Copyright Notice

The document is provided by the contributing author(s) as a means to ensure timely dissemination of scholarly and technical work on a non-commercial basis. This is the author's version of the work. The final version can be found on the publisher's webpage.

This document is made available only for personal use and must abide to copyrights of the publisher. Permission to make digital or hard copies of part or all of these works for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. This works may not be reposted without the explicit permission of the copyright holder.

Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the corresponding copyright holders. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each copyright holder.

IEEE papers: © IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The final publication is available at http://ieeexplore.ieee.org

ACM papers: © ACM. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The final publication is available at http://dl.acm.org/

Springer papers: © Springer. Pre-prints are provided only for personal use. The final publication is available at link.springer.com

# Interrupt Costs in Embedded System with Short Latency Hardware Accelerators

Sébastien Lafond
Turku Centre for Computer Science, Embedded Systems Laboratory
*sebastien.lafond@abo.fi*

Johan Lilius
Åbo Akademi University, Centre for Reliable Software Technology
*johan.lilius@abo.fi*

## Abstract

*The current trend in handheld devices is to provide users with various embedded multimedia applications. Architecture developers have to use dedicated hardware accelerators to meet the timing requirements of these new applications. For physical and economical reasons the use of dedicated monolithic hardware accelerators is impractical. Instead, because the multimedia applications share common functionalities, monolithic hardware accelerators can be split into smaller accelerators to remove redundancy and save on silicon area. Unfortunately, lowering the granularity of accelerators increases synchronization calls between the main processor and the accelerators.*

*This paper presents a methodology for analyzing the impact of short latency hardware accelerators on a typical embedded system. We show that hardware accelerator granularity has a direct effect on system performance in terms of cache misses, execution time and thus energy consumption.*

## 1. Introduction

Handheld devices integrate more and more functionality, and providing more multimedia applications is becoming a de facto requirement. Solutions are therefore needed for accelerating these computationally intensive applications in order to fulfill the requirements. The main acceleration approaches can be classified into two categories [8]:

1. A short portion of code is accelerated by extending the processor instruction set with a corresponding instruction. In this case the new instruction has a typical execution latency from 1 to 4 cycles, thus limiting the size of the accelerated software. Developing longer instruction would make the pipeline execution flow inefficient.
2. A full application functionality is accelerated with a monolithic hardware accelerator used as peripheral device. The hardware accelerator is then synchronized with the application by the means of interrupts. In this case the hardware accelerator has a typical execution latency from several thousand cycles up to several hundreds of thousands of cycles. However dedicated monolithic hardware accelerators are onerous to achieve due to physical and economical constraints.

The use of fine grained hardware accelerators has the advantage of saving silicon area by allowing collaborative use of common accelerated functionalities among several applications, thus cutting down implementation redundancy over several accelerators. For example, applications using reconfigurable media coding (RMC) [3] [2], where arbitrary combinations of algorithms may be assembled without predefined standardization, could easily take advantage of collaborative use of common accelerated functionalities. Also one could accelerate the time consuming DCT function in a MPEG4 video decoder and share the created accelerator with a JPEG decoder application. In such a case an access management system or dedicated scheduler is needed in order to avoid blocking state when two tasks would request the use of an accelerator at the same time. The study of such access management system or dedicated scheduler is however beyond the scope of this paper. This would restrict our study to a specific set of applications while we are here exclusively interested in analyzing the impact of short latency hardware accelerators on a typical embedded system.

Splitting a monolithic hardware accelerator into several fine grained hardware accelerators can also in some cases permit a pipelined execution of the accelerators. Nevertheless, it transfers control complexity to the software running on the processor and as a consequence increases the synchronization frequency between the accelerators and the processor. Synchronization between an accelerator and the processor is needed to inform the processor about execution

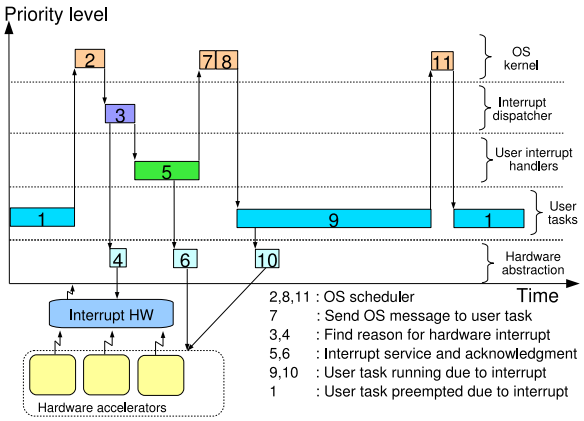


**Figure 1. Execution sequence for accelerator synchronization [9]**

termination of the accelerator. As a result, the use of short latency hardware accelerators tends to amplify the interface cost used for synchronization between the processor and the hardware accelerators.

A hardware accelerator is typically used as a peripheral device and synchronized with the OS running on the processor with an interrupt [9]. Figure 1 shows the sequence of operations needed in a multitasking environment when an interrupt instructs the OS about the termination of an accelerator task. Task 9 previously called a hardware accelerator. After termination of the accelerator execution an interrupt is triggered by the accelerator which will wake up the calling task 9 in order to fetch the computed results.

In a single tasking environment this extra synchronization cost is limited to the execution of the interrupt mechanism and handler as no scheduler is needed. But in a multitasking environment the extrinsic (intra-task) cache behavior will be affected by the synchronization mechanism. Each time an accelerator is called the content of the cache is changed by the new scheduled task running during the accelerator execution. This will result in a performance lost called the *cache refill penalty* [7] [6] each time an accelerator is called and an interrupt is triggered. The cache refill penalty is due to an increase of cache misses each time a context switch is performed. It introduces an increase in execution cycles and energy consumption since a cache miss leads to more bus and main memory activity. The cache refill penalty could be reduced by using various cache partitioning approaches where the cache is logically divided into multiple partitions and each partition is exclusively accessed by a single task [12] [5]. However such partitioning techniques are relevant only in the case the number of tasks is fixed and completely defined for the whole system life time. In the case of reconfigurable media coding applications, where arbitrary combinations of algorithms may be used, cache partitioning approaches would require one cache partition for each algorithm combination. This would request an unreasonable total cache size.

Moreover, since the pace of instruction execution speeds up much faster than main memory access time, the cache refill penalty has increased and will in the future continue to increase along with the difference between processor and memory speed.

The major contribution of this paper is the establishment of a simulation framework showing the overall cost due to interrupts used for synchronization between the hardware accelerators and the processor on a typical embedded system. This overall cost is composed by (a) a direct cost due to the use of hardware accelerators as peripheral devices and (b) indirect cost due to the cache refill penalty.

The methodology presented in this paper can be re-used with other platform configuration for evaluating the granularity range of new hardware accelerators which will provide a good trade off between implementation redundancy and synchronization cost.

The rest of this paper is organized as follow: In Section 2 we present the simulation framework established for this study. Section 3 gives the simulation parameters used to run the simulation framework, section 4 evaluates our results and section 5 concludes the paper.

## 2. Simulation framework

The simulation framework presented in this section models a typical handheld device featuring basic multimedia applications. It includes a hardware platform, an operating system and a set of applications and hardware accelerators.

The Sim-Panalyzer [10] processor simulator is used for this study. Sim-Panalyzer is based on the SimpleScalar [1] processor simulator and performs cycle accurate simulation of a strongARM SA-1100 processor. It computes at every simulated cycle the energy consumption of each module constituting the ARM core (clock, alu, cache, etc.). Such processor simulator permits the execution of an operating system ported on ARM architecture.

As RTEMS 4.6.2 has been ported onto SimpleScalar by Jack Whitham [11], RTEMS was chosen as the real-time operating system for this study. This port includes a SimpleScalar extension for supporting an interrupt based programmable timer which is needed by RTEMS. RTEMS is a free open source real-time operating system designed for embedded systems and supporting a variety of application programming interfaces (APIs) and interface standards. This real time operating system allows us to execute a set of applications as independent tasks in a pre-emptive multitasking environment, a prerequisite for our simulation.

**Table 1. Selected functions to be accelerated**

| Application | Chosen functions | Nb of calls |
|---|---|---|
| GSM coder | APCM_quantization() | 532 |
| GSM decoder | GSM_RPE_Decoding() | 532 |
| JPEG comp. | forward_DCT() | 128 |
| JPEG decomp. | h2v2_fancy_upsample() | 512 |

A set of 4 applications are chosen from the MiBench benchmark suite [4]. These applications are present on typical handheld devices: a GSM audio coder, a GSM audio decoder, a JPEG compressor and a JPEG decompressor. For each application an execution time profiling was carried out in order to identify the most time consuming functions. Out of this profiling some functions were selected to be executed on dedicated hardware accelerators. Table 1 shows the selected functions and the number of times the functions are called. Each application is implemented as a task running on the OS. An idle task with low priority is also implemented and is executed in the case all other tasks are waiting for their hardware accelerators to terminate.

The presented applications and hardware accelerators define our reference environment. In addition to this reference environment a fifth application was implemented. This last application will be called the *exploration application*, and will be used to explore the impact of a short latency hardware accelerator synchronized by interrupts on the overall performance of the system. The exploration application can be seen as an added task disturbing the reference environment.

Figure 2 represents the parameters influencing the execution pattern of the exploration application in a single task environment. Executed in the pre-emptive multitasking environment of our simulation framework, the OS will schedule other tasks to run during the suspended state of the exploration application. The exploration application is the task used for measuring the cost of short latency hardware accelerators. One hardware accelerator with variable latency is associated with the exploration application. Thus the simulation framework requires two parameters: (a) the length in cycles of execution performed before a call to the hardware accelerator is done (see Figure 2) and (b) the latency in cycles of its associated accelerator. When the accelerator execution terminates the exploration application will be scheduled by the OS to run depending on its priority and the priority of other tasks.

The complete simulation framework now consists of five tasks and their respective hardware accelerators. Figure 3 shows the system architecture used for this study. The five tasks running on the RTEMS operating system need to communicate with their corresponding hardware accelerators. For each accelerator a new system call is assigned and Sim-Panalyzer is modified to catch these five new system calls.

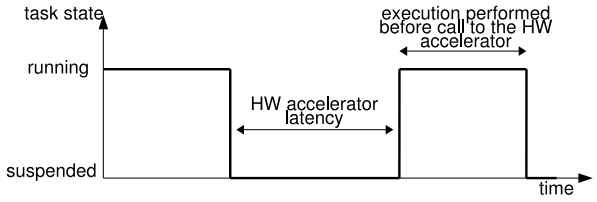


**Figure 2. The exploration application in single task environment**

Figure 4 represents the sequence of executed operations following a hardware accelerator call. These operations are explained as the following:

1. Sim-Panalyzer reads the possible parameters from defined registers and executes the hardware accelerator job. Then it writes the possible results on defined registers.
2. Sim-Panalyzer sets the corresponding interrupt flag valid in $X$ cycles, $X$ being the accelerator latency.
3. RTEMS suspends the calling task by changing its priority to a low level, making the task non-executable.
4. RTEMS schedules the remained non-suspended tasks to run.
5. The interrupt handler will acknowledge the triggered interrupt and call the OS to resume the corresponding task by restoring its previous priority.
6. Sim-Panalyzer gets the interrupt acknowledgement and clears the associated flags.
7. RTEMS schedules all non-suspended tasks to run.

It is important to note that the hardware accelerator jobs are executed within the Sim-Panalyzer simulator, which means that their executions are performed outside the simulated platform. The hardware accelerator execution costs, including possible data transfer between the processor and accelerators, are thus not taken into account in this study. This omission does not affect the measurements because the cost due to the use of interrupts and the indirect cache refill

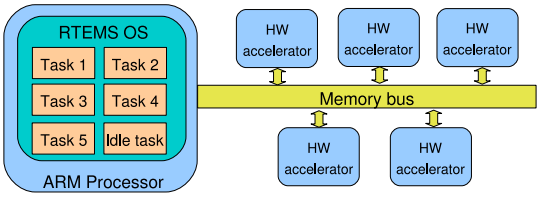


**Figure 3. System architecture**

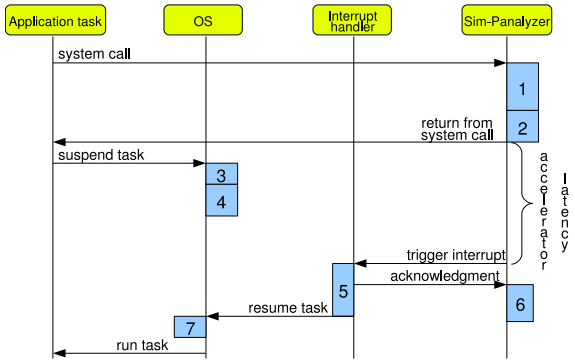


**Figure 4. Sequence of operations**

penalty cost are not affected by the internal accelerator activities or read/write operations initiated by the accelerator.

## 3. Simulation parameters

The simulation parameters bind the simulation framework within a defined execution window. This section defines the constant and variable parameters and their corresponding values used in the simulation framework.

### 3.1 Sim-Panalyzer

Sim-Panalyzer defines the processor parameters and the configuration of the caches. For this study the processor speed was set at 233 MHz. The configuration for the level 1 instruction cache, level 1 data cache and the unified level 2 cache is presented on table 2. Table 3 shows the different latencies for each memory level. This configuration tries to target an average embedded system performance that could be used for a multimedia handheld device. The relatively small level 1 and 2 caches compensate for the relatively small footprint of the benchmark applications (see *Applications and HW accelerators* subsection). All other parameters used by Sim-Panalyzer were set to their default values.

### 3.2 RTEMS

RTEMS has a few parameters influencing the timing behavior of the executed tasks. For this study all tasks, except the idle task, have their priority levels set to 10. The idle

**Table 2. Caches configuration**

| Caches | Associativity | Size | Nb blocks | Block Size |
|---|---|---|---|---|
| il1 | direct mapped | 4 Kb | 128 | 32 bytes |
| dl1 | direct mapped | 4 Kb | 128 | 32 bytes |
| ul2 | 4-way | 8 Kb | 256 | 32 bytes |

**Table 3. Memory Latencies with a clock frequency of 233Mhz**

| | il1 | dl1 | ul2 | main memory *first chunk access* | main memory *inter chunk access* |
|---|---|---|---|---|---|
| Latency in cycles | 2 | 2 | 6 | 30 | 4 |

task has a priority of 20. Following a call to a hardware accelerator the task priority level is changed to 250, making the task un-executable by the RTEMS scheduler. The interrupt handler will restore the task priority level to 10 when the corresponding interrupt is triggered. The preemptive execution is activated and the time slice is set to 5 ticks, one tick representing one hundredth of a second. In absence of hardware accelerator interrupt, the scheduler is then set to run 20 times per second.

### 3.3 Applications and HW accelerators

The hardware accelerators used by the applications defining the reference environment have fixed execution latencies and table 4 shows their latency values in cycles. As input data, the jpeg compression and decompression application process a 512 by 512 pixels image, and the GSM encoder and decoder process a 2 seconds 8-bit audio signal. The executable file containing the 5 applications and the idle task consists of 370kB of instructions and 17kB of data.

**Table 4. Hardware accelerator latencies**

| Application | Accelerator latency |
|---|---|
| GSM coder | 2500 cycles |
| GSM decoder | 2500 cycles |
| JPEG comp. | 3000 cycles |
| JPEG decomp. | 2500 cycles |

The exploration application implements an empty loop in ARM assembly code followed by a system call to its hardware accelerator.

With such rudimentary implementation the exploration application is used to look into the effect of hardware accelerator granularity on the overall system running several other applications. This implementation has a very small instruction and data footprint in order to interfere as little as possible with the cache behavior. At first only the impact of having different hardware granularities is studied. The hardware granularity is adjusted by dividing at the same time the loop length and the accelerator latency by a multiple of 2. Splitting the hardware accelerator into 2 independent smaller accelerators is thus simulated by dividing

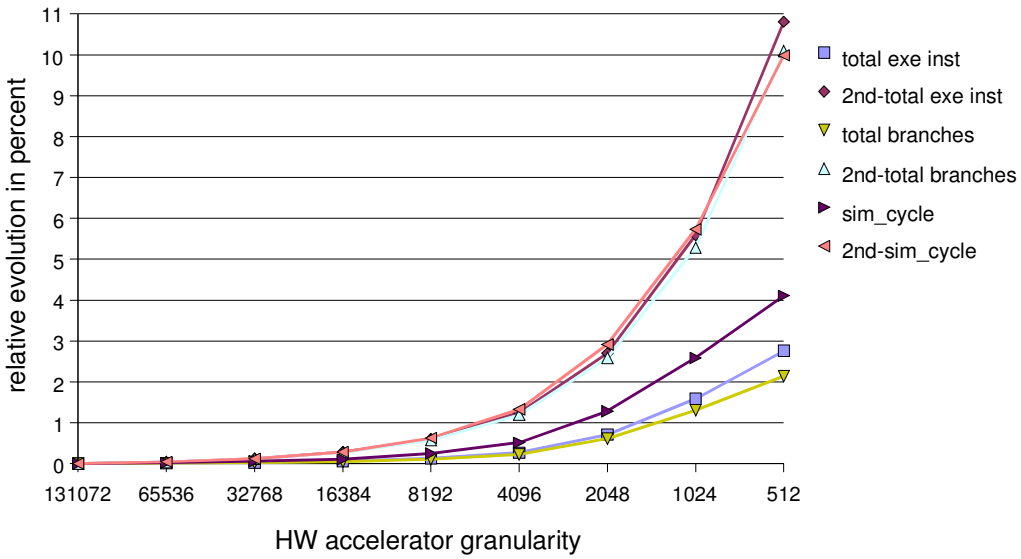


**Figure 5. Influence of granularity on execution time**

the exploration application loop length and its accelerator latency by 2.

A second measurement series is performed with the exploration application having data accesses after the loop. Data accesses are obtained by copying in an array of 200 integers the $n^{th}$ array value into the *n-1* location. Adding data accesses in the exploration application is done in order to obtain a more realistic application behavior.

## 4. Results

We run the simulation framework with different granularities for the hardware accelerators of the exploration application. For each granularity the loop length of the exploration application is set to the accelerator latency value. For all measurements nine different granularities are used: from the coarse-grained system having 10 calls to a 131072 cycles hardware accelerator latency to the fine-grained system calling 2560 times a hardware accelerator with a latency of 512 cycles. We express the granularity in term of cycles: a granularity of 131072 represents a system having one hardware accelerator with a latency of 131072 cycles and called in our experiment 10 times. In the same way, a granularity of 65536 represents a system with one hardware accelerator having a latency of 65536 cycles and called 20 times. In our experiment a granularity of 65536 is then equivalent to a system with two hardware accelerators being called 10 times and both having a 32768 cycles latency, introducing a split coefficient of 2.

As we can see from Figure 4, if the latency is too short the accelerator will trigger an interrupt before the RTOS finished to suspend the calling task, which will result in deadlocking the calling task. In our simulation framework the fastest accelerator we are able to simulate is an accelerator with a 512 cycles latency. This indicates the mechanism for lowering a task priority in RTEMS (sequence 3 on Figure 4) takes less than 512 cycles.

All results presented in this section are relative measurements using the values obtained for the coarse-grained system as reference. In other words, all figures trace the relative evolution in percent of the measured elements compared to the values obtained for the system having a granularity of 131072. The figure legends indicate the results for the second measurement series with the term *2nd*.

Figure 5 presents for the two exploration applications the evolutions of the number of executed instructions, taken branches and total execution time in cycles for all the tasks running on the system. For the two exploration applications the three measurements show an exponential increase when the hardware granularity is reduced. For the exploration application having data accesses the total execution time increases by about 11% with the finest grained accelerator compared to the coarse-grained implementation. This is about twice of the increase obtained by the rudimentary exploration application. This difference can be explained by an increase in data cache misses, being presented afterward, due to the extra data accesses.

Figure 6 shows for the two exploration applications the granularity influence on level 1 instruction cache. On Figure 6 *il1.pdissipation* represents the total energy dissipated by the cache. The two exploration applications present similar results concerning the number of cache misses, with an increase of almost 25% when fine grained accelerators are used. This is explained by the fact that the two ex-

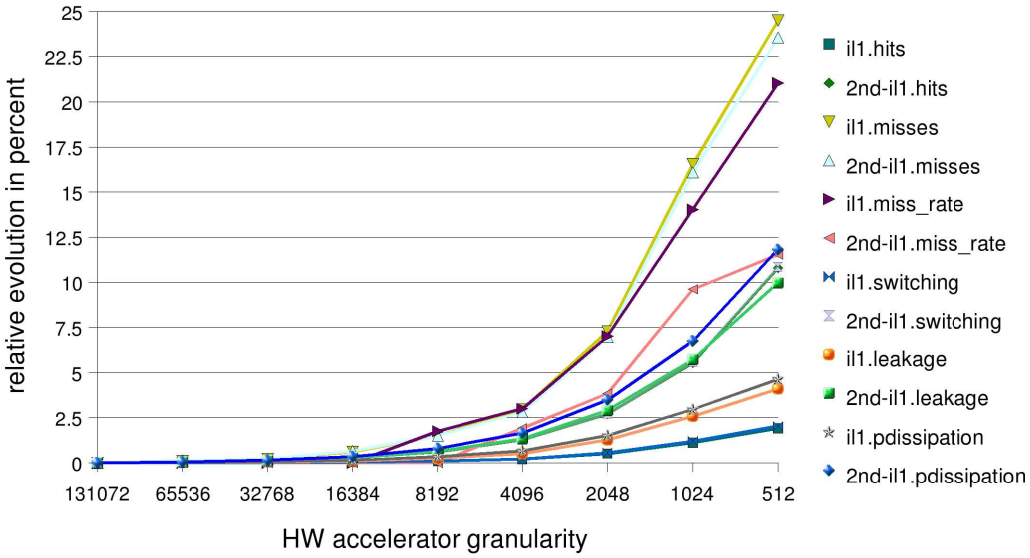


**Figure 6. Influence of granularity on level 1 instruction cache**

ploration applications have little difference in term of instruction footprint. Indeed only very few instructions were added in order to implement extra data accesses. However the number of cache hits is bigger with the second exploration application due to the added data accesses. Thus, as the number of cache misses is equivalent for the two exploration applications, the cache miss rate for the second exploration application is increasing at a slower rate. The leakage current is a static cost in a memory and its resulting energy consumption is only dependent on the time the memory is in use. This explains why the energy, *il1.leakage*, wasted due to leakage current inside the cache is increasing at the same rate than the increase of the total execution time. On the other hand the energy consumption due to switching activities in a memory is dependent on the number of occurring read and write accesses. As a general comment, one can say that decreasing the hardware granularity will have a relatively large impact on the level 1 instruction cache miss rate with a 25% increase for the finest grained accelerator compared to the coarse-grained implementation.

The granularity influence on the level 1 data cache for the two exploration applications is presented on Figure 7. Adding data accesses to the exploration application results in a roughly 30% cost increase compared to the rudimentary exploration application implementation. As the number of misses and hits increases in a similar proportion in the level 1 data caches, the value for the cache miss rate stays equal for the two exploration applications. Figure 7 shows that for the finest grained accelerator we obtain a 20% cache miss and 10% energy consumption increase in the level 1 data cache for the second exploration application. As a general comment we observe that decreasing the hardware granularity will primarily introduce an increase in level 1 data misses, thus driving an increase of the cache energy consumption.

Figure 8 shows for the two exploration applications the influence of granularity on the unified level 2 cache. An interesting observation is the drop of the unified level 2 cache miss rate with the reduction of hardware granularity. The miss rate decrease is due to a fast increase of the number of cache hits while the number of cache misses stays almost constant. For the two exploration applications, reducing the hardware granularity increases significantly the number of cache hits which raises the energy consumption because of the increase in switching activity.

As a general comment we can say that decreasing the hardware accelerator granularity on a typical embedded system relying on interrupt mechanism for synchronization between the accelerators and the processor will slow down the system execution time not only by increasing the number of instructions to execute but also by increasing the level one and two cache hits and misses. On the simulated system, cost increases start to be seen from the granularity level 4096 which correspond to splitting the reference hardware accelerator into 32 separate smaller accelerators.

### 4.1 Implication of the results

For the two exploration applications we also measured the average time needed to perform the synchronization between the accelerators and the processor. For the simple exploration applications each synchronization costs on average an extra 1375 cycles, with a standard deviation of 365 cycles, while for the second exploration application it costs

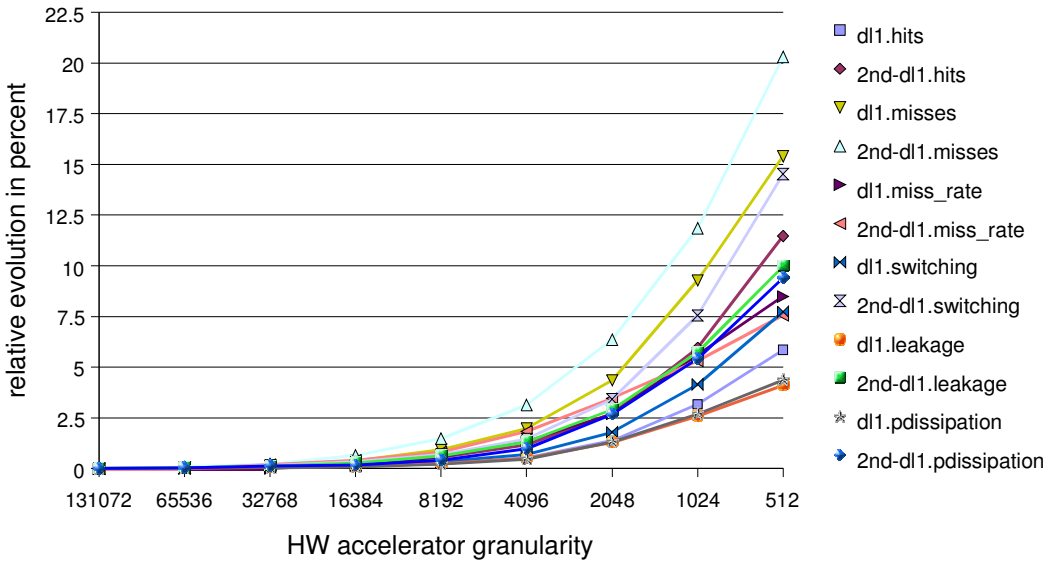


**Figure 7. Influence of granularity on level 1 data cache**

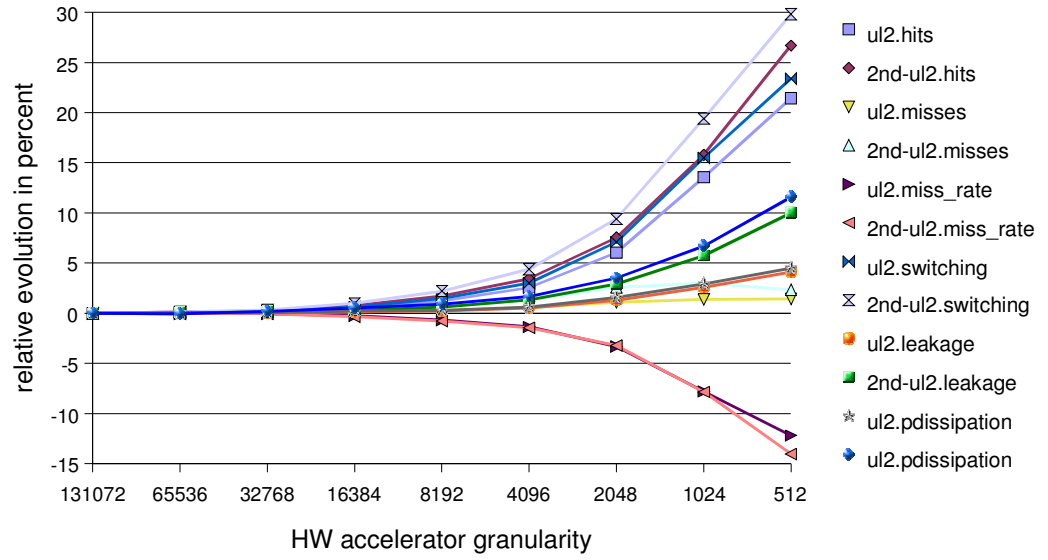


**Figure 8. Influence of granularity on level 2 unified cache**

on average an extra 3250 cycles with a standard deviation of 315 cycles. The cost difference between the two exploration applications is explained by the increase of the cache refill penalty in the second exploration application.

If one extracts a discrete cosine transform out of a video decoder implementation and use it as a hardware accelerator decoding a QVGA (320x240 pixels) video at 25 frames per second, the hardware accelerator will be called 45 000 times per second. Assuming on average a total cost of 3250 cycles per call, 145 Millions processing cycles per second will be wasted due to the overhead introduced by the synchronization mechanism between the processor and hardware accelerator.

## 5. Conclusion

In this study we presented a methodology for analyzing the impact of short latency hardware accelerators on a typical embedded system. The presented methodology can be re-used with other platform configurations for evaluating the granularity range of new hardware accelerators which will provide a good trade off between implementation redundancy and synchronization cost.

We demonstrated that when approaching the bottom line imposed by the RTOS speed in the mechanism of suspending a task, decreasing the hardware accelerator granularity will introduce relatively important extra costs in terms of cache misses, execution time and energy consumption. Therefore using a hardware accelerator can become inefficient if the accelerator latency is too short. This is due to an increase of the number of instructions to execute and number of level one and two cache hits and misses. Reducing the OS mechanism speed used in synchronization between the accelerators and the processor will push down the minimum execution latency the accelerators can have, but will also considerably increase the system execution time and energy consumption. As a direct consequence, the additional cost due to the synchronization introduced by the fine grained hardware accelerator will be in some case preponderant on the gain obtained by the accelerated software.

If one needs to implement a frequently used hardware accelerator that has a short latency, an original "interrupt and context switch free" synchronization mechanism needs to be used in order to provide an efficient solution.

## 6. Acknowledgement

We are thankful to professor Olli Silvén for his helpful comments concerning this study.

## References

[1] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, 1997.

[2] C. L. et al. Reconfigurable media coding: a new specification model for multimedia coders. In *Proceeding of the IEEE 2007 Workshop on Signal Processing Systems (SiPS)*, 2007.

[3] J. T.-K. et al. Reconfigurable media coding: Self-describing multimedia bitstream. In *Proceeding of the IEEE 2007 Workshop on Signal Processing Systems (SiPS)*, 2007.

[4] M. R. G. et al. Mibench: A free, commercially representative embedded benchmark suite. In *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14, Washington, DC, USA, 2001. IEEE Computer Society.

[5] F. Mueller. Compiler support for software-based cache partitioning. In *LCTES '95: Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers & tools for real-time systems*, pages 125–133, New York, NY, USA, 1995. ACM.

[6] F. Sebek. The real cost of task pre-emptions — measuring real-time-related cache performance with a hw/sw hybrid technique. Technical report, Mälardalen Real-Time Research Centre, Department of Computer Science and Engineering, Mälardalen University, Sweden, Aug. 2002.

[7] F. Sebek and J. Gustafsson. Determining the worst case instruction cache miss-ratio. In *Proceedings of Workshop On Embedded System Codesign (ESCODES'02)*, San Jose, California, USA, 24, 2002.

[8] O. Silven and K. Jyrkkä. Observations on power-efficiency trends in mobile communication devices. *EURASIP Journal on Embedded Systems*, 2007.

[9] O. Silven, T. Rintaluoma, and K. Jyrkkä. Implementing energy efficient embedded multimedia. In *Multimedia on Mobile Devices II, Proceedings of the SPIE, Volume 6074*, 2006.

[10] Sim-Panalyzer. http://www.eecs.umich.edu/~panalyzer.

[11] J. Whitham. http://www.jwhitham.org.uk/simplescalar/.

[12] A. Wolfe. Software-based cache partitioning for real-time applications. In *Proceedings of the 3rd International Workshop on Responsive Computer Systems*, 1993.