Study of Multimedia Application Characteristics

Navneet Aron, Honggo Wijaya, Arjun Singh, Varun Malhotra

April 17, 2003

Abstract

In this document, we study the characteristics of various multimedia applications in an attempt to figure out the optimal architectural support for these applications. `CJPEG`, which is used for image compression and decompression and is useful in a variety of video applications, is analyzed at the programming language level to identify the scope of exploiting data level and loop-level parallelism in image applications.

1 Brief Overview of Multimedia Applications

The main characteristics of Multimedia applications are as follows:

- Intensive computation for highly regular operations
- Intensive I/O or memory accesses, and data locality
- High control complexity in less computational intensive tasks
- Frequent encounters of small integer operands
- Usually demand real-time processing capabilities

Some of the common multimedia applications include Video conferencing, video on demand. We primarily focus on video/image-synthesis applications rather than speech applications which also constitute multimedia.

2 General Characteristics of Multimedia applications

Fritts et al. [1] did an exhaustive study of multimedia applications which primarily included video and image systems. Although they didn't include some of the latest *graphics* applications in their benchmarks but it is a pretty exhaustive study of video/image systems.

2.1 Benchmarks

Fritts et al. [1] used an enhanced version of *MediaBench*[3] benchmarks which they called *MediaBench +*. The *MediaBench+* benchmark includes:

- Video: MPEG-2, *MPEG-4*, *H.263*
- Audio: ADPCM coder

- Graphics: Mesa[1]
- Image: JPEG, EPIC, Ghostscript
- Security: PGP, Pegwit
- Speech: GSM, G.271, Rasta

Very recently though, need for including new applications has been felt so that we can include a representative set of current multimedia applications. An enhanced version of MediaBench called *MediaBench-II*[4] has been proposed, however, we are not aware of any studies of multimedia applications using this enhanced benchmark. In addition to this, there has been talk of splitting the *MediaBench* benchmark into different benchmarks for area specific-applications like Video, audio which can have different characteristics.

2.2 Instruction Mix

Multimedia applications (as shown in figure 1) have a lot of integer ALU operations as compared to scientific codes and other integer programs. The combination of integer and floating point operation is about the same as the one in scientific codes showing that multimedia applications are as computational intensive as scientific applications. Moreover, the operands in these integer ALU instructions are 16 bit operands for more than 75% of the instructions Even more, most of the integer operands which re-

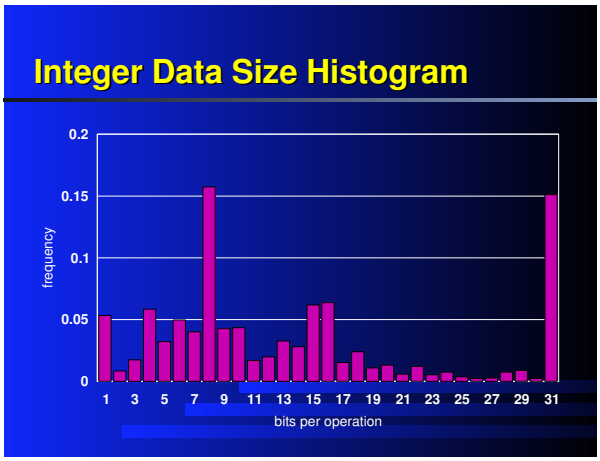


Figure 1: *This data is taken from [1]*

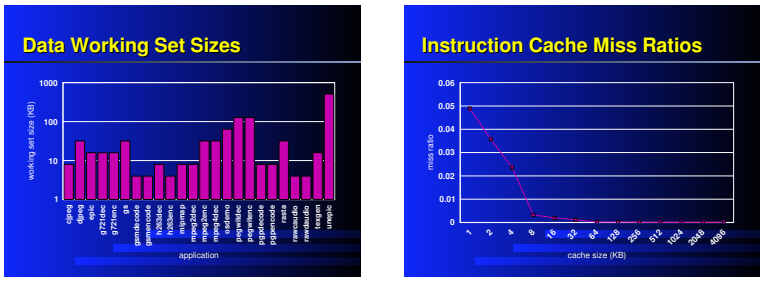
quire more than 16 bits are concentrated near the 32 bit zone and there occur primarily due to pointer-based references. It is important to note here that this trend is true for video/image appliactions but not all the multimedia applications in general, graphics applications being an exception. For instance, the graphics applications have lots of floating point operations than video/image applications. The average basic block size of multimedia applications is about 8 which indicates the frequency of branch intructions in the codes of multimedia applications. Primarily

[1]Comment: Mesa is not representative of today's graphics applications

most of the branches are due to loops which makes branch prediction easier. As a consequence of this, static branch prediction performs nearly as good as dynamic branch prediction [1][2] which indicates that we don't need complex logic in hardware for predicting branches if the intended applications are video/image applications.

2.3 Memory Characteristics and Cache Behavior

Multimedia applications exhibit lower cache miss rates .The average working set for Data is 32KB while for Instructions is 8KB. Since access to data is sequential, prefetch buffers can be used. There is small temporal locality due to different code segments operating on same data. But in most cases, caching could be avoided by explicit message passing between such code segments.[5]



(a) (b)

Figure 2: This data has been taken from [1]

2.4 Parallelism

There is enormous amounts of data-level and loop-level parallelism in multimedia applications. However, there is not huge amounts of ILP in these applications as is evident from the fact that the average basic block size is 8 (as compared to basic block size of applications in SPECInt benchmark where the average block size is 5.5). On similar note, take the example of MPEG-2. In a study [6], the MPEG-2 coding and decoding algorithm is run on an 8-way CMP exploiting TLP and also a 6 way issue superscalar processor (SP6) exploiting ILP. The CMP performs $50\% - 130\%$ better than the SP6 on both the coding and decoding. Moreover, the TLP implementations are extremely straight forward and do not require the programmer to have any knowledge of the underlying machine architecture beyond the idea of multiple thread support.

2.5 Summary of functioning of Video applications

Most video playback applications require computing reverse discrete cosine transform (RDCT), involving integer additions and multiplications and a CPU intensive operation called dithering. *Dithering* is the process of approximating colors you don't have by mixing colors you do have to

improve picture quality. Most of the operations (or rather *hot-spots*) used in these applications use $O(n^2)$ steps where n is the size of the input.

2.6 Existing Techniques

Before analysing specific applications, let us consider the various hardware software techniques currently used to optimise performance for multimedia applications.

Usually, function-specific ASICs dedicated to particular Media application subtasks are used. Although they can typically achieve highest performance levels at minimum cost for silicon area and optimized power consumption but they lack the flexibility to accommodate to algorithm modifications, and the large design effort justifies a dedicated implementation only for high-volume consumer market products of fixed functionality (embedded market).

- SIMD (Single Instruction Stream, Multiple Data Streams) architectures are characterized by several data paths executing the same operation on different data entities in parallel. While a high degree of parallelism can be achieved with little control overhead, data path utilization rapidly decreases for scalar program parts. In general, pure SIMD architectures are not an efficient solution for complex multimedia applications; they are best suited for algorithms with highly regular computation patterns.
- Architectures featuring a split-ALU are based on a principle similar to SIMD: a number of lower-precision data items are processed in parallel on the same ALU. The advantage of this approach is its small incremental hardware cost provided a wide ALU is already available. However, the exploitable data parallelism decreases where processing with higher precision/word-length is required.
- Instruction level parallelism in multimedia applications is targeted by VLIW (Very Long Instruction Word) architectures, which specify several operations within a single long instruction word to be executed concurrently on multiple function units. VLIW processors have to rely on static instruction scheduling performed at compile-time and require no hardware support for dynamic code reordering. VLIW might not perform really well for multimedia applications due to small amount of ILP in multimedia applications.
- Task level as well as data level parallelism can be exploited by MIMD (Multiple Instruction Streams, Multiple Data Streams) architectures, which are characterized by a number of parallel data paths featuring individual control units. Thus, MIMD processors offer the highest flexibility for algorithms to be executed in parallel. However, they incur high hardware cost for multiple control units as well as for a memory system delivering sufficient bandwidth to supply all required instruction streams.
- On the other hand, in the programmable processor world techniques like introduction of specialized instructions for frequently recurring operations of higher complexity, e.g., a multiply-accumulate operation with saturation. This technique usually significantly reduces the instruction count, resulting in faster program execution. The decision about which instructions to implement depends on the probability of their use.

2.7 Interesting facts

Most multimedia applications apply block-partitioning and work on small units of data which fit into the cache. Within these block there is significant data reuse as well as spatial locality. Large number of references are to the internal data-structures, which are relatively small and can fit into cache.

2.8 Experiments with Image compression-decompression tools

DJPEG was run on a sample file (100KB) for 100 runs and the data of profiling experiments is as follows:

Function name	Filename	%Time spent
jpeg_idct_islow	jidtint.c	42.86
h2v2_fancy_upsample	jdsample.c	18.80
ycc_rgb_convert	jdcolor.c	18.80
decode_mcu	jdhuff.c	12.03
decompress_onepass	jdcoefct.c	3.01
jpeg_fill_bit_buffer	jdhuff.c	1.50
sep_upsample	jdsample.c	1.50

2.9 Internals of DJPEG, *jpeg-to-ppm* converter

In this section, we analyze some of the *hot* methods in *DJPEG* as shown by the profiling data.

void jpeg_idct_islow()
/ This routine performs dequantization and inverse DCT on one block of coefficients. */*
There is a *for* loop which iterates DCTSIZE number of times. For the experiments, this variable was set to the value of 8. Most of the memory accesses are array accesses, which are highly regular. Moreover, there are no dependencies between independent iterations of this loop providing scope for exploiting *loop-level* parallelism.

void h2v2_upsample()
/ Upsampling: Fast processing for the common case of 2:1 horizontal and 2:1 vertical.*/*
This procedure copies the input array into registers (local variables stored in registers) does some operations on that array in a loop and copies it back into the output array.

void ycc_rgb_convert()
/ Converts some rows of samples to the output colorspace. */*
This function too has huge amounts of *thread-level-parallelism* with a *for* loop nested inside a *while* loop and highly regular array accesses.

void start_pass_huff_decoder () */* This method does some initialization for a Huffman-compressed scan.*/*
In this method, we have the same trend with lots of *loop-level* parallelism. However, inside the loop body, there are lot of conditional checks, henece there are lot of branch instruction in the code of a single iteration.

References

[1] Understanding multimedia application characteristics for designing programmable media processors, *J. Fritts*, *W. Wolf*, and *B. Liu*, *SPIE Photonics West, Media Processors '99*, San Jose, CA, pp. 2−13, Jan. 1999

[2] Evaluation of Static and Dynamic Scheduling for Media processors, *J. Fritts*, and *W. Wolf*, *2nd Workshop on Media Processors and DSPs (held in conjunction with MICRO-33)*, Monterey, CA, pp. 34 − 43, Dec. 2000

[3] MediaBench: A Tool for Evaluating and Synthesising Multimedia and Communication Systems, *C. Lee*, *M. Potkonjak*, and *W. H. Mangione-Smith*, MICRO-30, 1997

[4] MediaBench II - Technology, Status, and Cooperation, *J. Fritts*, and *W. H. Mangione-Smith*, 4^{th} Workshop on Media and Stream Processors (MSP-4, held in conjunction with MICRO-35), Istanbul, Turkey, Nov. 2002

[5] MediaBench II - Technology, Status, and Cooperation, *S. Sohoni*, *Z. Xu*, *R. Min*, and *Y. Hu*, ACM SIGMETRICS 2001

[6] Exploiting Coarse-Grain Parallelism in the MPEG-2 Algorithm, *E. Iwata*, and *K. Olukotun*, Stanford University CSL Technical Report CSL-TR-98-771, September 1998