Interface Overheads in Embedded Multimedia Software

Tero Rintaluoma[1], Olli Silven[2], and Juuso Raekallio[1]

[1] Hantro Products Oy, Oulu, Finland
{Tero.Rintaluoma, Juuso.Raekallio}@hantro.com
[2] Department of Electrical and Information Engineering, University of Oulu, Finland
Olli.Silven@ee.oulu.fi

Abstract. The multimedia capabilities in battery powered mobile communication devices should be provided at high energy efficiency. Consequently, the hardware is usually implemented using low-power technology and the hardware architectures are optimized for embedded computing. Software architectures, on the other hand, are not embedded system specific, but closely resemble each other for any computing device. The popular architectural principle, software layering, is responsible for much of the overheads, and explains the stagnation of active usage times of mobile devices. In this paper, we consider the observed developments against the needs of multimedia applications in mobile communication devices and quantify the overheads in reference implementations.

1 Introduction

Current high-end mobile communication devices integrate wireless wide band data modems, video cameras, net browsers, and phones into small software controlled packages. The small size of the devices is a design constraint as the sustained heat dissipation should be kept low, and long untethered active usage times should be provided [1]. Their software systems must satisfy a multitude of requirements, resulting in a complex software solution that can only be implemented via concerted action of experts.

To facilitate this task most mobile communication device manufacturers have created common platforms for their product families and define application programming interfaces that remain the same across products, regardless of system enhancements and changes in hardware/software partitioning, including the number of processors used. Obviously, the software architectures and the components used need to be generic and reusable, but it is at the cost of efficiency. Consequently, middleware is widely applied in these systems as a key challenge is to enable uncomplicated integration of hardware and software components to the defined platform.

An exhibit of the undesired side-effects of this development is the stagnation of the talk-times of the mobile phones to around the 3h level, although the basic application has not changed in an essential manner [2]. The reasons have been traced to increased software architecture and interface overheads. In multimedia applications the overheads can be expected to be even more significant due to the need to support numerous standards, such as JPEG, H.264, MPEG-4 and VC-1, in the same execution environment. To provide control over these alternatives, more software layers are needed on top of them, adding to the number of instructions to be executed. The number of instructions executed matters, because the relative energy per instruction of embedded processor

S. Vassiliadis et al. (Eds.): SAMOS 2006, LNCS 4017, pp. 5–14, 2006.

Table 1. Energy efficiencies and silicon areas of ARM processors [3]

Processor	Max. clock frequency (MHz)	Power consumption (mW/MHz)	Silicon area (mm2)
ARM7 (720T)	100	0.2	2.4
ARM9 (926EJ-S)	266	0.45	4.5
ARM10 (1022E)	325	0.6	6.9
ARM11 (1136J-S)	550	0.8	5.55

Table 2. Cycle and instruction counts of software based MPEG-4 decoders end encoders (VGA 30frames/s, 512kbit/s)

Processor	Core/Bus CLK	decoder		encoder	
		MIPS	Mcycles/s	MIPS	Mcycles/s
ARM7 (720T)	2/1	129,7	303.9	646,8	1446,3
ARM9 (926EJ-S)	2/1	129,2	211,9	638,2	948,5
ARM10 (1022E)	3/1	129,2	151,5	638,2	722,9
ARM11 (1136J-S)	3/1	99,2	147,3	570,6	740,1

architectures has grown during the last few years. Table 1 shows the characteristics of ARM processors implemented using a 130nm CMOS process. Obviously, the advances at silicon level have been swallowed by the solutions that enable higher clock rates.

In Table 2 we illuminate the impact of architectural improvements at application level by comparing the instruction and cycle counts of software based MPEG-4 decoders. The results come from simulations made using the RVDS 2.2 tool [4] and assuming 0-wait-state memory accesses. The results for ARM11 are not completely cycle accurate. We notice that the number of instructions to be fetched and executed is slightly reduced between ARM7 and ARM10, indicating moderate instruction set improvements. Clearly, the real performance increases have come from higher clock rates.

Consequently, a multiprocessor based on lower performance processors could be more energy efficient than a single processor solution. However, larger silicon area adds to the cost, and the accompanied increasing leakage currents add to static energy consumption. We may also ask, whether a multiprocessor solution with middleware is really more energy efficient than a conceptually simpler single processor system.

Multitasking, APIs and middleware have big impacts on system performance due to cache effects and the execution of instructions needed by the interface mechanisms. Based on overhead measurements by Mogul and Borg [5] in 1991 and Sebek [6] in 2002 the context switch latencies appear to have remained the same for more than a decade despite processors becoming much faster. This is explained by the low cache hit ratios during the context switches.

Park et al (2004) measured the operating system effects on the performance of a MPEG-4 codec run as a single task on an ARM926 processor with embedded Linux. With this operating system the encoder run 20% and the decoder 27% slower [7]. Again, the cache effects were pinpointed as the key reason for the slowdown. Using a Linux platform, Verhoeven et al (2001) found that the performance of different middleware solutions varied between 260 and 7500 calls per second [8].

Based on our findings presented in the following, middleware layers in embedded system software may increase the overheads in a very significant manner. A contributing factor is the constantly increasing number of abstraction layers between software platform generations. As a result, monolithic hardware accelerators even in computing intensive multimedia processing are very attractive due to their low internal overheads.

2 Mobile Video Codecs and System Platforms

Typical mobile video codecs are currently built to adhere to MPEG-4 and H.264 standards. Both encoders and decoders consist of 10-20 algorithms that are in total invoked around 1-2 million times each second for a VGA sequence, making the overheads of the invocation mechanisms important, regardless of whether the implementation is in the software or hardware.

Table 3 shows the typical overheads of interface mechanisms as ARM11 processor cycles on a Symbian operating system. Due to the interrupt latency it is obvious, why the commercially available implementations are either pure software or monolithic hardware accelerators, that interrupt the control processor, for example, once for each frame. Fine grained hardware accelerators would be an inefficient approach due to the high software overheads from interrupt based hardware/software interfacing. Middleware as an interfacing mechanism must be exploited sparingly, limiting its use to rare long latency services. In general, the total architectural overhead costs are unknown and hidden in the application performance.

Table 3. Typical software interface costs in an embedded system environment (Symbian 9)

Mechanism	Overhead/cycles
Procedure call	3-7
System call (user-kernel)	1000-2500
Interrupt latency	300-600
Context switch	400
Middleware	60000

2.1 MPEG-4 Software Decoder

Figure 1 shows the rough organization of a software based MPEG-4 decoder [9] that consists of layers that each provide decoding functions for the upper layer. This is the structure designed already into the standards. The sequence layer is executed once for each frame or video packet, and extracts information on the employed coding tools and parameters from the input stream. The macro-block layer in turn controls the block layer decoding functions that have been designed to ensure the locality of addressing. For a VGA bit stream, the macro-block layer is invoked at most 1200 times per frame, while the block layer is run at most 7200 times.

Table 4 demonstrates the costs of software interfaces, when the APIs enabling reusability of functionalities are placed on the sequence, macroblock and block layers, and the assumed call overhead is 7 cycles. The figures do not contain the costs of

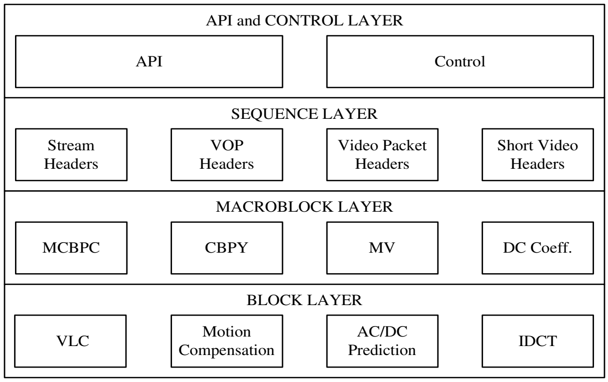

Fig. 1. Layered software architecture of a MPEG-4 video decoder

Table 4. The internal overhead share and energy costs of an MPEG-4 decoder with three API layer options (VGA 30 frames/s, 512kb/s, ARM926EJ-S implemented at 130nm CMOS)

APIs	Overhead (cycles/s / $\sim$ MHz)	Energy consumption (mW)
Sequence layer only	1806 / $\sim$ 0	0
Sequence and macroblock layers	2671599 / $\sim$ 2.7	1.2
Sequence, macroblock, and block layers	11376267 / $\sim$ 11.4	5.1

any functionality in the layers, and the experiments have been run without an operating system for maximum efficiency.

Based on the above, the internal overheads of the decoder on the ARM926 are about 5.4% of the total decoder cycles given in Table 2. Energywise they cost about the same as a hardware implementation of the MPEG-4 decoder using the same silicon technology. The control code in the sequence layer consumes additionally about 1.5 MHz, while the share of control load elsewhere in the code is difficult to quantify.

The MPEG-4 software encoder and decoder codes do not fit in typical 16-32 kbyte instruction caches. With an operating system, based on Park et al. [7], we should reserve at least 20% of the processor cycles to cache related overheads alone.

2.2 MPEG-4 Hardware Decoder

The monolithic hardware decoder API is almost identical to the above software decoder implementation [10]. Internally only a part of the sequence layer is implemented in the software and already the bit-oriented stream parsing is in the hardware for the sake of efficiency. The hardware interrupts the CPU after decoding each frame, on average 30 times per second. The sequence layer control software requires about 1 MHz, which is somewhat less than with the software implementation (1.5MHz). The internal organization of the accelerator is again as instructed by the MPEG-4 standard.

2.3 Multimedia Software Frameworks

Mobile multimedia software frameworks are defined software architectures, including APIs and middleware, intended to standardize the integration of software and hardware based video coding solutions into embedded devices. In addition, the goal is to provide mechanisms that enable building multimedia applications that are portable between platform generations.

The Symbian Multimedia Framework (MMF, Figure 2) is a multithreaded approach for handling multimedia data, and provides audio and video streaming functionalities. Regardless of whether the codecs are implemented in software or hardware, they are interfaced as plugins to the Multimedia Device Framework (MDF). With actual hardware codecs the plugins hide the vendor specific device drivers.

MDF with plugins is middleware that can be used to hide the underlying possible distributed implementation, for example, a decoder plugin may hide a decoder running on a Texas Instruments DSP processor behind an XDAIS interface. The codec vendors implement the MDF plugins with specified interfaces, and the MMF controller plugins that take care of synchronization between audio and video [11], for example. The application builders use the Client API that handles requests such as *record*, *play*, *pause*. At minimum, these activations of requests go through five software interface layers before reaching the codec. The performance depends greatly on the vendor provided controller plugins.

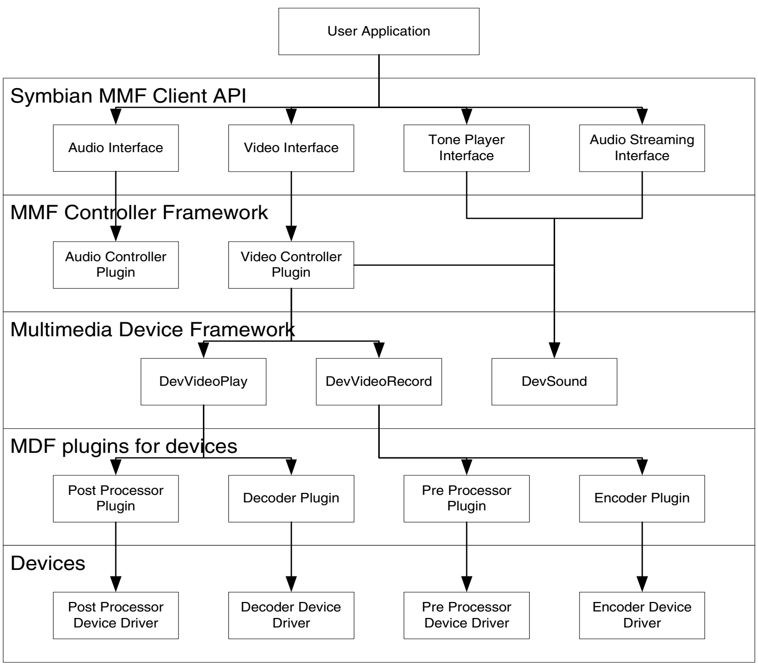

Fig. 2. Symbian Multimedia Framework

Table 5. The costs of multimedia APIs

	Decoder software interfaces		
	Proprietary API	Symbian MMF	Difference
Total cycles	220890240	225165845	2.14 MHz
D-cache misses	1599992	1633425	33433
I-cache misses	250821	322635	71813
D-cache hit ratio	94.7%	94.6%	0.1%
I-cache hit ratio	99.7%	99.6%	0.1%

In the Symbian operating system version 7 of 2003 the MDF was the whole framework, and that increased with two new abstraction layers, Client API and Controller Framework, in version 9 released in 2005. We are probably safe assuming additional layers in the future to support more versatile multimedia applications, based, for example, on the emerging MPEG-21 standard.

The proprietary solutions from mobile video codec manufacturers approach the portability issue from a different angle. For instance, in [10] thin software wrapper layers are used to facilitate porting the hardware and software codecs to the *multimedia engines* that provides, for example, video recording and playback functionalities in a tightly integrated manner. Table 5 compares the costs of accessing the video decoder functionality directly via a proprietary API, and through the Symbian MDF level. These costs are approximately the same for both software and hardware decoders. In power consumption the difference between the multimedia frameworks would be around 1mW on the ARM926 processor of Table 1.

The above measurements were made by running an MPEG-4 software decoder without display post-processing and audio for a QVGA sequence (320x240 pixels, 30 frames/s). The experiments were made on an actual ARM11 platform without SIMD optimizations and with a system supporting a single video coding standard. With more codecs the overheads of using any of them are slightly higher, especially when middleware interfaces are employed.

The results also provide a ballpark estimate on operating system and memory related overheads. The decoding of a QVGA stream requires around 110MHz, while 0-wait-state simulations predict half of that.

3 Energy Efficiency

To understand the role of the software interfaces in the energy efficiency of multimedia, it is necessary to consider the characteristics of whole implementations. For this purpose we use commercial hardware and software implementations of MPEG-4 and H.264 VGA video codecs [12]. Table 6 shows the estimated power consumptions of hardware based codecs with their necessary control software (1MHz in all cases) on a proprietary API. The applications were run on an ARM9 processor, and a 130 nm low power 1V CMOS process is used for all hardware.

Due to the disparity between the algorithmic and computational complexities of H.264 and MPEG-4 codecs, their monolithic accelerators differ significantly by gate

Table 6. Gate counts and estimated power needs of 30 frames/s hardware codecs

	MPEG-4		H.264	
	kGates	Power (mW)	kGates	Power (mW)
Decoder	161	5.6	373	24.2
Encoder	170	9.6	491	33.4

Table 7. Power consumption estimates (mW) for software based MPEG-4 and H.264 decoders

	MPEG-4	H.264
ARM7 (720T)	64	140
ARM9 (926EJ-S)	96	232
ARM10 (1022E)	92	232
ARM11 (1136J-S)	118	348

counts and required silicon area. The above H.264 codec also supports MPEG-4 as that adds only a few percentage points to the total gate count. The hardware shares of the power consumption are almost independent of the bit rate that is an essential difference to software implementations.

Table 7 shows the approximate power consumptions for architecture optimized software implementations of MPEG-4 and H.264 decoders. The figures have been determined for 30 frames/s VGA 512kbit/s stream and the decoders are the only tasks being run on ARM processors implemented using a 1V low power 130nm CMOS process. The costs of system software interfaces and post-processing the video for display are not included. Based on these results, multiprocessor solutions can indeed provide energy efficiency benefits.

The Symbian MDF supports multiprocessing and adds approximately 1mW to the decoder power consumptions. This is not significant except with the MPEG-4 hardware decoder (18%). We also observe that the power consumption of the ARM11 implementation of the software decoder is roughly 20% more than with the ARM9, which may not justify the added complexity of a multiprocessor system.

Figure 3 compares the findings for both software and hardware decoders in terms of normalized silicon areas (Mpixels/s/mm2) and power efficiencies (Mpixels/s/W) of the

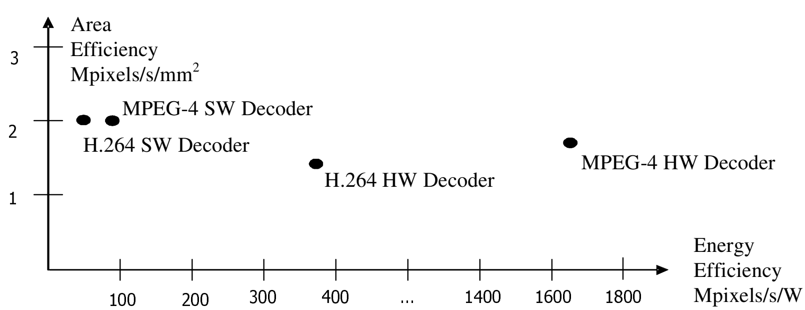

Fig. 3. Area and energy efficiencies of video decoder implementations

MPEG-4 and H.264 implementations. The gap between respective software and hardware implementations is striking, and there are no implementation options in between.

Returning to Table 4 that itemized the software function interface costs, we can estimate that an interrupt driven macroblock accelerator implementation would need around 50mW for software interfacing alone with an ARM926 (130nm CMOS). This eliminates most of the potential energy gains from hardware acceleration, and is not an attractive option.

4 Directions for Development

With an efficient software/hardware interfacing mechanism the energy overhead of fine grained hardware acceleration should not exceed that of a pure software implementation. We estimate that the lower bound power consumption (again 130nm CMOS and ARM926) for such a decoder would consist of 5.1mW from software interfaces and 5.6 mW from hardware accelerators and control software, totalling 10.7mW. Software implementation defines a 96mW upper bound, so the energy efficiency should fall midway between hardware and software implementations in Figure 3.

A model for the energy efficient approach can be obtained from periodically scheduled embedded real-time systems that run their tasks in a fixed order, and use hardware accelerators without interrupts relying on their deterministic latencies. Even some early GSM mobile phones employed this principle that in essence results in a multithreaded system [13]. In those implementations fixed hand made schedules could be used. However, video coding has data dependent control flows, so the scheduling of the threads and the allocation of hardware resources must be done dynamically. This can be performed, for instance, by using a Just-In-Time (JIT) compiler. Figure 4 below illustrates decoding an inter-macroblock using fine grained short latency hardware accelerators with a schedule created from the contents of the video bit stream.

The accelerators, color conversion for display, inverse quantizer (IQ)+IDCT, and bilinear interpolator, have deterministic latencies, and the software uses the results when they become available. Color conversion to display executes in hardware simultaneously with sequence and macro-block layer decoding. The threads alternate between software and hardware execution without an interrupt based synchronization overhead.

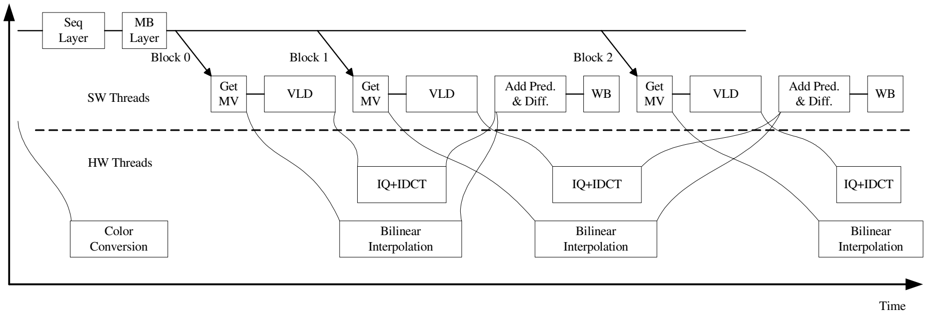

Fig. 4. Multithreaded decoding of an inter-macroblock from a coded video bit stream

To implement the hardware/software multithreading applications, efficient means for generating the schedules are needed. One option is to employ a set of fixed schedules to choose from based on the task at hand, while JIT compilers providefor more flexibility, although at the cost of higher overheads. Such compilation techniques could perhaps reduce the number of defined layers in software architectures, in turn providing compensationary savings. From the system developer's point of view fine grained accelerators cut the design verification time and provide faster time-to-market capability. The enabling missing elements appear to be on the side of software technology.

5 Summary

In computing intensive software applications, such as video coding, the interface overheads are in principle only a small portion of the total processor cycles. Much of the overheads originate from the layered software architecture style, and are amplified by cache related phenomena due to the decreased locality of code execution and data accesses. The operating systems have similar effects on the performance. In total, the overheads can exceed the number of cycles needed by the actual application.

When improved energy efficiency is targeted by the utilization of hardware accelerators, the software overheads may play a very significant role. Based on our experience, even the multimedia framework software interfaces may demand more processor cycles than the actual control of a hardware accelerator.

The efficiency of software/hardware interfaces is becoming a critical issue, because of the increasing leakage currents of silicon implementations. This makes run-time silicon re-use, for example, via fine grained acceleration very attractive. The conventional interrupt driven approach for hardware/software interfacing results in high overheads, in fact, much higher than in pure software implementations. If fine grained hardware accelerators could be interfaced to software at the cost of software functions, flexible energy efficient solutions could be implemented.

Current comparable MPEG-4 decoder implementations in hardware and software (ARM11) need 5.6mW and 118mW of power, respectively, without operating system and cache overheads that range between 20% and 100%. Solutions that fall between these figures are needed. The proposed simultaneous hardware/software multithreading is a possible option that is under investigation.

Multiprocessor implementations offer 20-50% improved energy efficiency in video coding when older processor architectures are used instead of the most recent ones. However, for the same performance, twice the silicon area is needed, resulting in higher static power consumption due to leakage currents. Furthermore, interprocessor communications can add significant overhead that falls in to the range of middleware costs.

Acknowledgements

Numerous people have contributed to this paper by providing their comments, questions and technical expertise. In particular, we wish to thank Mr. Jani Huoponen and Mr. Jarkko Nisula from Hantro Products, and Mr. Kari Jyrkkä from the Nokia Corporation.

References

1. Neuvo, Y.: Cellular phones as embedded systems. In: Solid-State Circuits Conference. Volume 1. (2004) 32–37
2. Silven, O., Jyrkkä, K.: Observations on power-efficiency trends in mobile communication devices. In: Proc. 5^th Int. Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, LNCS 3553 (2005) 142–151
3. ARM: Processor core overview. In: www.arm.com/products/CPUs. (2005)
4. ARM: RealView Developer Suite. In: www.arm.com/. (2005)
5. Mogul, J., Borg, A.: The effect of context switches on cache performance. In: ASPLOS-IV, Santa Clara, ACM (1991) 75–84
6. Sebek, F.: Instruction cache memory issues in real-time systems. Master's thesis, Department of Computer Science and Engineering, Mälardalen University, Västerås, Sweden (2002)
7. S. Park, Y.L., Shin, H.: An experimental analysis of the effect of the operating system on memory performance in embedded multimedia computing. In: EMSOFT-04. (2004) 26–33
8. P.H.F.M. Verhoeven, J.H., Lukkien, J.: Network middleware and mobility. In: PROGRESS workshop. (2001)
9. Hantro: 4100 MPEG-4 / H.263 Software Decoder. In: www.hantro.com. (2006)
10. Hantro: 8300 Multimedia Application Development Platform. In: www.hantro.com. (2006)
11. Symbian: Introduction to the ECOM Architecture. In: http://www.symbian.com/. (2006)
12. Hantro: Hardware and Software Video Codec IP. In: www.hantro.com. (2006)
13. Jyrkkä, K., Silven, O., Ali-Yrkkö, O., Heidari, R., Berg, H.: Component-based development of DSP software for mobile communication terminals. Microprocessors and Microsystems **26** (2002) 463–474