneously with the transform blocks described above. The 1024-point FFT is similar to the MDCT in function and implementation. However, this transform was selected because it is less computationally intensive than a variety of alternatives and the result is a high-resolution representation of the entire frequency spectrum for the current frame.

3.3.4 Psychoacoustic Modeling

The psychoacoustic modeling stage is fairly irregular in contrast to the previous three signal processing algorithms. This function utilizes the data generated by the FFT and compares the energy components at each frequency with pre-determined tables to rank the spectral energy against thresholds of human hearing. Following tonality calculations, this stage computes the local instantaneous masking effects and temporal masking effects from analysis of previous frames. The computation results in a set of Signal to Mask Ratio (SMR) values for each band [25]. This function is also responsible for the decision to switch to small window sizes to compensate for potential pre-echo artifacts. A short FFT is used in for this analysis to match the smaller MDCT. These results are used to determine how many bits to allocate to each frequency line for acceptable audio quality.

Psychoacoustic analysis requires in two significant behaviors: predictable MAC computations to evaluate masking intensity and irregular control flow as the function determines and records the dynamic signal masking effects. Thus, this function is dependent on both the current sample set as well as a history of previous samples. The dynamic operation of this routine can be significantly impacted by the characteristics of the PCM source.

3.3.5 Quantization

The final substantial stages of the MP3 encoder include the quantization and Huffman coding. These are organized as two nested loops, typically referred to as an inner loop for bitrate control and outer loop for noise control [8]. The inner loop contains the core quantization algorithm and Huffman coding. On the first pass through this loop, the samples are quantized and the resulting data is condensed with Huffman coding. To achieve the desired bitrate, a global scaling of the subbands is altered and this process is repeated until the final output contains no more than the maximum allowed number of bits, thus controlling the bitrate. Quantization methods exhibit a predictable control flow as each sample is multiplied against a table.

After the inner loop compresses the frequency domain data, functions in the outer loop evaluate the induced quantization noise. If the resulting distortion from the original signal is greater than the acceptable level determined at the perceptual modeling stage, individual factors are adjusted to scale the offending subbands in

an effort to reduce quantization noise. These operations are divided into three phases: a straight-forward noise analysis which compares the results with the original, a noise balancing phase which adjusts the scalefactors for the bands which require more bits, and a decision process to evaluate the results and restrict the number of attempts to optimize the overall distortion.

3.3.6 Huffman Entropy Encoding

Huffman entropy encoding is a lossless algorithm typically implemented with a large amount of tables. The Huffman algorithm attempts to reduce the final output size by replacing commonly-occurring sequences with a smaller binary representation. As it processes the data, a Huffman coder searches for patterns and selects one of 32 entropy tables. After table selection, the raw bitstream is replaced with its optimal entropy encoded representation. This procedure involves numerous comparisons, table lookups, and data-dependent control flow breaks as the datastream is analyzed and replaced by Huffman codes.

3.3.7 Bitstream Formatting

The final set of functions transfer the data generated in the compression stages into an MP3-compliant bitstream. Several copy operations collect data scattered throughout the encoding stages into a single repository. Although this process is not computationally intensive, this task requires significant processor bandwidth to locate and move data. The final stages of MP3 file creation also

include the formalization the appropriate headers, cyclic redundancy check calculation, and other formatting operations.

4 TOOLS AND METHODOLOGY

In order to better understand the limitations of general purpose processors and how to enhance their performance when executing MP3 compression, a detailed analysis of this workload must be performed. This section explains the software tools, methodology, and files used in the analysis.

4.1 Application Analysis Tools

The Shade simulation engine was selected to more efficiently study the execution characteristics of the MP3 encoder algorithm. Shade generates custom traces and provides a framework to simulate and analyze aspects of application execution on a processor [5]. The primary targets for Shade analysis are the SPARC v8 and v9 microprocessors. Functions provided in the Shade suite in addition to the simulation kernel expand its value beyond that of a simple trace mechanism. These analysis tools allow the user to customize Shade's behavior in order to design an elaborate analysis tool which examines a specific quality of a program's execution. Optional libraries include opcode selection, address range specification, and several built-in functions that generate a variety of execution statistics.

A typical analyzer built on Shade will trace a subset of the machine opcodes and memory regions. The Shade infrastructure captures the dynamic instruction traces and this data is used to correlate the static program code to the actual

execution trace. A Shade analysis can also record the interactions between the significant routines of a program. This data enables the study of routine profiling and the percentage of instructions that each contributes to the dynamic instruction stream. Further analysis can evaluate the effects of non-retired instructions, the dynamic efficiency of an algorithm implementation, and the memory architecture utilization.

4.1.1 Shade Spixstats and Spixcounts

Designed on the Shade trace platform, two tools included with the analyzer distribution are spixcounts and spixstats. Spixcounts is a tool that executes the Shade simulation engine and gathers trace information in order to understand register utilization, branch behavior, immediate data values, and the interactions between the most frequently called functions. The spixstats tool reads symbol information from the binary file and formats the raw trace data generated by spixcounts into a usable report. The functions that individually account for at least 0.5% of the program's instruction streams are reported by spixstats as the major functions. The report contains the following information:

- number of invocations of each SPARC opcode including percentage contribution to the overall program
- detailed branch analysis including taken/not taken and direction percentages

- delay slot utilization
- register accesses
- addressing modes used
- immediate field usage (including displacement where applicable)
- percent of opcodes executed in the significant functions
- function caller/callee relationships.

A limitation to the spixstats results is the tracing of program flow through subroutines. Although this analyzer captures the total instructions sorted by functions and caller/callee information, it can not selectively trace through a sequence of sub-routine calls; thus, it is difficult to accurately capture results for routines that rely on shared sub-routines. Unfortunately, each call to a routine can result in different behavior, due to the arguments decided at the time of invocation. Therefore, there is no way to use the spixstats and spixcounts tools to completely isolate the behavior of each MP3 encoding stage.

4.1.2 Custom Shade Analyzer

For the purposes of this research, the author created a new analysis function to capture the load/store characteristics of the entire program and then for that of specific functions. This load/store frequency application (*lsfreq*) operates similar to the spixcounts/Shade flow, but in a more selective manner. The custom analysis engine utilizes the flexibility of the Shade infrastructure to select a subset of the

opcodes in the dynamic trace, specifically the load and store instructions, and then limits the address range so that specific functions can be isolated. By limiting the number of instructions captured in the trace, analysis can focus on specific regions and the simulation time is greatly reduced. The *lsfreq* utility generates a memory bandwidth report by counting the number of load and store operations and accumulating a byte count according to the width of the data transferred by the opcode. Although this result is perhaps more dependent on the quality of the compiler rather than the true bit-requirement of the algorithm, it serves the purpose of analyzing a typical workload with standard levels of optimization.

4.2 Analysis of MP3 Compression

To study the LAME application, the C-language source files were first compiled into a single binary executable program with the gcc compiler. An UltraSparc general purpose processor served as the host machine for the media processing analysis contained in this report. Shade trace and analysis engines simulated the execution of the MP3 encoder application and collected pertinent information related to the dynamic opcode and function utilization. For a more complete evaluation of how the encoder responds to input file characteristics and compression level, five different audio source files were each encoded to final MP3 data rates of 128 Kbps and 320 Kbps.

Discrete analysis of the major MP3 encoding stages was achieved by capturing the characteristics of several C-functions that compose each stage. For this study, the compiled LAME MP3 encoder application was first analyzed with the spixcounts and spixstats functions and then program was simulated with the *lsfreq* utility. Due to the excessive quantity of instructions and function calls required to process this non-trivial algorithm, the profiles are reported as a percentage rather than absolute number of function calls and dynamic instructions executed. In the interest of reducing instruction-tracing complexity, only the top several functions of each encoding stage are captured by the tracing tools. The actual quantity of functions that are reported for each stage varies according to the final dynamic trace results.

The major functions reported by Shade were then sorted into one of the seven stages of encoding as described in section 3.3. An eighth category, "Miscellaneous", contains several functions that require noticeable processing resources, but are either shared across multiple stages or serve as auxiliary functions in addition to the major stages. The remaining functions are omitted from consideration due to their relatively low utilization, less than 0.5%. The total instruction content of the omitted functions typically contributed 3% to 5% of the total dynamic instruction stream. Therefore, this analysis contains sufficient information for a comprehensive understanding of the key components of the MP3 encoder application.

4.3 Input-Output Files

Five audio files encoded in the "wav" format were captured from music CDs and used as PCM input for the MP3 encoder algorithm. The analysis included a variety of selections to determine the effect of audio complexity on the encoding process. Table 4.1 describes pertinent information for these files including genre, original size, and MP3 file size when compressed at 128 Kbps and 320 Kbps. The audio files range from a relatively short 40-second track to several minutes of audio. A common classification based on genre roughly describes the content of the music contained in each file.

Table 4.1: Input file information for test cases

General Audio Input Information				File Size (Bytes)		
Filename	**Artist**	**Genre**	**length (sec)**	**original**	**320Kbps**	**128Kbps**
pnp.wav	Dave Matthews Band	Rock	40	7126604	1619591	647835
george.wav	George Winston	Piano	91	31941224	7246366	2898546
antonio.wav	Antonio Vivaldi	Orchestral	215	38096938	8642350	3456939
chem.wav	Chemical Brothers	Alternative	96	32932502	7471019	2988407
ravi.wav	Ravi Shankar	Instrumental	557	98320136	22298121	8919248

5 RESULTS

The process of compressing each input file with the LAME MP3 encoder was traced and analyzed with the Shade simulation engine. The results of the analysis are divided into four major sections:

- routine-level profiling
- instruction-level profiling
- memory access characteristics
- computational workload analysis.

Each of these sections presents aspects of the compression application and describes how file size, compression ratio, and audio complexity affect the execution characteristics of the MP3 encoder and its major stages.

5.1 Routine-Level Profiling

Routine-level profiling determines the quantity of dynamic instructions a routine contributes to the overall instruction stream. In this context, a routine is equivalent one of the LAME functions written in the C programming language. Table 5.1 lists static size of the major MP3 functions grouped by encoding stage. Most of these functions were deemed highly utilized, i.e. reported by spixstats, in every experiment. From this table, it is evident that a majority of the static instructions are found in the psychoacoustic modeling routines. This is no surprise

based on the complexity of this routine and variety of operations performed at this stage.

Table 5.1: MP3 stages and associated functions

***Stage* / Functions**	**Static Size (Bytes)**		***Stage* / Functions**	**Static Size (Bytes)**
Windowing & Polyphase	***6999***		***Huffman Entropy Encoding***	***1418***
window_subband	5961		ix_max	79
fill_buffer	391		count_bit_ESC	271
unpack_read_sample	647		count_bit_noESC_from2	163
MDCT	***3917***		count_bit_noESC_from3	263
mdct_sub48	2095		HuffmanCode	343
mdct_long	1387		choose_table_nonMMX	299
mdct_short	435		***Quantization***	***7817***
FFT	***1541***		quantize_xrpow	367
fht	703		init_xrpow	207
fft_long	415		calc_xmin	1807
fft_short	423		count_bits	1455
Miscellaneous	***3279***		outer_loop	1199
sqrt	139		amp_scalefac_bands	1111
fabs	19		calc_noise	1671
exp	899		***Psychoacoustic Modeling***	***11471***
log	999		l3psycho_anal	11471
lame_encode_buffer_sample_t	1079		***Bitstream Formatting***	***252***
fast_log2	144		putbits2	252

A dynamic analysis of these functions shows how many instructions each stage contributes to the total instruction count of the complete application. Figures 5.1 and 5.2 compare the stages according to their dominance in the dynamic instruction stream. This analysis profiled the results from two of the audio input files pnp.wav and ravi.wav. These samples contain considerably different audio properties, and thus give insight into the effect of acoustic complexity on the

application execution. As depicted by the chart, the majority of the instruction stream is devoted to the quantization, psychoacoustic modeling, and Huffman encoding. Nested in the iterative looping structure, the Huffman and quantization functions often execute several times for each block of PCM samples analyzed. The psychoacoustic processing requires irregular control flow sequences and large table searches resulting in a large demand on the CPU instruction bandwidth.

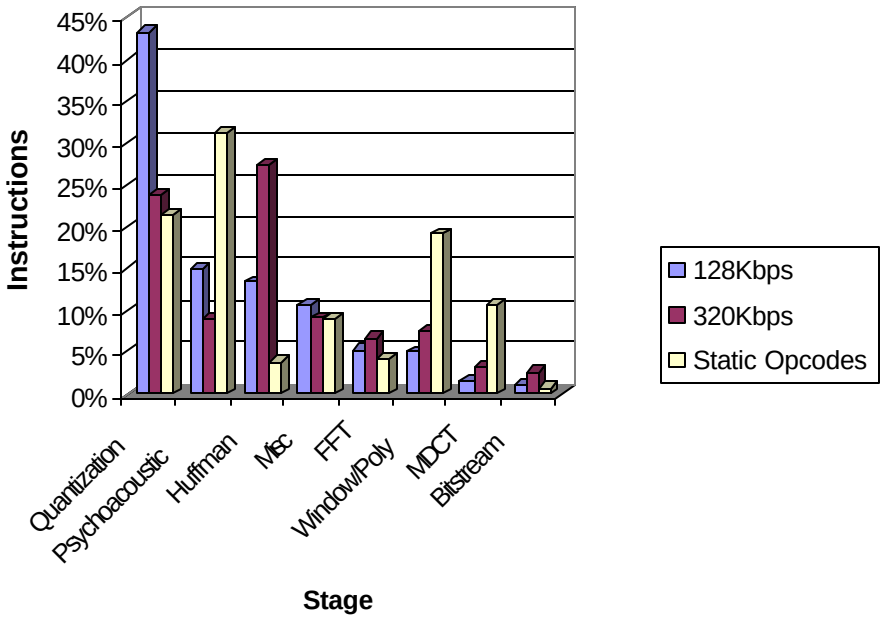

Figure 5.1: Routine profile for pnp.wav

Figure 5.2 depicts the compression of a significantly different audio source. However, the similarities between this and Figure 5.1 show that the dynamic mix or routine execution is not dramatically affected by file size and audio characteristics, particularly at lower bitrates. A noticeable difference can be found in the routine

profiles of 128 Kbps compression and 320 Kbps compression. As the compression ratio is reduced, less emphasis is placed on the quantization stage because acceptable noise levels are easier to achieve, however the Huffman encoding contributes to a higher overall percentage of the instruction trace due to the increased size of the final bitstream.

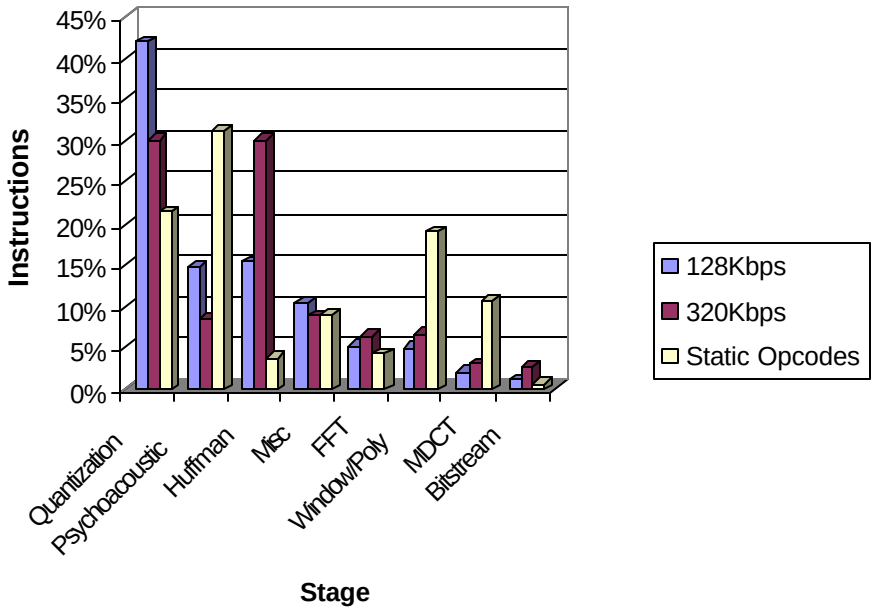

Figure 5.2: Routine profile for ravi.wav

Generated from the average profile of all five input files, Figure 5.3 easily identifies the stages that require significant instruction processing. Clearly, optimization efforts are better spent on the quantization and Huffman encoding stages. The MDCT, FFT, and polyphase filtering signal processing algorithms

contribute to less than 25% of the total instruction stream. It is important to note that these stages are probably not the performance bottleneck in this general-purpose implementation of the MP3 encoder application. Figure 5.4 shows how 320 Kbps compression relies more heavily on these DSP routines, but they are still not a dominant factor in the encoding process.

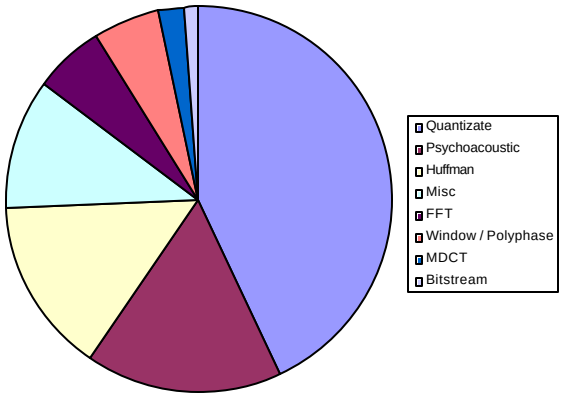

Figure 5.3: Average routine profile for 128 Kbps compression

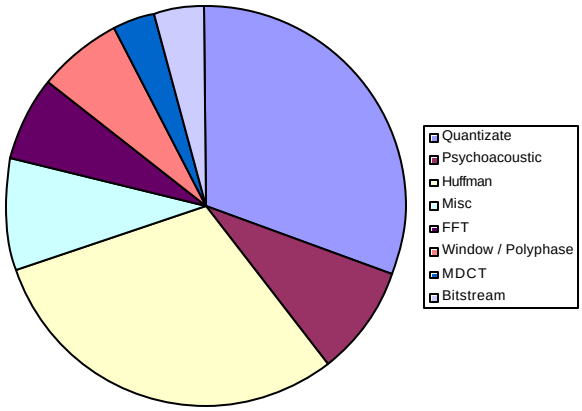

Figure 5.4: Average routine profile for 320 Kbps compression

The chart in Figure 5.5 depicts the MP3 algorithm behavior as bitrate is increased from 128 Kbps to 320 Kbps. Hyen-O Oh, et. al., expect significantly fewer iterations through the quantization loops at high bitrates, and thus a reduction in instructions in the quantization steps [24]. This assumption is confirmed by the quantization category change in the figure. However, an increase in Huffman encoding instructions can be attributed to the larger code words and more extensive searches required to determine optimal entropy coding. Reductions in algorithmic complexity at 320 Kbps are also noted by the reduced amount of psychoacoustic modeling and bitstream instructions executed. In spite of a constant or increased function call rate, the instruction count demonstrates that these stages are simplified by the addition of available bits in the output datastream.

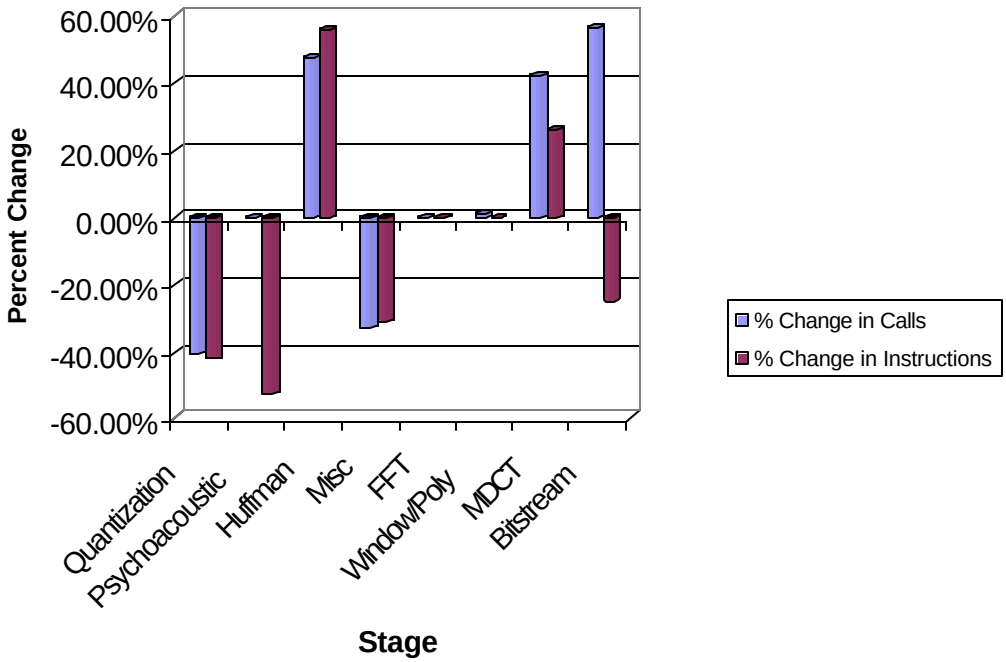

Figure 5.5: Percent change in dynamic functions calls and instruction count from 128 Kbps to 320 Kbps compression rate for ravi.wav

5.2 Instruction-Level Profiling

Instruction-level profiling consists of capturing the dynamic instruction trace for the MP3 compression application and sorting the opcodes into six categories: load, store, integer, floating-point, branch, and other. The "other" category accounts for machine-specific instructions such as save, restore, nop and various maintenance operations. Application-level and stage-level results for the five input files are presented in this section.

Application-level analysis based on traces from the entire program execution proves that the instruction mix was very similar for each of the audio input files. Figure 5.6 contains the instruction class percentages for MP3 encoding at 128 Kbps averaged across all input files. According to this figure, it is evident that ALU operations comprise the majority of the instruction stream at 52%, while the combined load and store memory access instructions contribute to 34% of the trace. The arithmetic instructions (integer and floating point operations) together account for approximately half of the total instruction mix; of that, integer dominates floating point by about 2-to-1. In the category of memory access, the load instructions clearly contribute more than the store instructions. Although media kernels are typically considered to be memory access constrained, the instruction analysis of the complete MP3 encoding application shows that the general purpose instruction stream relies more heavily on the processor's ability to execute ALU instructions.

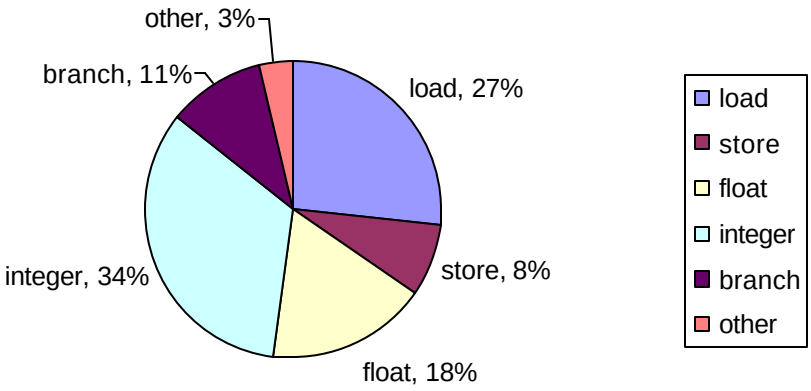

Figure 5.6: Average instruction mix for 128 Kbps

Comparing the results of 128 Kbps and 320 Kbps compression in Figure 5.7 shows that the instruction mix is not heavily dependent on compression ratio or input file characteristics. However, as the compressed bitstream data-rate increases, the branch instructions tend to take a slightly higher percentage of the overall stream with a corresponding decrease in memory instructions.

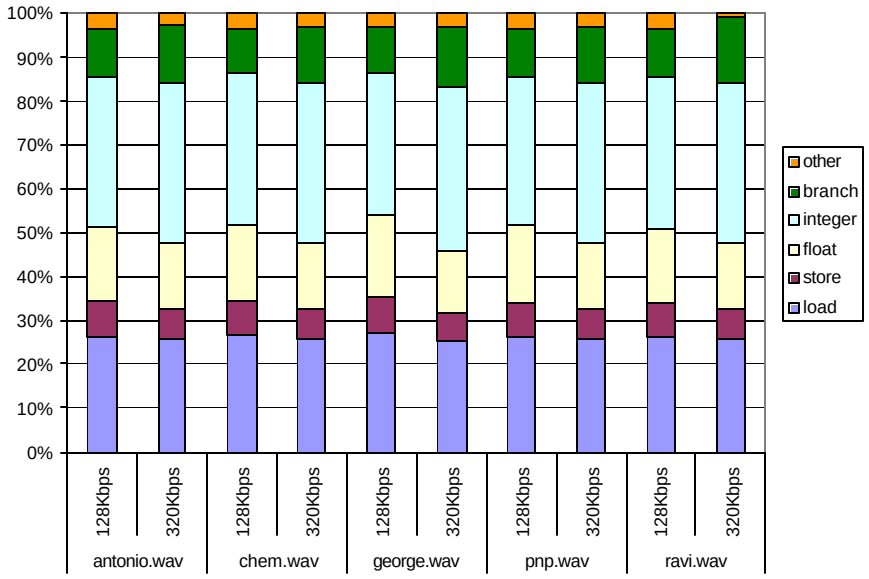

Figure 5.7: Application instruction mix

MP3 encoding stages each contain a different mix of instructions as required by their routines. Figure 5.8 compares the instruction makeup averaged across traces from the five audio input files. Memory accesses are only a dominant part of the window/polyphase filtering and MDCT stages, but remain noticeably present in all stages aside from Huffman encoding and Miscellaneous routines. It is also evident that the DSP stages (the first three in the chart) are not overwhelmed by large amounts of branch operations. Performance of these three DPS stages could be enhanced by a focus on the ALU and memory workload.

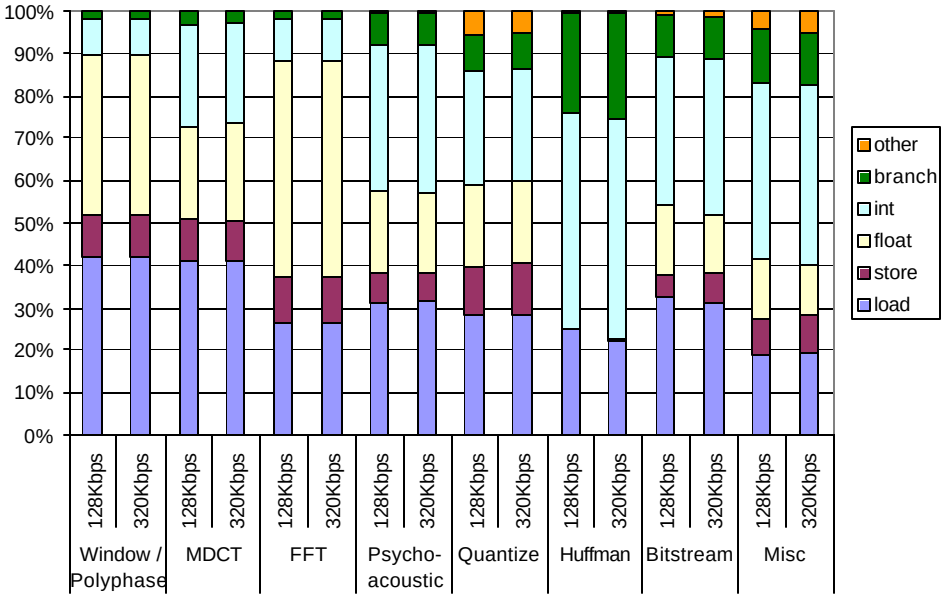

Figure 5.8: Instruction mix for MP3 stages

The Huffman encoding stage is a conspicuously different from typical mixes found in the rest of the MP3 application. In this stage, the processor is heavily reliant on load and branch operations to compute a large amount of integer instructions which are mostly comparisons and address calculations. Because of its high overall percentage of dynamic instructions, as described in section 5.1, specialized hardware designed to enhance the runtime performance of this stage would be beneficial to the MP3 encoding application performance.

5.3 Memory Access Characteristics

As with most multimedia applications, the MP3 encoding algorithm places a significant demand on the memory infrastructure of a general purpose processor. The combination of results from the spixstats and lsfreq analyzers give a clear indication of this requirement. An application-level view of the memory access is first presented, and then results from each stage of the encoding process are reviewed to understand how each contributes to the overall memory access profile of the MP3 encoder.

A common measure of memory workload is the comparison of input data size with the amount of data transferred to and from the memory during the execution of the application. MP3 compression algorithms require appreciably more memory transactions than that which is required to load the input data. For example, the pnp.wav input is contained in a 6.8 Megabyte PCM file. Table 5.1 shows the LAME compression results for 128 Kbps compression rate. In this case, the source reduces to a 632 Kilobyte file. From Table 5.2, it can be determined that the application exchanges 13 Gigabytes of data with the memory hierarchy while processing this input; this is nearly 2000 times the size of the source file. Clearly, the MP3 encoding memory bandwidth is dominated by transactions other than those that load and store the input and output data.

The final column of Table 5.2 contains a common memory access metric that measures the traffic in relation to the size of the original and post-transform

data [15][19]. The total memory traffic volume is divided by the sum of the input and output files to indicate memory access requirements independent of file size. It also gives a measure of memory traffic, typically considered overhead, which can be compared with other application workloads. In the case of pnp.wav, the algorithm averages 1,786 bytes of memory traffic for each byte of input or output data. Routine-level and instruction-level profiling did not show differences according the type of audio input file; however, the spectral complexity of antonio.wav caused noticeably more memory traffic for compression than the other input files. It can also be observed that relaxing the audio compression rate to 320 Kbps reduces the memory transactions per input or output byte.

Table 5.2: Memory access characteristics

Source Information				Memory Access Characteristics			
Compression	Filename	Source Size Mbytes	Output Size Mbytes	Mbytes Loaded	Mbytes Stored	Mbytes Transferred	(L+S) Mbytes / (I + O) Mbytes
128Kbps	pnp.wav	6.8	0.6	10,050	3,189	13,238	1786
	george.wav	30.5	2.8	39,709	12,958	52,667	1585
	antonio.wav	30.5	3.3	51,995	16,511	68,506	2029
	chem.wav	31.4	2.8	42,070	13,358	55,428	1618
	ravi.wav	93.8	8.5	138,030	43,669	181,698	1777
320Kbps	pnp.wav	6.8	1.5	7,340	2,122	9,462	1134
	george.wav	30.5	6.9	36,573	10,286	46,858	1254
	antonio.wav	30.5	8.2	40,620	11,661	52,281	1351
	chem.wav	31.4	7.1	36,036	10,567	46,603	1209
	ravi.wav	93.8	21.3	104,783	29,857	134,641	1170

The next memory access analysis compares the demands of the major MP3 encoding stages. Figure 5.9 shows how load and store instructions in each group contribute to the overall memory bandwidth required by the MP3 algorithm for the ravi.wav input. The other audio input files are omitted from this analysis because each exhibits very similar stage-level trends. When combined with the data in Figure 5.2, it is evident that functions with a higher percentage of dynamic instructions typically have higher memory bandwidth requirements. A key exception is the bitstream stage which executes more memory transactions per instruction than some of the other stages. This stage is relatively insignificant on Figure 5.2, but among the top contributors in the memory traffic chart.

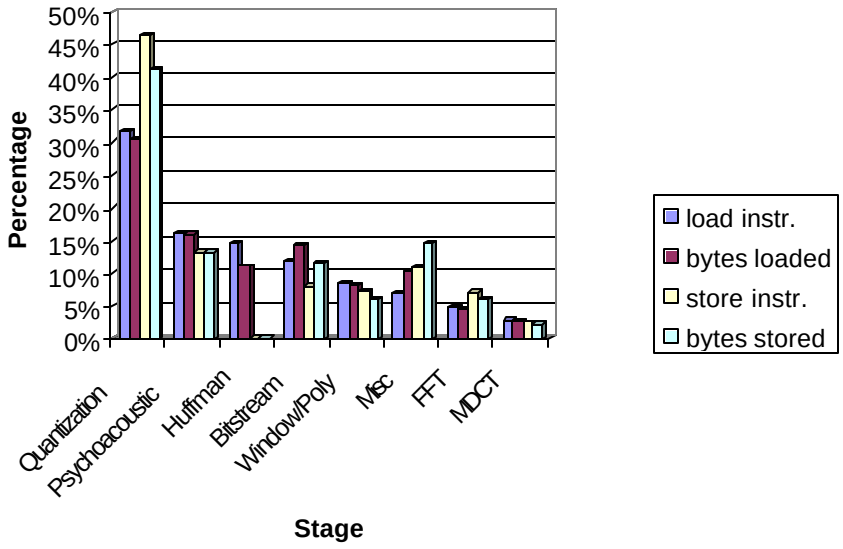

Figure 5.9: Percentage of memory instructions and memory traffic for ravi.wav encoded at 128 Kbps

The data in Figure 5.9 also gives an indication of the efficiency of the load and store instructions. It is evident that a large quantity of instructions is required to transfer data to and from the central processor. Some stages take advantage of load and store instructions that transfer larger amounts of data per instruction. The bitstream and miscellaneous categories demonstrate higher efficiency due to the fact they transfer a higher percent of data than percent of instructions executed.

When compared to the behavior of the encoder at 128 Kbps, the data in Figure 5.10 depicts a noticeable rearrangement of functions when sorted from highest to lowest memory traffic for the 320 Kbps compression rate. Here, the Huffman coding accounts for a more significant portion of the overall load/store instructions. The Psychoacoustic routines conversely dropped in rank. This difference can be attributed to a relaxation on the complexity of calculations for signal-to-mask ratio which result in a reduced amount of instructions, as documented in Figure 5.5.

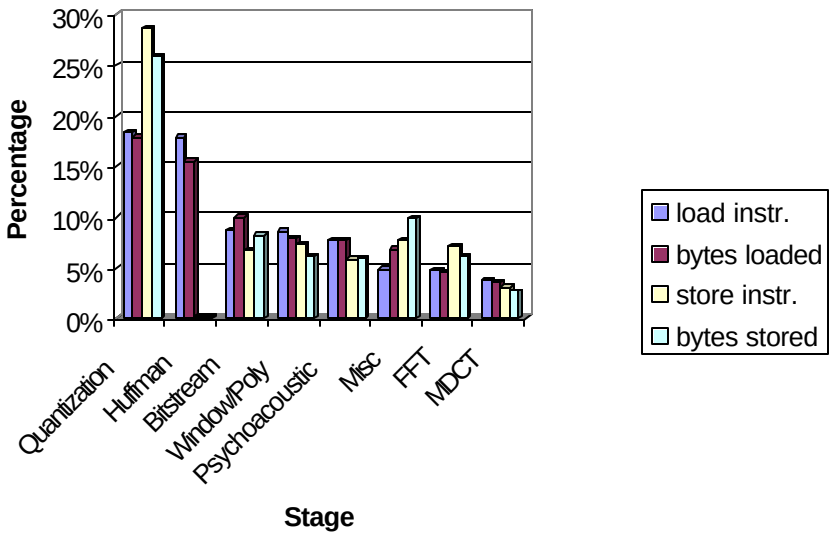

Figure 5.10: Percentage of memory instructions and memory traffic for ravi.wav encoded at 320 Kbps

The percentage of store instructions contributed by the quantization routines emphasizes the fact that this stage accounts for a major portion of the data created by the compression application. Although the quantization stage accounts for a less significant portion of load instructions and bytes, the absolute quantity of load instructions and loaded bytes exceeds that of store instructions and stored bytes.

A more significant exception to the symmetry of load and store utilization is the Huffman coding routines. This function his highly table-intensive, and therefore requires mostly load instructions. At 320 Kbps, this disparity is even larger as the algorithm conducts more extensive searches through the tables. Conversely, the quantization steps are more reliant on balanced computations, and thus require fewer load transactions for each store instruction.

Figure 5.11 compares the average memory throughput for each of the major categories. Several MP3 stages sustain more than 1.5 bytes of memory bandwidth for each dynamic instruction in that function. However, the Huffman stage again provides an exception with its low data throughput of less than one byte per instruction for both compression ratios. As evident in this chart, the architecture studied in this report has an apparent limit of two bytes of data throughput per instruction. A uniquely designed memory architecture and address co-processor could affect a dramatic change in this mix by alleviating the overhead memory access instructions.

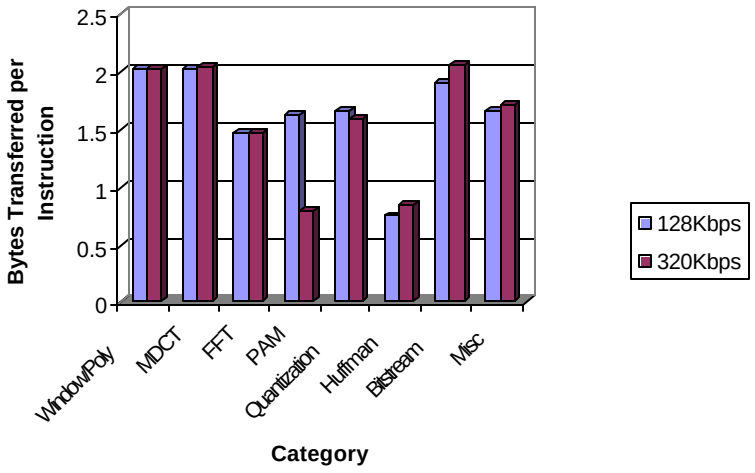

Figure 5.11: Bytes transferred per instruction for ravi.wav

5.4 Computational Workload

The final analysis consists of a measurement of the computational workload of the MP3 encoding algorithm. Figure 5.12 evaluates the ratio of ALU instructions to the memory and branch instructions present in the dynamic instruction stream. In every stage of the application, ALU instructions exceed both memory and branch instructions. It is likely that an optimizing compiler can affect the ratio for the branch comparison by loop unrolling and other optimizations, but the memory traffic is generally fixed by the demands of the algorithm.

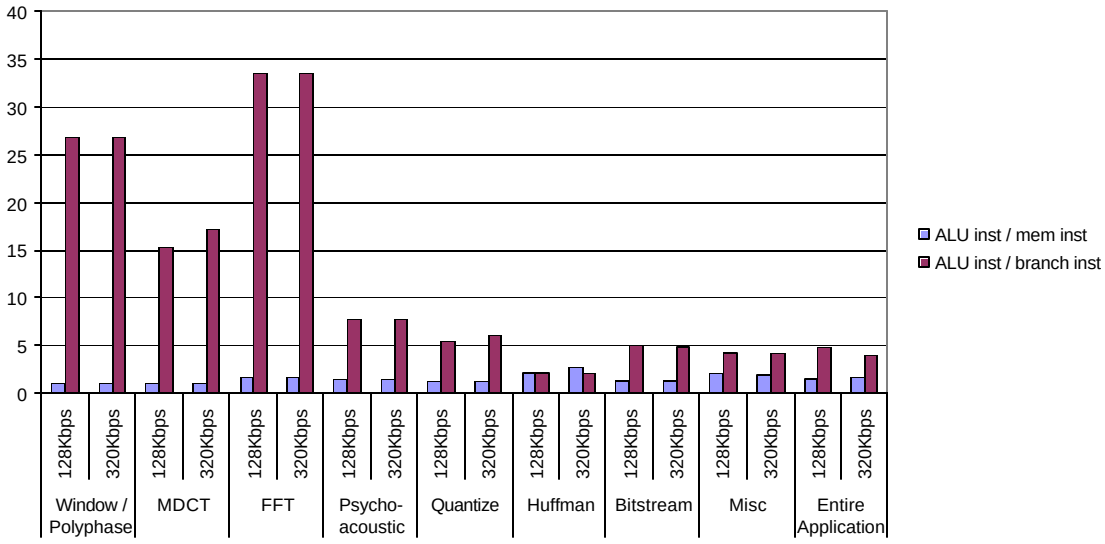

Figure 5.12: Ratio of ALU instructions to memory and branch instructions

ALU operations effectively dominate the machine workload for the Signal processing stages, thus providing a high level of computations for each branch

instruction. Although it would seem that enhancing or adding arithmetic units could enhance the MP3 algorithm, the low ratio of ALU to memory instructions in these and other stages caution that architectural changes that do not provide a comparably scaled memory infrastructure could lead to lower than expected performance gains. Stages aside from those related to signal processing show a much higher dependence on branch instructions. In order to maximize utilization of the computational units during these stages, hardware must effectively handle the relatively frequent changes in control flow.

6 CONCLUSIONS

This report analyzed the significant characteristics of an application that compresses PCM audio data into the MP3 format. From the results of detailed simulation, it can be determined that the architecture of a general-purpose machine cannot handle the large table lookups and significant intermediate data structures without an excessive load on its memory architecture. The MP3 encoder instruction and routine profile must be carefully considered before making architectural enhancements to increase performance. It is likely that the register set and local memory hierarchy are not well suited to this application. However, it is evident in some cases that the memory access instructions do not dominate the dynamic instruction stream.

It can also be noted that several aspects of the dynamic instruction mix vary for each of the major MP3 encoder stages and this instruction stream is sensitive to the requested compression ratio. For example, the psychoacoustic calculations of this compression algorithm are dependent on both the input data and quality expectations of the result. A custom-designed multimedia architecture must accommodate the possibility that the true performance bottleneck might be different for each invocation of the program.

The properties of a general purpose processor give it the ability to handle multimedia workloads, but it is not the most optimal architecture for the task. As an alternative to a general purpose machine, dedicated hardware could improve the execution performance on this multimedia application. A key area to consider is the vast amounts of data transferred to and from the processor core. A potential solution might address this demand by placing several direct-access "cache" memories near the processor to reduce demand on the external memory resources. These memories could contain the tables required for the key kernel operations: quantization, Huffman coding, psychoacoustic modeling, filtering, MDCT, FFT, and the intermediate data required for each stage of the compression routine. Processor enhancements that improve the performance of highly runtime-dominant stages such as Huffman coding, quantization, and psychoacoustic modeling would significantly impact overall application performance.

References

[1] E. Ambikairajah, A. G. Wong, W. T. K. "Auditory masking and MPEG-1 audio compression". *Electronics & Communication Engineering Journal*, Volume 9, Issue 4, pages 165-175, August 1997.

[2] J. N. Barkdull and S. C. Douglas. "General-purpose microprocessor performance for DSP applications". *Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers*, Pages 912-916, November 1996.

[3] K. Brandenburg and H. Popp. "An Introduction to MPEG Layer-3". *EBU Technical Review*, June 2000.

[4] Tian-Sheuan Chang, Chein-Wei Jen. "Embedded memory module design for video signal processing". *IEEE Signal Processing Society VLSI Signal Processing*, Pages 501-510, September 1995.

[5] Bob Cmelik and David Keppel. "Shade: A Fast Instruction-Set Simulator for Execution Profiling". *ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1994.

[6] Grainger David. "MP3s Are Big Music's Savior, Not Slayer". *Fortune*, Volume 146, Issue 5, September 2002.

[**7**] K. Diefendorff, and P. K. Dubey. "How multimedia workloads will change processor design". *Computer*, Volume 30, Issue 9, Pages 43-45, September 1997.

[8] Fraunhofer IIS. http://www.iis.fraunhofer.de/amm/techinf/layer3/

[9] B. Furht, R. Westwater, J. Ice. "Multimedia broadcasting over the Internet. I". *IEEE Multimedia*, Volume 5, Issue 4, Pages 78-82, December 1998.

[10] C. H. Gebotys and R. J. Gebotys. "Performance-power optimization of memory components for complex embedded systems". *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, Volume 5, Pages 152-159, January 1997.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown. “Mibench: a free, commercially representative embedded benchmark suite”. *IEEE International Workshop on Workload Characterization*, Pages 3-14, December 2001.

[12] M. Hans and R. W. Schafer. “Lossless compression of digital audio”. IEEE *Signal Processing Magazine*, Volume 18, Issue 4, Pages 21-32, July 2001.

[13] C. N. Hinds. “An enhanced floating point coprocessor for embedded signal processing and graphics applications**”**. *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, Volume 1, Pages 147-151, October 1999.

[14] ISO/IEC 11172-3 Compression of Audio standard

[15] L. K. John, V. Reddy, P. T. Hulina, L. D. Coraor. “Program balance and its impact on high performance RISC architectures”. *First IEEE Symposium on High-Performance Computer Architecture*, pages 370-379, January 1995.

[16] B. Kapoor. “Analyzing memory bandwidth requirements of video algorithms”. *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, Volume 4, pages 170-173, June 1998.

[17] N. Kavvadias and S. Nikolaidis. “Parametric architecture for implementing multimedia algorithms”. *14th International Conference on Digital Signal Processing*, Volume 2, Pages 1261-1264, July 2002.

[18] LAME Ain't an Mp3 Encoder. http://lame.sourceforge.net/

[19] Byeong Kil Lee and L. K. John. “Implications of programmable general purpose processors for compression/encryption applications”. *The IEEE International Conference on Application-Specific Systems, Architectures and Processors*, Pages 233-242, July 2002.

[20] A. Mehis and R. Radhakrishnan. “Optimizing applications for performance on the Pentium 4 architecture”. *IEEE International Workshop on Workload Characterization*, Pages 59-67, November 2002.

[21] Joan L. Mitchell. *MPEG Video: Compression Standard*. New York Kluwer Academic Publishers, 2002.

[22] P. Moravie, H. Essafi, C. Lambert-Nebout, J. L. Basille. "Real-time image compression using SIMD architectures". *Computer Architectures for Machine Perception*, Pages 274-279, September 1995.

[23] P. Noll. "MPEG digital audio coding". IEEE Signal Processing Magazine, Volume 14, Issue 5, pages 59-81, September 1997.

[24] Hyen-O Oh, Joon-Seok Kim, Chang-Jun Song, Young-Cheol Park, Dae-Hee Youn. "Low power MPEG/audio encoders using simplified psychoacoustic model and fast bit allocation". *IEEE Transactions on Consumer Electronics*, Volume 47, Issue 3, Pages 613-621, August 2001.

[25] D. Pan. "A tutorial on MPEG/audio compression". IEEE Multimedia, Volume 2, Issue 2, Pages 60-74, 1995.

[26] M. J. Serrano and Youfeng Wu. "Memory performance analysis of SPEC2000C for the intel itanium processor". *IEEE International Workshop on Workload Characterization*, Pages 184-192, December 2001.

[27] P. Singh, W. Moreno, N. Ranganathan, H. Neinhaus, "A flexible MPEG audio decoder layer III chip architecture**"**. *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, Volume 4, Pages 37-40, June 1998.

[28] N. T. Slingerland and A. J. Smith. "Cache Performance for Multimedia Applications". *International Conference on Supercomputing*, Pages 204-207, 2001.

[29] S. Sohoni, Rui Min, Zhiyong Xu, Yiming Hu. "A Study of Memory System Performance of Multimedia Applications". *Joint International Conference on Measurement and Modeling of Computer Systems*, Pages 206-215, 2001.

[30] D. Talla, and L. K. John. "Cost-effective hardware acceleration of multimedia applications". *International Conference on Computer Design*, Pages 415-424, September 2001.

[31] D. Talla, L. K. John, V. Lapinskii, B. L. Evans. "Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures". *International Conference on Computer Design*, Pages 163-172, Sept. 2000.

[32] D. Talla and L. K. John. "Execution characteristics of multimedia applications on a Pentium II processor". *Conference Proceeding of the IEEE*

International Performance, Computing, and Communications Conference, Pages 516-524, February 2000.

[33] J. Tyler, J. Lent, A. Mather, Huy Nguyen. "AltiVecTM: bringing vector technology to the PowerPC^{TM} processor family". *IEEE International Performance, Computing and Communications Conference*, Pages 437-444, February 1999.

[34] Jerry C. Whitaker and K. Blair Benson. *Compression Technologies for Video and Audio*. New York McGraw-Hill Professional, 2000.

[35] S. Wong, S. Cotofana, S. Vassiliadis. "General-purpose processor Huffman encoding extension". *International Conference on Information Technology: Coding and Computing*, Pages 158-163, March 2000.

Vita

Michael Lance Karm was born in Dallas, Texas on June 6, 1976 to Richard Gilbert Karm and Jean Ann Karm. After graduating from Lake Highlands High School in June 1994, he entered The University of Texas at Austin in August 1994. In December, 1998 he received the degree of Bachelor of Science from the College of Engineering and entered The Graduate School at the University of Texas at Austin in January 1999. He has held internship positions at Texas Instruments, Raytheon, and International Business Machines, and a full-time position at Wavefly Corporation. He is currently employed by Avnet, Inc.

Permanent Address: 512 South Lynnwood Trail
Cedar Park, TX 78613

This report was typed by the author.