An Experimental Analysis of the Effect of the Operating System on Memory Performance in Embedded Multimedia Computing

Sangsoo Park
School of CSE &
Inst. of Computer Technology
Seoul National University
Seoul, 151-744, Korea
sspark@cslab.snu.ac.kr

Yonghee Lee
School of CSE &
Inst. of Computer Technology
Seoul National University
Seoul, 151-744, Korea
yhlee@cslab.snu.ac.kr

Heonshik Shin
School of CSE &
Inst. of Computer Technology
Seoul National University
Seoul, 151-744, Korea
shinhs@cslab.snu.ac.kr

ABSTRACT

As embedded systems grow in size and complexity, an operating system has become essential to simplify the design of system software, for which more accurate analysis of its impact on memory performance is required. In this paper, we intend to investigate how the OS influences memory performance at run time by quantitatively evaluating the memory system behavior of an MPEG-4 application running on embedded Linux. Through the use of extensive simulations we have confirmed that the OS has poor memory performance with less memory locality than applications. The results of our experimental analysis are deemed useful for helping embedded system designers understand the memory performance of the OS and the application within a system, extending their capability to design a more power-aware and faster system.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design studies, Performance attributes; D.4.8 [**Operating systems**]: Performance—*Measurements, Simulation*

General Terms

Performance

Keywords

Operating system, multimedia, memory performance, embedded system

1. INTRODUCTION

Omnipresent embedded systems are as revolutionary in their way as the wheel. Diversified modern machines, however, demand a far higher degree of intricacy from embedded systems than ancient tools did from the wheel. These requirements are translated into technical restrictions on embedded systems which dissociates these systems from general-purpose systems to some degree. Functionality is important in both cases; moreover, embedded systems must meet many other application-specific constraints on, for example, performance, power consumption, manufacturing cost and time-to-market [21].

As embedded systems grow in size and complexity, an operating system becomes essential to simplify the design of system software. In addition, an OS provides not only the resource management but also a common API to help embedded application programmers develop target systems with ease and efficiency. Most previous studies of embedded system performance have, however, overlooked OS behavior and its impact on memory performance. That is why intend to investigate how the OS influences memory performance at run time.

Multimedia applications are quickly moving from general-purpose computers to embedded systems such as portable handhelds and networked terminal devices. For many of these systems, a pure software-based approach is cost-effective and allows the tracking of evolving new standards for multimedia applications. MPEG-4 [16] is one of the several widely used international standards for video compression, with applications ranging from mobile multimedia to Internet streaming video. As the speed gap between CPUs and memory widens, memory performance is coming to dominate overall performance in most applications [5]. While video decoding and encoding are computationally intensive, recent studies focus on their memory system performance as these applications also demand more and more memory bandwidth [10].

Analyzing the memory performance of applications in embedded systems is more difficult than for general-purpose systems. Varying a few parameters during system design may have significant effect on the overall performance and cost. To explore this problem, we set out quantitatively to evaluate the memory system behavior of a software MPEG-4 application running on an embedded operating system, by varying both hardware design parameters and the selection of software components.

The rest of this paper is organized as follows: Background and related work are reviewed in Section 2. Section 3 introduces our system model and Section 4 describes our methodology, experimental environments and the experimental results. In Section 5, we analyze the experimental results in order to understand the memory system behavior. Finally, this paper is concluded in Section 6.

2. RELATED WORK

In general, the impact of operating systems on the memory performance has not been analyzed thoroughly because it is often difficult to obtain a meaningful trace of OS activity. Chen and Bershad [7] have evaluated the memory performance of two different implementations of the UNIX OS on a general-purpose workstation. Their evaluation was based on software instrumentation which rewrites assembly code so that it records a complete trace of instruction and memory addresses. Chen and Bershad analyze the locality of memory references and look for interference between the OS and the application. They focus on conventional applications such as text manipulation and a C compiler running in a fixed hardware setup. In this experiment, however, it should be noted that it is possible for software instrumentation to contaminate the traced memory references.

McKee et al. [17] have evaluated the memory performance of a software MPEG-4 decoder on a workstation, using a hardware monitoring tool that provides a limited range of functions, such as event counters for cache misses, TLB misses and write buffer stalls. Their results show the impact of both application and OS on the memory performance but do not differentiate the OS from the application.

Xu et al. [20, 22] and Soderquist and Lesser [19] use trace-driven cache simulators to evaluate the memory performance of multimedia applications as well as conventional applications by varying cache design parameters. They also used a hardware monitoring tool for their hardware setup to verify and compare the simulated results. Because the cache simulator only deals with memory references, it provides no information on the overall performance. Also, their traces only contain the memory reference for the application programs.

To refine the art of memory performance analysis, we must be able to trace the memory references of both application and OS, and evaluate their overall performance, while varying hardware design parameters and software components, all without altering the system behavior through our use of instrumentation.

Previous studies dealing specifically with the memory behavior of multimedia systems have produced contradictory reports. On one hand, early studies in multimedia applications suggested that the memory system may be a performance bottleneck. Diefendorff and Dubey [10] point out that cache performance is poor because typical data sets and working sets are so huge that current cache architectures cannot handle them. Consistent with this suggestion, Chen et al. [6] and Zucker et al. [23] addressed the memory performance bottleneck problem in software MPEG-1 and MPEG-2 decoders by employing both software and hardware prefetching techniques.

On the other hand, some recent papers claim that multimedia applications actually have better memory performance than conventional applications. Xu et al. [20, 22] and Slingerland and Smith [18] compare the memory performance of conventional and multimedia applications. Their results show that multimedia applications are characterized by high cache hit rates and low bus bandwidth requirements. McKee et al. [17] also present similar results.

3. SYSTEM MODEL

A hardware system needs to be designed specifically to meet the requirements of embedded system. Most embedded processors such as the ARM, MIPS and SuperH devices offer a wide range of CPU cores for various design goals, because cost and performance of the embedded system are greatly affected by the choice of a CPU core organization [14]. The relevant design criteria usually include the cache size, the clock frequencies of the CPU and bus, the characteristics of the floating point unit (FPU) and also those of the memory management unit (MMU). Another notable emerging technology is the system-on-a-chip (SoC) which consists of a CPU core and a number of hardware control blocks for external I/O devices. In order to minimize the cost for a given performance level, device size, and power consumption by the target system, a SoC is configured with a selective use of CPU components. Embedded system designers should be able to decide which components to select to meet their design goals.

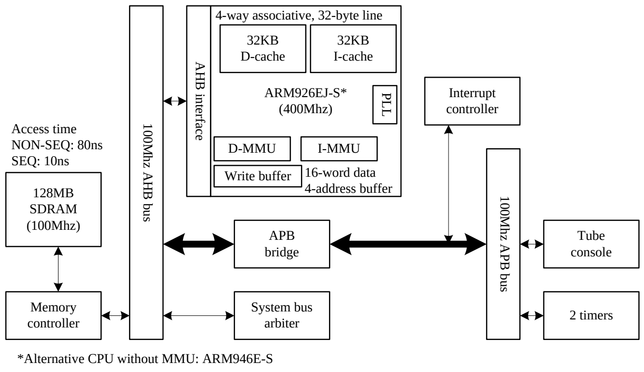


Figure 1: An example of a hardware system organization: the system configuration used in our experiment.

Currently audio and video playback and streaming in mobile devices such as PDAs and mobile phones are the most popular multimedia applications in embedded computing. Their requirements are characterized by high performance to support real-time video decoding, but with low power consumption and small size for portability. In this paper, we will focus on alternative CPU core organizations for performance and size.

The minimum set of components required to support an OS includes an interrupt controller, a timer, and some medium for a file system, together with a CPU and RAM. To experiment with varying hardware design parameters like cache size and MMU, we selected the synthesizable ARM9E core as a target platform. Fig. 1 illustrates the physical system configuration used in our study. Note that we have used the ARM926EJ-S as the CPU with MMU, and the ARM946E-S as the CPU without MMU. The lowest 64MB of SDRAM is reserved for the file system, which stores compressed and decompressed video as well as the executable binaries of the application.

It is now commonplace for embedded systems to adopt Linux as an operating system and we have followed this lead. Linux runs on the target both with and without MMU, but the software architecture of Linux is quite different in each case. As our software MPEG-4 application, we have used a simple profile version of the optimized MPEG-4 reference software from NCTU [3]. Because the MPEG-4 software contains floating-point arithmetic and the target CPU core has no FPU, some operations need to be emulated by software by kernel floating-point emulation (FPE) or by an FPE library.

As depicted in Fig. 2, the processing steps for our MPEG-4 application are as follows: After initiation of the OS and loading of the MPEG-4 application, the application reads a video frame from the file system, encodes or decodes it, stores the result in the memory buffer, and then writes the buffer to the file system until it encodes or decodes the last video frame. In this paper, we will deal

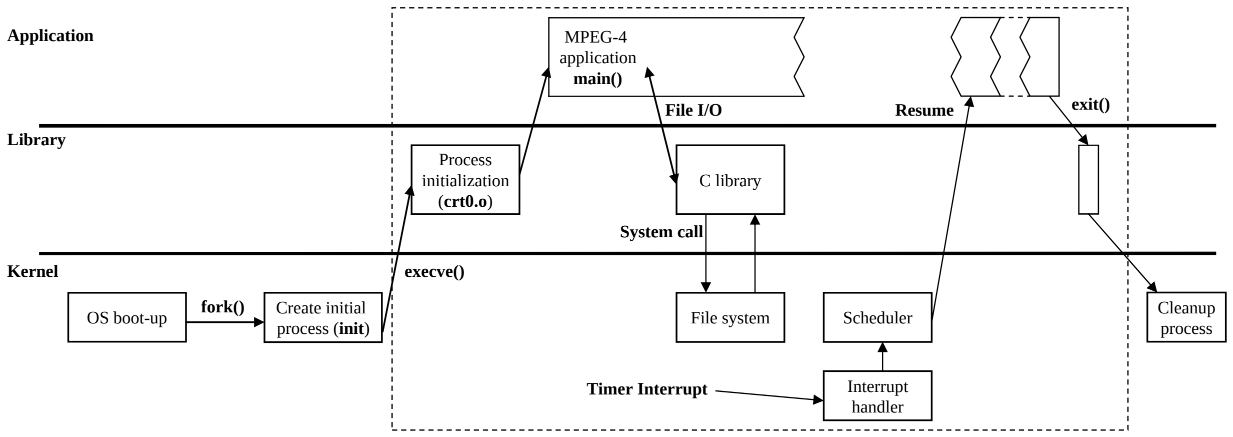


Figure 2: Software run-time behavior: Stages in the MPEG-4 application adopted for our experiment.

with the processing steps inside the dotted line shown in Fig. 2 because we are only interested in the processing of the MPEG-4 application.

We will start by investigating the overall performance [1] and the way that the memory behavior varies with changes in the design criteria, namely cache size, clock frequency, MMU and FPU. We evaluate the memory performance of a software MPEG-4 application while changing these design criteria within the feasible ranges of the parameters, considering what is commercially available rather than theoretical cache design parameters such as cache line size and associativity [20, 22]. Subsequently, we will analyze the effect of each software component - kernel, library and application - on memory performance.

4. MEASUREMENT

4.1 Experimental Setup

Before the target platform is built, an in-depth analysis of the system performance requires a full-scale system simulation that can run the target application together with its OS. Although RTL-level simulators and transaction-level simulators are often employed as analysis tools in the early stages of system design, they often turn out to be much too slow [8].

We therefore used ARMulator, a highly configurable system simulator provided by ARM which it can not only simulate various CPU cores but also model different types of memory, a range of cache architectures, and also external hardware. It is not 100% cycle-accurate, but it is known to have acceptable accuracy and moderate simulation time requirements, making it suitable for performance comparison [4, 15]. To evaluate the OS behavior and its impact we have ported Linux to ARMulator and used it to run the MPEG-4 application. Table 1 summarizes the software environment for the experiments.

The uClibc C library is much smaller than the GNU library, while it is able to support CPUs both with and without MMU. The file system is managed by an Memory Technology Device (MTD) subsystem and formatted as an ext-2 file system.

The quality of an MPEG-4 video depends on a set of physical properties: its resolution, bit rate, and the number of encoded frames. These parameters affect the memory performance as well as the encoding and decoding time and the input data size. To analyze their influence on the performance, we conducted experiments with some data sets, with the results reported elsewhere [1].

[1]The overall performance is defined as the system performance measured as a result of application processing and expressed in terms of, for example, jobs/sec or frames/sec.

Table 1: Software environment.

CPU	ARM926EJ-S	ARM946E-S
Kernel	linux-2.4.21-rmk1	linux-2.4.21-uc0
Library	glibc-2.2.3	uClibc-0.9.19
Compiler	gcc-2.95.3	gcc-2.95.3
Compile option	-O3	-O3

These results may be summarized as follows: (1) the memory performance of an encoder is better than that of decoder in all cases; (2) the cache miss ratio for decoding or encoding 100 frames is slightly lower than decoding or encoding 10 frames; (3) decoding a video with a higher bit rate or a lower resolution gives better memory performance, whereas encoding a video with a lower bit rate or a higher resolution gives better memory performance.

In the following experiments, we use a short but the representative sequence of operations which decoding and encoding 10 frames of 112kbps cif-format video.

4.2 Experimental Results for Cache and Clock Frequency

The results of using different cache sizes and clock frequencies are compiled elsewhere [1]. To summarize these results, the execution time of a decoder decreases by about 43% as the cache size increases from 4KB to 64KB, but it saturates at 32KB; and the execution time of the encoder decreases by about 20% as the cache size increases from 4KB to 64KB, but it too saturates at 32KB. The data cache write miss ratio of the decoder was found to be relatively high. This results in a high bus utilization, which implies that speed of memory access is responsible for a large part of the overall performance in the case of the decoder.

Another point to be addressed is the cache write policy. One of two alternative policies, write-back and write-through, is activated while the OS boots up. In general, how the cache write policy affects the overall performance depends on the application [12]. Since we have observed that write-back always shows the better performance, we use only write-back in the following experiments.

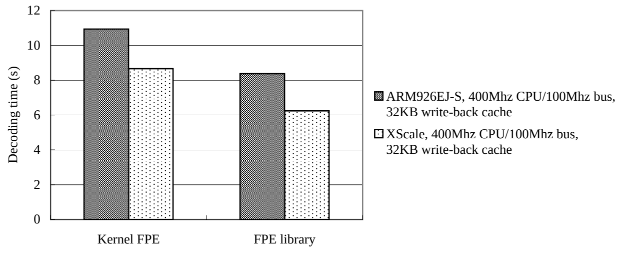


Figure 3: Simulated and measured MPEG-4 decoding time for performance comparisons (112kbps cif format).

In the meantime, we have compared the execution times of the decoder and the encoder as we change the CPU and memory bus clock frequencies. According to the results from this experiment, we have also confirmed that speed of memory access for the most part affects the overall performance in the decoder, which is memory-intensive as well as computation-intensive.

4.3 Experimental Results for the MMU

The experimental results for the system with and without an MMU are summarized in Table 2. In general, the system without an MMU gives better overall performance for the same application because it is free from TLB miss handing overheads. As shown in Table 2, the system without an MMU has better memory performance. Despite this fact, the execution of both decoder and encoder in the system without an MMU takes longer than in the system with an MMU, by about 20% and 26%, respectively. This unexpected result will be analyzed in Section 5.1.

4.4 Experimental Results for the OS

To evaluate the impact of the operating system on memory performance, we conducted simulations with applications only, excluding the OS; otherwise, the environment remains the same as described in Section 3. The experimental results of the simulations with and without the OS are compared in Table 3. With the operating system, the execution time of the decoder increases by 27% in the system with an MMU, and by 44% in the system without an MMU, whereas the encoder takes 20% longer in the system with an MMU and 44% in the system without. These results demonstrate that the OS comes with its own overhead and affects the memory performance.

We will attempt to explain about the column labeled ”App. + OS(uClibc)” in Table 3 in terms of OS behavior and memory performance in Section 5.2.

4.5 Experimental Results from FPE

Fig. 3 depicts the way in which the MPEG-4 decoding time varies with the FPE method used in the simulated system illustrated in Fig. 1. These experimental results are also compared with a real hardware platform, namely Intel’s XScale-based BRH evaluation board from ADI Engineering [2]. This has some additional features improving performance, such as a branch target buffer, a fill buffer and a 2KB mini data cache [13]. As shown in Fig. 3, there is a performance gap of around 25% between the two methods and this gap will be analyzed in Section 5.3.

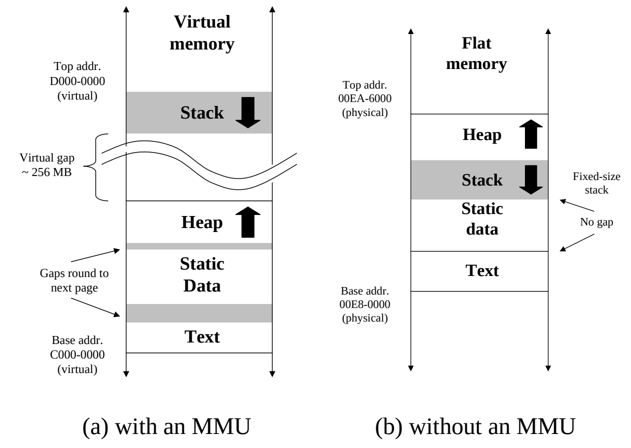


Figure 5: Memory map for the system with an MMU and without an MMU.

5. ANALYSIS

In Section 4, we evaluated the performance of a software MPEG-4 by varying the design parameters. On the basis of the experimental results, we are able to characterize its overall performance and to identify opportunities for improving the design of hardware and software components. Fig. 4 summarizes the experimental results. In the figure, the performance improvement ratio archived by varying each design parameter - cache size, CPU clock frequency, and memory bus frequency is depicted. The performance improvement ratio for the system with MMU and without MMU as well as kernel FPE and FPE library are also compared.

From the figure, we see that the performance of the encoder increases proportionally as the CPU clock frequency increases. But it saturates quickly when the memory clock frequency and the cache size are increased, variables which are tightly related to the memory performance. However, the performance of the decoder continues to be improved by increasing CPU and memory clock frequency. The performance of the decoder does saturate as the cache size increases but the cache does have a significant effect on the performance. This confirms that the encoder is highly computation intensive, whereas the decoder is both computation and memory intensive. Changes to the MMU and FPE have a similar impact on the performance of both the decoder and the encoder.

5.1 Effect of the MMU

In Section 4.3, we unexpectedly observed that the execution time in the system without an MMU is longer than in the system with an MMU. The C library is the only software component that changes between the two systems. We therefore rebuilt the MPEG-4 application with uClibc to run on the system with MMU, and performed additional experiments in order to find out what caused this result.

In Table 3, the column ”App.+OS(uClibc)” contains the results from this new setup. The execution times in the system without an MMU are now shorter than for the system with an MMU by about 5-10%, as we initially expected. This implies that the GNU C library outperforms uClibc and that the system without an MMU gives better overall performance for the same application because it is free from TLB handing overheads.

The OS for the system without an MMU has advantages that it does not have to handle per-process page tables (or TLB misses) and the associated protection required in the virtual memory model.

Table 2: Impact of the MMU: 400Mhz/100Mhz CPU/bus clock frequency, 32KB I,D-cache, write-back cache.

	Decoder		Encoder	
	w/ MMU	w/o MMU	w/ MMU	w/o MMU
Instr. cache miss ratio (%)	0.09	0.03	0.01	0.02
Data cache miss ratio (%)	11.44	10.66	1.17	0.91
Read	1.34	1.05	0.17	0.11
Write	32.68	26.61	8.32	3.02
Write buffer stall ratio (%)	20.01	28.56	2.58	4.24
TLB miss ratio (%)	0.92		0.9	
Instruction	0.08		0.01	
Data	1.99		0.33	
Bus utilization (%)	51.03	51.45	8.33	7.93
Execution time (s)	0.74	0.92	13.08	17.53

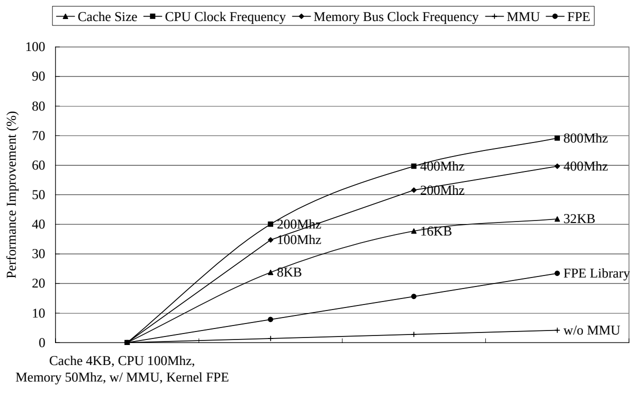
Table 3: Impact of the operating system: 400Mhz/100Mhz CPU/bus clock frequency, 32KB I,D-cache, write-back cache.

	ARM926EJ-S (w/ MMU)			ARM946E-S (w/o MMU)	
	App. only	App. + OS	App. + OS(uClibc)	App. only	App. + OS
Instr. cache miss ratio (%)	0.01	0.09	0.05	0.01	0.03
Data cache miss ratio (%)	17.25	11.44	9.89	17.25	10.66
Read	1.95	1.34	0.96	1.94	1.05
Write	52.07	32.68	24.84	52.07	26.61
Write buffer stall ratio (%)	22.75	20.01	25.44	24.78	28.56
TLB miss ratio (%)	0.00	0.92	0.57		
Instruction	0.00	0.08	0.05		
Data	0.00	1.99	1.16		
Bus utilization (%)	58.57	51.03	48.25	61.40	51.45
Execution time (s)	0.54	0.74	0.96	0.52	0.92

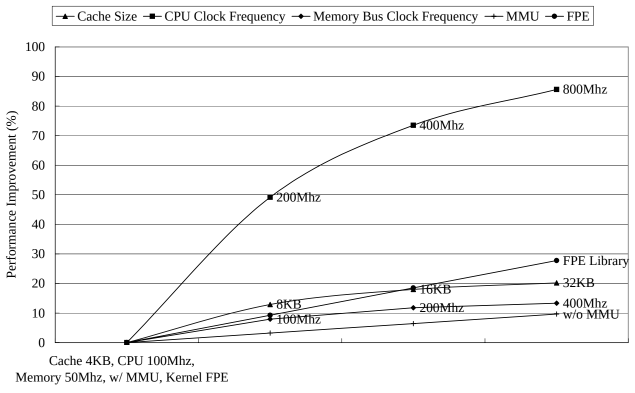
(a) Decoder

	ARM926EJ-S (w/ MMU)			ARM946E-S (w/o MMU)	
	App. only	App. + OS	App. + OS(uClibc)	App. only	App. + OS
Instr. cache miss ratio (%)	0.00	0.01	0.02	0.00	0.02
Data cache miss ratio (%)	1.32	1.17	1.03	1.32	0.91
Read	0.15	0.17	0.11	0.15	0.11
Write	29.62	8.32	3.43	29.64	3.02
Write buffer stall ratio (%)	3.55	2.58	4.37	4.10	4.24
TLB miss ratio (%)	0.00	0.09	0.11		
Instruction	0.00	0.01	0.01		
Data	0.00	0.33	0.18		
Bus utilization (%)	8.24	8.33	8.27	9.20	7.93
Execution time (s)	10.57	13.08	19.40	9.47	17.53

(b) Encoder



(a) Decoder



(b) Encoder

Figure 4: Performance improvement ratio archived by changes in design parameters.

Instead it uses the flat memory model as depicted in Fig. 5 [9]. But this introduces restrictions on the use of some features of the API such as the `fork()` system call, and on handling memory-related features. Application programmers are still able to allocate non-overlapped memory regions to the application, but with caution. They should define the stack size carefully in order to avoid an overflowing stack from polluting static data or code, which would lead to a system collapse.

5.2 Effect of the OS

We observed in Section 4.4 that employing the OS significantly increases the overall execution time. To identify the extent to which each software component impacts the CPU and memory traffic, we divided the experimental results in Table 3 into the three categories of kernel, application, and library. Table 4 describes the number of instructions that were performed in each category. The number of instructions corresponding to the application remains nearly constant, and we deduce that it is the additional instructions given by the library and the kernel that increased the CPU time.

Fig. 6 summarizes the proportion of each category - cache miss, write buffer stall and TLB miss ratio - in the memory performance metrics. In this figure, the proportion of the kernel instructions is less than 5% for the decoder and 0.26% for the encoder, but the memory traffic of the kernel is 14-26% for the decoder and 7-15% for the encoder. This implies that the OS kernel has poor memory performance, which is mainly attributable to the characteristics of OS routines, which are exception-driven and intermittently invoked [7].

Further results were obtained by repeating these experiments on a benchmark suite for embedded systems, MiBench [11], in order to find out how the OS behavior and its impact vary between different applications. These experimental results are presented elsewhere [1]. We observed that the OS kernel has poor memory performance in the benchmark applications which is consistent with the results presented in this paper.

5.3 Effect of FPE

Table 5 reports the number of instructions executed in the kernel with FPE. After this change, the number of kernel instructions increases while the number of library instructions is reduced. We also observe that a large proportion of the CPU and memory traffic in the library is caused specifically by the FPE library.

Table 5: Number of instructions executed in the kernel FPE (ARM926EJ-S).

	Decoder	Encoder
Kernel	66,771,291	1,593,901,622
Application	77,085,384	2,631,778,060
Library	3,962,210	3,767,988

An undefined exception (or software trap) encountered by any floating point arithmetic causes an op-code to be fetched and emulated in the kernel FPE as shown in Fig. 7 (a). That makes this method quite slow because of the frequent context switches between application and kernel, whereas no such context switch occurs when using the FPE library.

5.4 Implications

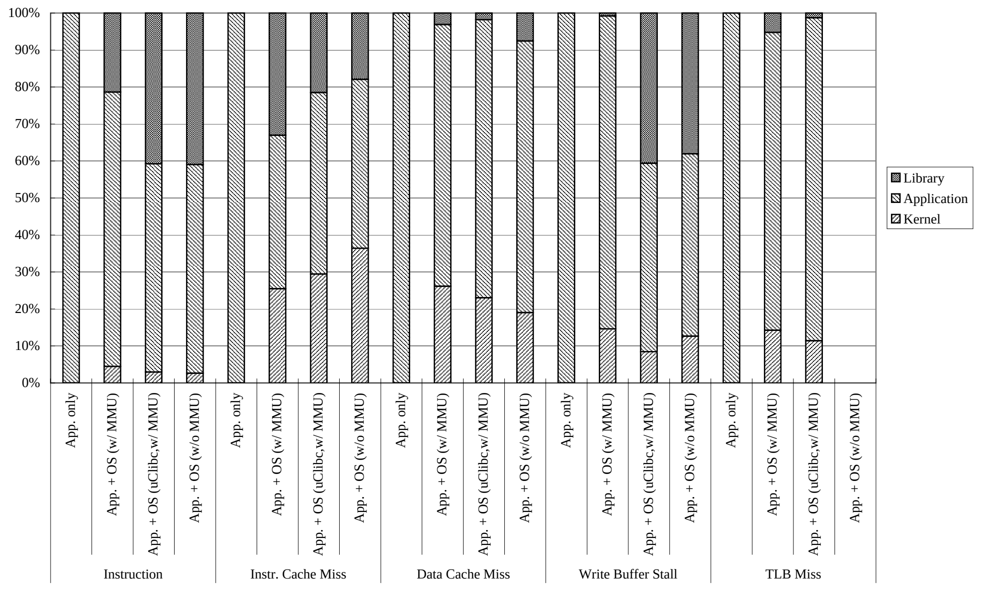
So far we have analyzed the effect of the OS on the MPEG-4 encoding and decoding at run time and observed the poor memory performance of the OS. Having quantitatively evaluated the OS behavior and its impact, we intend to help a system designer understand the overhead caused by the interplay between hardware and software components. While the system designer tries to determine a set of hardware design parameters that will meet requirements such as size, performance, and power consumption, they should be aware that the software components depending on their hardware counterparts will have a significant impact on performance. Whether the target application has FP arithmetic or not, or whether the maximum stack size is already known for the target application could be important design considerations. In this paper, we provide the system designer with clues about how the software components will affect their design goal.

6. CONCLUSIONS

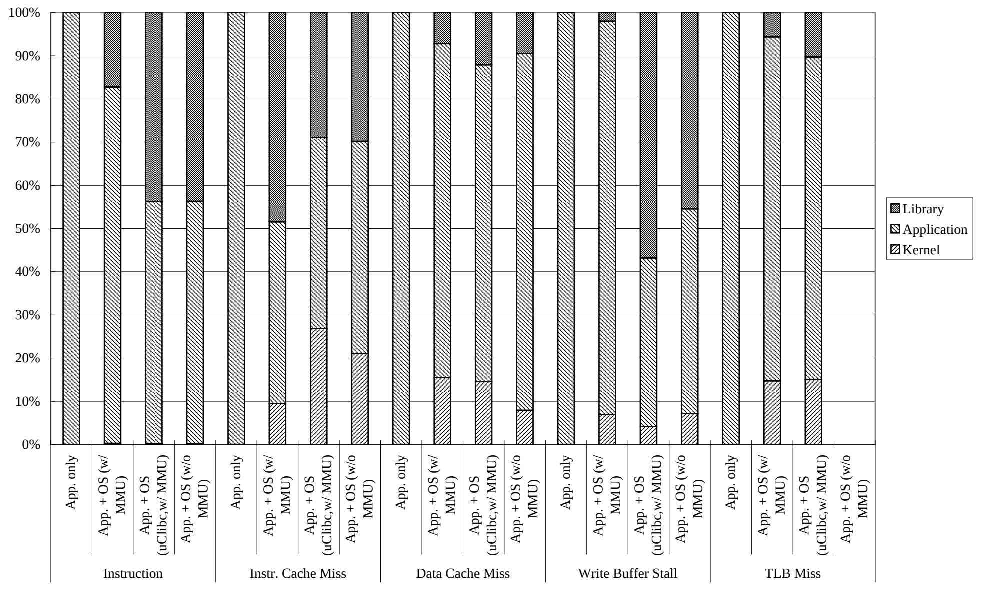
In this paper, we have quantitatively evaluated the memory system behavior of a software MPEG-4 application in an embedded system by varying both hardware design parameters and software components. We also analyzed the effect of each software component on memory performance. We have also investigated how the OS influences the system at run time and analyzed the behavior of the OS.

Table 4: Number of instructions executed.

		ARM926EJ-S (w/ MMU)			ARM946E-S (w/o MMU)	
		App. only	App. + OS	App. + OS(uClibc)	App. only	App. + OS
Decoder	Kernel		4,682,452	4,044,275		3,621,334
	Application	75,369,923	77,629,880	77,629,584	75,369,928	77,929,561
	Library		22,398,959	56,280,438		56,551,698
Encoder	Kernel		8,238,299	8,767,214		7,471,374
	Application	2,696,156,312	2,660,794,347	2,654,397,397	2,696,156,181	2,649,120,290
	Library		555,165,701	2,071,852,802		2,060,961,163

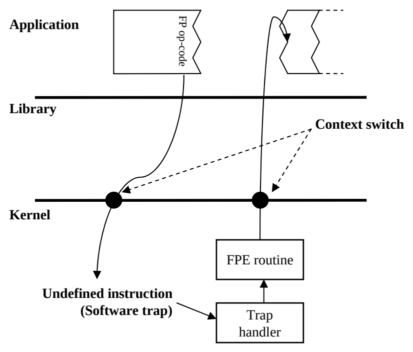


(a) Decoder

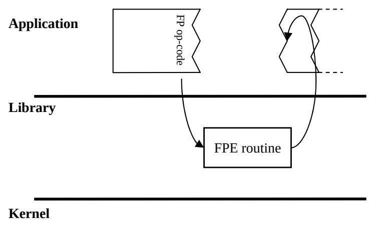


(b) Encoder

Figure 6: Run-time metrics apportioned to the kernel, the application, and library routines.



(a) Kernel FPE



(b) FPE library

Figure 7: Kernel FPE vs. FPE library.

As a result, we have observed that the OS has poor memory performance. These experimental results can be used to help the system designer understand the performance of the OS and the application within a system. Additionally, the interplay between hardware and software components in determining overall memory performance is also discussed in our study.

In future work, we will study the characteristic of operating systems and how they affect the OS subsystem and other services to applications, with the aim of alleviating the locality problem that operating systems typically exhibit.

7. REFERENCES

[1] http://cslab.snu.ac.kr/ sspark/papar/tr-osmp.pdf.

[2] http://www.adiengineering.com/productsBRH.html.

[3] Optimized MPEG-4 reference software contributed by nctu in taiwan. http://megaera.ee.nctu.edu.tw/mpeg/.

[4] ARM. Benchmarking with armulator. Application Note.

[5] K. Boland and A. Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, 14(4):59–67, 1994.

[6] H. Chen, K. Li, and B. Wei. Memory performance optimizations for real-time software HDTV decoding. In *Proc. IEEE International Conference on Multimedia and Expo (ICME'02)*, Lausanne, Switzerland, Aug. 2002.

[7] J. B. Chen and B. N. Bershad. The impact of operating system structure on memory system performance. In *Proc. ACM Symposium on Operating System Principles (SOSP'93)*, Asheville, NC, Dec. 1993.

[8] J. A. Darringer, R. Bergamaschi, S. Bhattacharya, D. Brand, A. Herkersdorf, J. Morell, I. I. Nair, P. Sagmeister, and Y. Shin. Early analysis tools for system-on-a-chip design. *IBM Journal of Research and Development*, 6(6):20–38, 2002.

[9] J. deBlanquier. Supporting new hardware environment with uclinux. *Journal of Linux Technology*, 1(3):20–28, 2000.

[10] K. Diefendorff and P. Dubey. How multimedia workloads will change processor design. *IEEE Computer*, 30(9):43–45, 1997.

[11] 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 *IEEE Annual Workshop on Workload Characterization*, Austin, TX, 2001.

[12] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1996.

[13] Intel. Intel 80200 processor based on intel xscale microarchitecture. Developer's Manual, Nov. 2000.

[14] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith. Application-driven synthsis of core-based systems. In *Proc. IEEE International Conference on Computer Aided Design (ICCAD'97)*, San Jose, California, USA, 1997.

[15] R. Klein, K. Travilla, and M. Lyons. Performance estimation of MPEG4 algorithms on arm based designs using co-verification. In *Proc. Embedded Systems Conference*, San Francisco, USA, 2002.

[16] R. Koenen. MPEG-4: Multimedia for our time. *IEEE Spectrum*, 36(2):26–34, 1999.

[17] S. A. McKee, Z. Fang, and M. Valero. An MPEG-4 performance study for non-simd,general purpose architectures. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'03)*, Austin, Texas, USA, Mar. 2003.

[18] N. T. Slingerland and A. J. Smith. Cache performance for multimedia applications. In *Proc. IEEE International Conference on Supercomputing*, Sorrento, Italy, June 2001.

[19] P. Soderquist and M. Leeser. Optimizing the data cache performance of a software MPEG-2 video decoder. In *Proc. ACM International Conference on Multimedia*, Seattle, Washington, USA, Nov. 1997.

[20] S. Sohoni, Z. Xu, R. Min, and Y. Hu. A study of memory system performance of multimedia applications. In *Proc. ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'01)*, Cambridge, Messachusetts, USA, June 2001.

[21] W. Wolf. *Computers as Components: Principles of Embedded Computing System Design*. Morgan Kaufmann, 2001.

[22] Z. Xu, S. Sohoni, R. Min, and Y. Hu. An analysis of the cache performance of multimedia applications. *IEEE Transactions on Computers*, 53(1):20–38, 2004.

[23] D. F. Zucker, M. J. Flynn, and R. B. Lee. A comparison of hardware prefetching techniques for multimedia benchmarks. In *Proc. IEEE International Conference on Multimedia Computing and Systems (ICMCS'95)*, Washington, USA, May 1995.