Technical Report : LCA-TR-020315

Implications of Programmable General Purpose Processors for Compression/Encryption Applications [*]

BYEONG KIL LEE AND LIZY KURIAN JOHN

Laboratory for Computer Architecture
Department of Electrical and Computer Engineering
The University of Texas at Austin
{blee, ljohn}@ece.utexas.edu

Abstract

With the growth of the Internet and mobile communication industry, multimedia applications form a dominant computer workload. Media workloads are typically executed on Application Specific Integrated Circuits (ASICs), application specific processors (ASPs) or general purpose processors (GPPs). GPPs are flexible and allow changes in the applications and algorithms better than ASICs and ASPs. However, executing these applications on GPPs is done at a high cost. In this paper, we analyze media compression/decompression algorithms from the perspective of the overhead of executing them on a programmable general purpose processor versus ASPs. We choose nine encode/decode programs from audio, image/video and encryption applications. The instruction mix, memory access and parallelism aspects during the execution of these programs are analyzed. Memory access latency is observed to be the main factor influencing the execution time on general purpose processors. Most of these compression/decompression algorithms involve processing the data through execution phases (e.g. quantization, encoding, etc) and temporary results are stored and retrieved between these phases. A metric called overhead memory-access bandwidth per input/output byte is defined to characterize the temporary memory activity of each application. We observe that more than 90% of the memory accesses made by these programs are temporary data stores and loads arising from the general purpose nature of the execution platform. We also study the data parallelism in these applications, indicating the ability of instruction level and data level parallel processors to exploit the parallelism in these applications. The parallelism ranges from 6 to 529 in encode processes and 18 to 558 in decode processes.

[*]This work is supported by the Defense Advanced Research Projects Agency under contract F33615-01-C-1892, NSF grants EIA 9807112 and ECS 0113105, a State of Texas Advanced Technology Program grant, and by Tivoli, IBM, Motorola, Intel and Microsoft Corporations.

1. Introduction

Multimedia applications are increasingly becoming a dominant workload in modern computing as the Internet and mobile communication technologies are growing. These multimedia workloads are characterized by structured and regular computations on large data sets with small element-widths [1]. Current industry support for multimedia application appears in multiple forms: ASICs, application-specific processors (ASPs), multimedia extensions to general-purpose processors, and multimedia co-processors [2]. ASICs while highly efficient for particular tasks, are not flexible enough to support the frequent changes and evolutions in the media marketplace. Application-specific processors offer low-cost alternatives for specific applications, however, again there is a tradeoff involved with flexibility. General purpose processors are flexible, however, necessitates use of software for the entire media application functionality, requiring a large number of instructions. They can however obtain considerable performance improvement with architectural extensions such as Intel's MMX/SSE, Sun Microsystems' VIS, AMD's 3DNow!, Motolora's AltiVec, DEC's MVI, HP's MAX2, and MIPS' MDMX [3][4].

There have been analysis of media workloads on general purpose processors to identify their major features [1][2][5][6][7][8][9][10]. It has been identified that these workloads have significant fine/coarse-grained data parallelism, high instruction-reference locality, high memory/network bandwidth, continuous-media data types, and extensive data reorganization. Many of the applications also need real-time response. In contrast to traditional applications characterized by the scalar processing of 32-bit integer data types, multimedia applications deal with vectors of packed 8-, 16-, and 32-bit integer and floating-point numbers.

While characteristics of these applications on general purpose processors have been studied, no previous research analyzes these workloads from the perspective of execution efficiency on general purpose processors versus application specific environments. There have been research on special purpose chips such as those in [11][12]. In spite of the flexibility offered by software paradigms we realize that there is a significant overhead when these applications are executed on programmable processors. Quantitative characterization of this overhead will provide quantitative indication on the importance of ASPs and ASICs for these workloads. It will also be interesting for understanding the tradeoffs involved, for optimizing the implementations, and for helping hardware software codesign.

The rest of the paper is organized as follows. Section 2 provides overview of encode/decode applications and the experiment methodology. Section 3 presents encode/decode application's characteristics, especially focused on the amount of memory accesses for processing these applications and the parallelism in them. An important contribution is the concept of overhead memory accesses, which denote the temporary memory accesses required while processing these applications, in addition to the real memory accesses for accessing actual inputs and outputs. Section 4 is devoted to evaluate the robustness of overhead memory-access factor as a metric. We conclude the paper in section 5.

2. Experimental Methodology

2.1. Benchmarks/Applications

The commonly used benchmarks for DSP (Digital Signal Processing) and multimedia applications can be classified into kernel-type benchmarks and full application benchmarks. Kernel-type benchmarks (FIR, IIR, Dot product, FFT, etc.) are small programs extracted from real programs and often represent a significant portion of the execution time of the original application. Application benchmarks are full applications performing a particular task with real input data sets. Full application benchmarks are normally preferred to kernel benchmarks because they perform the actual task of interest in the same manner as the real workload [10]. Even if several kernels are used in a full application, the characteristics of each of those kernels cannot accurately tell the features of the entire application.

Previously proposed media benchmarks [9][10] consist mainly of encode/decode applications of audio(sound) and video(image) signals. Many security applications such as fingerprint recognition and speech recognition are also based on encode/decode techniques. Similar to overall multimedia application's features, these encode/decode applications require real-time processing, and they also have the characteristics of large I/O bandwidth, inherent data parallelism and processing regularity. We can say that these encode/decode applications are an important component of modern multimedia applications.

In this paper, we consider a suite of applications from the audio, video and encryption domains. Table 1 shows the selected multimedia encode/decode applications, which we name as the *MediaZip* suite. This suite is composed of 9 full applications including 3 audio applications (ADPCM, G.721, GSM), 3 video/image applications(JPEG, MPEG, EPIC) and 3 encryption applications(PEGWIT, Blowfish, Rijndael). The first 7 applications come from the MediaBench suite [9] and the last 2 applications come from the MiBench suite [13]. They all involve encode/decode operations.

Table 1: **MediaZip benchmarks.**

Classification	MediaZip	Description
Audio	**ADPCM**	Adaptive Differential Pulse Code Modulation is well-known 16 bit PCM to 4 bit ADPCM encoder and decoder
	G.721	Voice compression encoder and decoder based on the CCITT(International Telegraph and Telephone Consulative Committee) G.711, G.721, and G.723
	GSM	Global Standard Mobile communications is full-rate speech transcoding encoder and decoder based on European 06.10 provisional standard.
Image/Video	**JPEG**	A standardized compression method for full-color and gray-scale images. JPEG is lossy image compression encoder(cjpeg) and decoder(djpeg)
	MPEG-2	MPEG-2 is video compression encoder and decoder for high-quality video transmission. The main kernel is Discrete Cosine Transform.
	EPIC	An image compression utility based on wavelet decomposition including run-length / Huffman entropy coding. EPIC is designed to allow fast decoding without floating point hardware
Encryption	**PEGWIT**	A program for public key encryption and authentication. It uses an elliptic curve and secure hash algorithms.
	Blowfish	A symmetric block cipher with a variable length key.
	Rijndael	A block cipher with the option of 128-, 192-, and 256-bit keys and blocks. Rijndael is selected as the National Institute of Standards and Technologies Advanced Encryption Standard.

The chosen multimedia encode/decode applications have been the main workload for personal appliances (computer, mobile communication, etc.) and embedded systems. A few years ago, static media applications related to image and sound were the main workload and only one-way (unilateral) media applications were generally used on the personal and industry computing. As modern technology gives high speed processing ability to personal appliances (PC, cellular phone, PDA, DVD, etc.), current trend of media applications is towards dynamic media application and two-way (interactive) application such as video conference, interactive TV and interactive game/movie.

The reason why we focus on multimedia encode/decode application is that these applications will still be a dominant workload of computers for years to come. Compression and encryption will continue to be important components of media workloads. Uncompressed multimedia data requires large storage space and it takes long time to send the data through limited transmission bandwidth. With the growth of multimedia-centric Internet applications, efficient encoding technology is required for storage and transmission.

In summary, the nine applications in the MediaZip suite are widely used in each area and form a representative component of multimedia applications. Especially, encryption also involves encode/decode process and will be one of the important workloads in the future as IT technology and e-business grows.

2.2. Simulation Environment and Methodology

We used the SUN Shade binary instrumentation tool [14] to obtain dynamic traces while executing MediaZip applications with 2 sets of data(small and large size of data) and different types of data having different statistical properties. To accurately extract microarchitecture independent characteristics of MediaZip applications, we use only classical optimization option in the compiler which can eliminate redundancies only in the assembly level code (common sub-expression elimination, constant propagation, etc). More aggressive optimizations such as unrolling, inlining or global scheduling optimizations are excluded since they can add or remove non-redundant operations and can change the characteristics of the workload [2].

2.3. Metrics

There are two primary characteristics that we investigate (i) memory accesses behavior and (ii) instruction level parallelism.

In general, most media applications are obtained by cascading a few stages consisting of a few kernels. Data that gets entered to each stage gets transformed by a kernel. In order to transfer the result of each stage to the next, temporary storage will be used. This results in several temporary memory accesses. In processors with specialized architectures such as the systolic array architecture [11][12], these temporary memory accesses will be avoided by directly forwarding data output from one stage to the next. Application specific processors can employ special techniques such as data forwarding between functional units without any real memory accesses. However, general purpose processors offer nothing more than the general purpose register file or caches in order to forward the data to the following stage. Registers are inadequate due to the large number of data elements that need to be forwarded. Forwarding through caches still necessitate accessing caches and the involved cache ports, etc.

In order to understand the significance of such temporary loads and stores, we define a metric called overhead memory access. During dynamic execution of programs, the load operation is used for actual input data fetch, coefficient data fetch for computational process, loop index fetch and the fetch of temporarily stored data. Similarly, the store operation is used for storing actual output data, updating next index in the loop and storing temporary data between computational stages. Therefore memory-accessing operations can be divided into essential load and store for input and output data, and intermediate (temporary) load and store for the process with coefficients, loop index and temporary data. Anything that we have classified as temporary represents something that could be avoided from being stored/loaded explicitly, if a non-programmable or application specific architectural solution is used. We define the overhead memory access instructions as all load and store instructions other than those that are true input and true output access instructions. The overhead memory access instructions per input/output access instruction is then computed as a ratio. We also quantify the overhead data transported to the central processor unit. Overhead data transported is the ratio of the number of bytes transported during temporary storage and retrieval of temporary data to the total number of bytes transported. This approach is similar to that in [15].

In addition, we attempt to quantitatively characterize the Instruction Level Parallelism (ILP) of the MediaZip application suite. To extract the critical path length and available parallelism information, the tool *Tetra* was used [16]. Tetra extracts a canonical form of the program from a serial execution trace to produce an execution graph. It is essentially the data flow graph of the program. In our simulation, we allow infinite resources to find the upper bound on the inherent ILP of each application.

3. Characteristics of MediaZip Applications

3.1. General Characteristics

The encoding process of MediaZip applications normally consists of transformation, quantization and entropy encoding as shown in Figure 1. Decoding is the inverse process of the encoding, because the coefficients of the encoding process used for quantization and entropy coding, have to be applied in decoding process to recover the original data. Most of the MediaZip applications consist of several stages of DSP kernels and other regular / irregular processes, and use them iteratively. For example, JPEG has four stages: color conversion, DCT (Discrete Cosine Transform), quantization and entropy encoding. Some stages have regular properties, which are related to regular memory accessing workload to fetch data and store the results of computation. Some stages have irregular operations, such as entropy encoding in which memory accessing depends on the statistical properties of the input data.

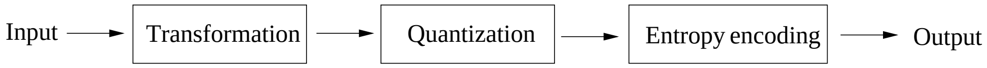
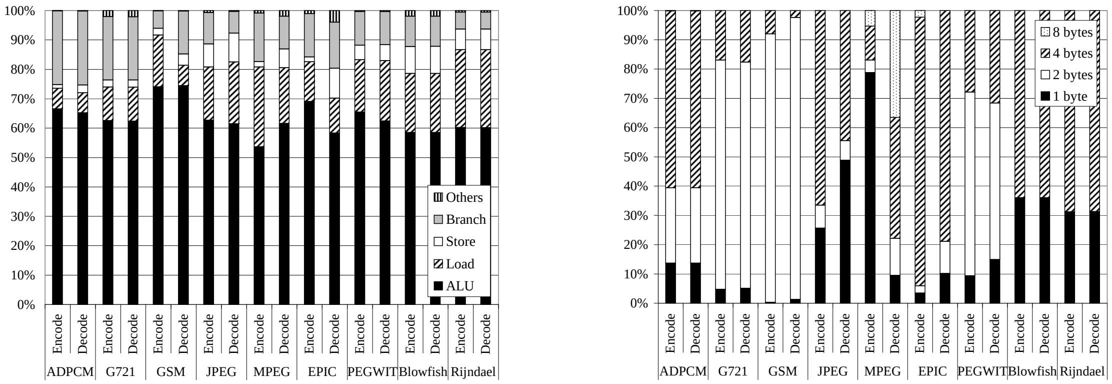


Figure 1: **Encoding process.**

To give a general idea of the benchmarks, first we present some basic properties of the benchmarks such as instruction mix, operand size mix, etc. Figure 2 (a) shows the dynamic instruction distribution during execution. This workload distribution provides the balance information of functional units in the design of superscalar processor. From this graph we can observe that there are overall similarities between the applications. For example, ALU operations consume a gigantic share of the overall instruction mix (65% on the average) which means that these applications are ALU-centric applications.

If multimedia extensions such as MMX, SSE-2 or AltiVec are used, 4 or 8 operations get done in one SIMD instruction, changing the computation intensive instruction mix to a memory intensive mix [4] [17]. Instruction mix information with SSE and AltiVec are available in [6][18]. In this paper, we leave them as the basic uniprocessor operations, to indicate the basic operation mix without heavy influence of the instruction set architecture (ISA). Another noticeable fact that emerges later on in this section is that, irrespective of the dominance of the computation instructions, the overall



(a) Instruction distribution

(b) Data type of load operation

Figure 2: **Instruction distribution and data type of load operations in MediaZip.**

execution time is strongly correlated with the amount of memory access operations.

From Figure 2 (a), it can be seen that the distribution of loads and stores is different between one dimensional applications (e.g. audio) and two dimensional applications (e.g. image). Even though 2-D applications can be separated into two 1-D processes for computational simplicity, they have to consider spatial correlations around pixel data which causes the application to consume more loads for fetching input, intermediate data and coefficients repeatedly. Encryption applications also use load and store operations as much as the 2-D applications.

Actual data size of load and store instructions are distributed between several types (1, 2, 4 and 8 bytes) as presented in Figure 2 (b). Most applications have used 1 byte, 2 byte and 4 byte data. 8 bytes data were used in ADPCM, JPEG, MPEG and EPIC, but only MPEG encode/decode and EPIC encode process have used 8 bytes data significantly. ADPCM, Blowfish, Rijndael and image applications heavily use 4 byte data, and G721, GSM and PEGWIT use mainly 2 byte data.

3.2. Overhead Memory-accesses per input/output element access

We quantified the number of temporary memory accesses per inevitable memory access. As shown in Table 2, overhead memory load frequency of load operation can be computed as (L-I)/I, where L is the number of load operations and I is input data size. In the same way, overhead memory store frequency can be (S-O)/O, where S is the number of overall store operations and O is actual output data size. Since the input and output access information is available in bytes and not as instructions or accesses, the ratio that can be meaningfully computed is with respect to the temporary/overhead memory access bandwidth. The first column in Table 2 indicates the ratio of temporary bytes loaded to the number of actual input bytes. The second column indicates similar information for stores. The third column in Table 2 represents overall memory-access overhead per input byte, which is represented by (L+S-I-O)/I. The fourth column indicates overhead memory-access bandwidth(OMB), the number of memory access instructions per useful input/output byte that is handled by the application. MPEG encoder has the biggest overhead and ADPCM has the smallest. The rightmost column in Table 2 presents overhead memory-access instruction frequency(OMIF), which indicates the number of load/store instructions required to load/store the overhead (temporary) data. The gap between overhead memory-access instruction frequency and overhead memory bandwidth tells data type used in whole process of each application. If all data accesses are byte accesses, the last two columns of data will be identical.

Table 2: **Overhead memory-access bandwidth.**

L,S,I,O : # of bytes, **L#,S#** : the number of instructions

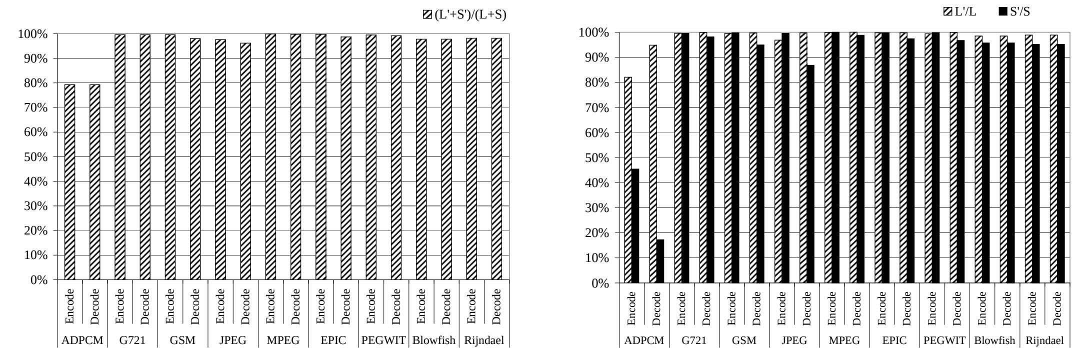
Applications		**Overhead Memory-access Bandwidth (OMB)**				**OMIF**
		(L-I)/I	**(S-O)/O**	**(L+S-I-O)/I**	**(L+S-I-O)/(I+O)**	**(L#+S#-I-O)/(I+O)**
ADPCM	Encode	4.6	0.8	6.0	3.8	0.6
	Decode	18.3	0.2	24.1	3.8	0.6
G721	Encode	240.8	218.4	296.7	236.3	102.6
	Decode	942.7	52.6	1,158.1	230.6	99.6
GSM	Encode	229.3	376.6	269.2	243.1	112.2
	Decode	341.1	18.8	534.1	48.9	23.5
JPEG	Encode	30.6	217.2	44.0	40.7	12.6
	Decode	355.2	6.6	492.6	25.0	9.8
MPEG	Encode	990.2	23,457.2	1,131.4	1,123.7	636.6
	Decode	58,375.8	81.5	72,181.9	427.8	86.1
EPIC	Encode	411.7	401.1	458.3	410.6	103.5
	Decode	397.5	37.5	738.5	74.2	20.6
PEGWIT	Encode	160.7	455.7	209.3	188.5	76.1
	Decode	960.8	29.4	1,253.7	117.3	46.7
Blowfish	Encode	64.8	22.4	43.6	43.6	14.3
	Decode	64.8	22.4	43.6	43.6	14.3
Rijndael	Encode	86.5	19.5	53.0	53.0	16.6
	Decode	86.6	19.5	53.0	53.0	16.7

MediaZip applications exhibit some symmetry in execution characteristics between the encode and decode processes. Applications ADPCM, G721, Blowfish and Rijndael exhibit good symmetry in overhead memory accesses, whereas the other applications do not. It is important to note that all 3 video applications are asymmetric with respect to the overhead access metric. Perhaps this points to opportunities that exist for dynamic resource partitioning between encode and decode phases of video applications.

3.3. Overhead accesses as a fraction of all accesses

Figures 3 shows the amount of temporary load (L′=L-I) and store (S′=S-O) operations for each application. Figure 3 (a) represents the temporary load/stores as a fraction of all load and store operations, and Figure 3 (b) represents the ratio split for loads and stores separately.

As shown in Figure 3 (a) and Figure 3 (b), an overwhelming fraction of the data transported between memory subsystem and the processor registers is temporary data. Considering loads and stores together, there is very good similarity between behavior of the encode process and decode process. The average value for the entire application suite is 96% in Figure 3 (a). A large part of these could have been avoided in an application specific processor specifically tailored to perform this task. A systolic array architecture to perform compression or encoding is a prime example of such a system [11][12].



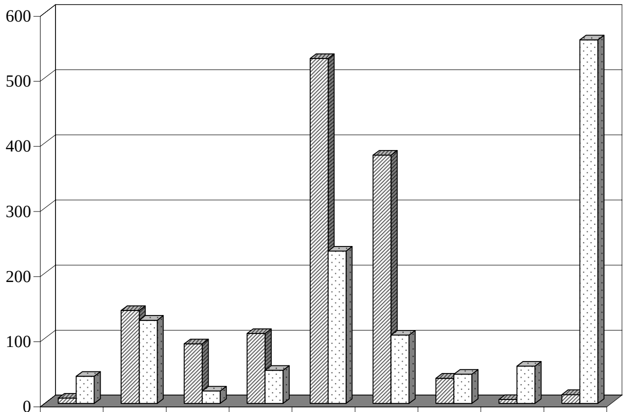
(a) (L′+S′)/(L+S)

(b) L′/L and S′/S

Figure 3: **Temporary data transport in MediaZip applications.**

3.4. Available Parallelism

Media applications are perceived to contain a lot of data parallelism, however, quantitative studies on the parallelism have been few. We characterize the intrinsic ILP(Instruction Level Parallelism) of MediaZip applications as a function of the inherent data dependencies and data flow constraints in the application. Essentially, a data flow graph of the application is constructed and relaxing all resource constraints, the maximum number of instructions that can be scheduled in each cycle is computed. To perform data dependency analysis, we used the tool Tetra [16] which is capable of evaluating program parallelism under specified resource and control constraints of the



	ADPCM	G.721	GSM	JPEG	MPEG2	EPIC	PEGWIT	Blowfish	Rijndael
▨ Encode	7.94	142.81	91.57	107.4	529.7	381.44	38.59	6.06	13.54
□ Decode	41.58	127.65	18.95	50.81	234	104.92	44.68	57.22	558.4

Figure 4: **Available parallelism of MediaZip application.**

hardware. In this study, we relax the hardware availability to infinite resources to extract micro-architecture independent (intrinsic) ILP. Most multimedia applications, including MediaZip application, have many loop iterations in the program. As the number of iterations are increased, the ILP can go up since each iteration of the loop can be scheduled in parallel. Figure 4 shows the available parallelism for the programs in the MediaZip application suite. The parallelism ranges from 6 to 529 in encode processes and 18 to 558 in decode processes. As shown in Figure 4, ADPCM has a small value of available parallelism. ADPCM quantizes the difference between the speech signals and a prediction signal, instead of quantizing current signal directly, imposing some serialization. The value of each sample is computed using previous sample values making it to contain very little parallelism.

Audio and image applications except for ADPCM have larger available parallelism in encoding than in decoding. On the other hand, encryption application and ADPCM shows large available parallelism in decoding, but extremely small available parallelism in encoding.

4. Evaluating the Robustness of OMB

4.1. The Effects of Data Properties on Overhead Memory Workload

In this section, we investigate the generality of overhead memory factor metric using several types of input data. The characteristics of the MediaZip applications are partially dependent of the statistical property of the input data. Normally, MediaZip applications consist of regular DSP algorithms (e.g. DCT) and irregular algorithms (e.g. entropy coding) as follows :

$$MediaZip\ \ Application = \alpha\cdot\mathbb{R} + \beta\cdot\mathbb{I}\mathbb{R}$$

where $\mathbb{R}$ represents regular kernel and α is its effect to the whole workload, on the other hand, $\mathbb{I}\mathbb{R}$ represents irregular algorithm and β is its effect to the whole application. We examined the effect with variation of data size and statistical property of the input data. Some applications like JPEG have an option for control of the quality of the compressed image. The option influences the compression ratio. We use JPEG application for comparing the results from different statistical environments.

Table 3: **The effect of input data size.**

Applications		Total icount	Input data [byte]	Compression ratio	(L+S-I-O)/(I+O)	
					OMIF	OMB
JPEG (small input)	encode	5,596,351	101,484	17.63	12.6	40.7
	decode	3,763,092	5,645	17.98	9.8	25.0
JPEG (large input)	encode	39,023,627	786,490	26.58	11.7	37.8
	decode	22,717,867	29,587	26.58	8.7	21.6
GSM (small input)	encode	185,109,936	295,040	0.10	112.2	243.1
	decode	74,062,478	30,426	9.70	23.5	48.9
GSM (large input)	encode	734,034,831	1,159,040	0.10	113.6	245.8
	decode	290,727,421	119,526	9.70	23.5	48.9
Blowfish (small input)	Encode	32,604,294	311,824	1.00	14.3	43.6
	Decode	32,596,554	311,825	1.00	14.3	43.6
Blowfish (large input)	Encode	337,690,865	3,247,552	1.00	14.2	43.4
	Decode	337,609,732	3,247,553	1.00	14.2	43.4

- The effect of input data size

We choose one application from each of the audio, image and encryption application groups to study the impact of input data size. Table 3 shows total instruction count, the size of input data, compression ratio and overhead memory factors in each application. We found that the overhead memory access values are largely independent of input sizes. In the case of GSM and Blowfish, the values are constant irrespective of input size. It is clear that these applications have large influence from regular algorithms. On the other hand, the results of JPEG are different suggesting greater influence of irregular algorithms than that of GSM and Blowfish.

- The effect of quality factor on JPEG

Table 4: **The effect of quality factor on JPEG input data.**

JPEG Application		**Total icount**	**Input data [byte]**	**Output data [byte]**	**Compression ratio**	**(L+S-I-O)/(I+O)**	
						OMIF	**OMB**
JPEG (quality: 30)	encode	37,538,681	786,490	14,733	53.38	11.5	37.2
	decode	22,052,906	14,733	786,447	53.38	8.5	20.9
JPEG (quality: 75)	encode	39,023,627	786,490	29,587	26.58	11.7	37.8
	decode	22,717,867	29,587	786,447	26.58	8.7	21.6
JPEG (quality: 95)	encode	42,842,506	786,490	69,053	11.39	11.9	39.0
	decode	28,083,790	69,053	786,447	11.39	9.0	23.1

JPEG permits quality control by varying number of quantization levels that result in varying compression ratios. Varying this factor, we investigate the sensitivity of overhead memory access ratio on compression ratio of MediaZip application. The JPEG application used in this paper has 100 levels for its quality switch (0 is worst case, 100 is best case, and default level is 75). We used 30, 75 and 95 as quality factors. Low value for the quality factor means that the quantization step is sparse, leading to more redundancy for entropy coding and higher compression ratio. But the decoded image quality is very poor because significant information is lost in the quantization process. Higher quality value means dense quantization steps, leading to lower compression ratios and high quality of decoded result. As shown in Table 4, total number of instructions increases as quality factor increases, and compression ratios show significant difference between the three cases. However, overhead memory bandwidth values are almost the same indicating that the effect of irregular algorithms is not significant when considering the effect of quality factor on JPEG.

- The effect of statistical property on JPEG

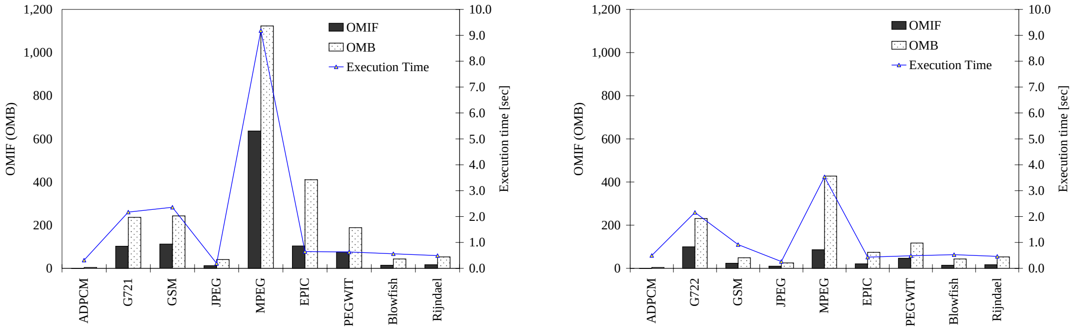
To measure the effect of irregular algorithms in another viewpoint, we use several inputs for JPEG with different statistical properties. To get different statistical properties for the data, we used the Gaussian blur filter as LPF (Low Pass Filter) and sharpening filter as HPF (High Pass Filter). The standard deviation values from each mean value in the luminosity channel of the input data are 45.20 (LPF), 50.03 (normal), and 54.37 (HPF) for three types of input data. In the rest of the study, we used default quality factor (75) in all cases. Basically, the input image with lower standard deviation shows higher spatial redundancy and it can get higher compression ratio as shown in Table 5. Even though compression ratios are very different, overhead memory instruction frequency (bandwidth) values are almost the same for the three cases. And they are also same with the values of Table 4.

Table 5: **The effect of statistical property on JPEG input data.**

JPEG Application		Total icount	Input data [byte]	Output data [byte]	Compression ratio	(L+S-I-O)/(I+O)	
						OMIF	OMB
JPEG (LPF)	encode	37,178,783	786,477	12,425	63.30	11.5	37.0
	decode	21,909,667	12,425	786,447	63.30	8.5	20.9
JPEG (Normal)	encode	39,023,627	786,490	29,587	26.58	11.7	37.8
	decode	22,717,867	29,587	786,447	26.58	8.7	21.6
JPEG (HPF)	encode	40,883,141	786,477	44,978	17.49	11.9	38.6
	decode	25,880,963	44,978	786,447	17.49	8.9	22.4

4.2. Correlation between Memory Access Overhead and Execution Time

Figure 5 shows overhead memory workload for each application's encoding and decoding process. In most applications, encoding process requires more overhead memory. Particularly, MPEG needs extremely high amount of overhead/temporary memory accesses. Figure 5 also shows execution time of each application. Although memory accesses are less than a third of the overall instruction mix, the amount of memory accesses and overhead memory accesses are well correlated with the overall execution time as shown in Figure 5, indicating memory access is a serious issue when media applications are implemented on GPPs.



(a) Encode application

(b) Decode application

Figure 5: **Overhead memory accesses and execution time of MediaZip.**

5 Conclusion and Future work

Media applications are typically executed on either ASPs, ASICs or GPPs. GPPs are flexible and allow changes to media algorithms and standards, however, utilize a large number of software instructions to accomplish the task. While it is intuitively clear that software implementations have execution overhead, previous research efforts have not quantified the impact of performing these applications in software. In this paper, we analyze the implications of using GPPs for media compression and encryption applications. We found that 4 to 1100 bytes of temporary data are transported to the CPU for every byte of actual input/output data fed into or out of the application. In order to accomplish this data transport, 1 to 630 memory access instructions are used per input/output byte, in addition to the memory accesses required to load actual input data or store actual output data. Special purpose architectures, especially those allowing direct forwarding of data from processing element to processing element (such as systolic array architectures) are extremely relevant in this context, because they can avoid a significant part of this overhead.

We also evaluated the parallelism of media applications. As expected media applications do contain large amounts of parallelism. Hence ASICs, ASPs and GPPs to execute media applications should be designed with features to extract this parallelism.

Future research can focus on fine-grain parallel architectures that explicitly support direct forwarding of temporary results between stages of computations in media applications. Media applications keep evolving. Future studies should also focus on new and emerging applications.

References

[1] D. Talla and L. K. John, “Cost-effective hardware acceleration of multimedia applications,” In Proceedings of the IEEE International Conference on Computer Design ’01, pp. 415-424, 2001.

[2] J. Fritts, W. Wolf and B. Liu, “Understanding multimedia application characteristics for designing programmable media processors” Proceedings of SPIE, vol. 3655, pp. 2-13, Jan. 1999.

[3] R. B. Lee, “Multimedia extensions for general-purpose processors,” Proc. IEEE Workshop on Signal Processing Systems, pp. 9-23, Nov. 1997.

[4] P. Ranganathan, S. Adve and N. Jouppi, “Performance of image and video processing with general-purpose processors and media ISA extensions,” Proc. 26th IEEE/ACM Symposium on Computer Architecture, pp. 124-135, May 1999.

[5] K. Diefendorff and P. K. Dubey, “How multimedia workloads will change processor design” Computer, vol. 30, pp. 43-45, Sep. 1997.

[6] D. Talla, L. K. John, V. Lapinskii and B. L. Evans, “Evaluating Signal Processing and Multimedia Applications on SIMD, VLIW and Superscalar Architectures,” In Proceedings of the IEEE International Conference on Computer Design, 2000.

[7] D. Talla, L. K. John, “Execution Characteristics of Multimedia Applications on a Pentium II Processor,” In Proceedings of the IEEE International Performance, Computing and Communications Conference, 2000.

[8] D. Talla, L. K. John, “Performance Evaluation and Benchmarking of Native Signal Processing,” Euro-Par ’99, pp. 266-270, 1999.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “MediaBench: A tool for evaluating and synthesizing multimedia and communication systems,” in Proceedings of the 30th Annual International Symposium on Microarchitecture, pp. 330-335, Dec. 1997.

[10] N. T. Slingerland and A. J. Smith, “Cache performance for multimedia applications” In Proceedings of the 15th IEEE International Conference on Supercomputing, pp. 204-217, Jun. 2001.

[11] N. Ranganathan and S. Henriques, “A Systolic VLSI Chip for Data Compression,” Proc. of IEEE International Symposium on VLSI Design, Jan. 1991.

[12] M. Kovac, N. Ranganathan, “JAGUAR: a fully pipelined VLSI architecture for JPEG image compression standard,” Proceedings of the IEEE, Vol. 83 pp. 247-258 Feb. 1995.

[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge and R. B. Brown, “MiBench: A free, commercially representative embedded benchmark suite,” In proceedings of the 4th annual IEEE International Workshop on Workload Characterization, pp. 3-14, Dec. 2001.

[14] R. F. Cmelik and D. Keppel, “Shade: A fast instruction-set simulator for execution profiling,” SUN Microsystems Inc., Technical Report SMLI TR-93-12, 1993.

[15] D. Hammerstrom and E. Davidson, “Information content of CPU memory referencing behavior,” In Proceedings of the 4th International Symposium on Computer Architecture, pp. 184-192, 1977.

[16] T. M. Austin and G. S. Sohi, “TETRA: Evaluation of serial program performance on fine-grain parallel processors,” University of Wisconsin Technical Report #1162, July 1993.

[17] R. Bhargava, L. K. John, B. L. Evans, R. Radhakrishnan, “Evaluating MMX Technology Using DSP and Multimedia Applications,” In Proceedings of the IEEE Symposium on Microarchitecture, pp. 37-46, 1998.

[18] H. Nguyen, L. K. John, “Exploiting SIMD Parallelism in DSP and Multimedia Algorithms using the AltiVec Technology,” In Proceedings of ACM International Conference on Supercomputing, pp. 11-20, 1999.