Energy efficiency analysis of multi-stream MPEG-4 decoder systems

Sébastien Lafond[a] and Jani Boutellier[b] and Johan Lilius[a] and Olli Silvén[b]

[a]Åbo Akademi University, Joukahainengatan 3-5, Turku, Finland
[b]University of Oulu, PO BOX 4500, Oulu, Finland

ABSTRACT

This paper presents a comparison of two systems that can simultaneously decode multiple videos on a simple CPU and dedicated function-level hardware accelerators. The first system is implemented in a traditional way, such that the decoder instances access the accelerators concurrently without external coordination. The second system implementation coordinates the tasks' accelerator accesses by scheduling. The solutions are compared by execution cycles, energy consumption and cache hit ratios. In the traditional solution each decoder task continuously requests access to the needed hardware accelerators. However, since the other tasks are competing on the same resources, the tasks must often yield and wait for their turn, which reduces the energy-efficiency. The scheduling-based approach assumes that the accelerator latencies are deterministic and assigns time slots for accelerator accesses required by each task. The accelerator access schedule is re-designed for each macroblock at run-time, thus avoiding the over-allocation of resources and improving energy-efficiency. Deterministic accelerator latencies ensue that the CPU is not interrupted when an accelerator finishes. The contribution of this study is the comparison of the accelerator timing solution against the traditional approach.

Keywords: Video codecs, Parallel processing, Power demand

1. INTRODUCTION

Multimedia applications running on modern mobile devices require huge amounts of computational resources. Performing all the required computations in software is not a feasible alternative, since general-purpose processors (GPP) offer low energy-efficiency and low data throughput. Thus, the computationally intensive application parts are often offloaded to dedicated processing elements (PEs) that can perform the computations faster and with lower power consumption.[1] However, running the application on multiple PEs raises the problem of synchronization: a processor must know before interacting with another PE, if the device is ready for the interaction. This synchronization can be performed either by polling the status of the other PE, or by letting the working PE announce when it is finished.[2] Recently, also an alternative way has been proposed for doing the synchronization of the accelerators: by using deterministic scheduling[3,4] it is possible to avoid repeated polling of other PEs, as well as interrupts.

In this paper we compare the synchronization overhead of a hardware-accelerated polling-based system against a more sophisticated synchronization-by-scheduling system. As a reference we also compare the aforementioned solutions against a full-software implementation. The comparison is based on the measurement of execution cycles, energy consumption and cache hit ratios. The measurement results have been acquired by running a multi-stream MPEG-4 decoding application on a cycle-accurate strongARM SA-1100 processor simulator.

Further author information:
Sébastien Lafond: sebastien.lafond@abo.fi
Jani Boutellier: bow@ee.oulu.fi

2. MEASUREMENT FRAMEWORK

The application framework for our measurements was multi-stream MPEG-4 video decoding performed by the open-source XViD[5] codec. MPEG-4 video decoding was chosen as the application framework, because it is computationally very demanding and dynamic. Decoding one second of a 320x240-pixel movie (with 25 fps) involves processing 7500 macroblocks, of which each can have a different decoding procedure. The decoding procedure of a macroblock is discovered as the video bitstream is read and can not be predicted in advance. Therefore, if the decoding is accelerated by hardware, the accelerator elements must be so fine-grained that they can be adapted to the decoding needs of each macroblock.[3] Naturally, the decoding hardware designer can also just assume the worst-case scenario and allocate the maximum amount of hardware resources for each macroblock, but this will lead to an inefficient solution. Although MPEG-4 is a bit dated as a video compression scheme, the results of this paper can also be applied to upcoming standards such as RMC.[6]

The measurements were conducted on three different system configurations: one of the systems ran several independent unmodified XViD software decoders on a single GPP by using multitasking, whereas the other two systems contained the GPP and several dedicated hardware accelerators to do the same task. Since the XViD codec is originally described in monolithic c-code, some effort was required to make the decoder suitable for hardware acceleration. These modifications introduced some processing-time and memory overhead.

The codec itself is only capable of decoding one video stream at a time, therefore an OS-like wrapper application was created for the two hardware-accelerated systems. Upon start, the wrapper application initializes 1...4 video decoders and starts decoding the video streams. The wrapper application does not run the decoding of separate streams in parallel, instead it uses time-division multiplexing of the GPP processing time. When the code execution of decoder instance *n* reaches a point that it requires code execution on hardware accelerators, it returns control to the wrapper application along with the accelerator access requests. The wrapper application stores these accelerator access requests (but does not activate the accelerators yet) and starts decoder instance *n+1*. When decoder *n+1* code execution reaches the stage that it can not proceed without use of accelerators, decoder *n+1* returns control to the wrapper application and so on. Once all the decoder instances have finished executing the control code and are waiting for computation results from the accelerators, the wrapper application starts to execute the accelerator access requests. How this is done, depends on the synchronization scheme.

The traditional, polling-based system starts processing the stored accelerator access requests in a first-requested first-served manner. However, if an accelerator is reserved, the new access request must yield and wait until the accelerator is freed – meanwhile the system continues polling the other accelerators to see if some other access request could be executed. The second system implementation uses run-time scheduling to plan the accelerator access pattern in advance. The scheduling algorithm itself causes some overhead, but on the other hand avoids completely the polling overheads later, because the schedule tells in advance when the resources will be freed. The completely software based stream decoding does not need further explanation: all computations are performed on the GPP in traditional time-division multitasking fashion.

Figure 1 shows approximately the behaviour of the hardware accelerated solutions in a Gantt chart. W represents processing time spent within the wrapper application and D1, D2 and D3 refer to activity in the control code of the respective decoder instances. The blocks on the accelerator rows below show an arbitrary schedule of task executions on the accelerators.

2.1 Scheduler

The scheduler used by the second accelerated system implementation is based on the idea that is described in.[4] Theoretically it is a permutation flow-shop (PFS)[7] scheduler, that has been applied to the problem of PE scheduling. In PFS terminology the processing units are *machines* and the tasks perfomed by the processing units are *operations*. Dependencies between tasks are described by grouping tasks into *jobs*. A job is defined to contain an operation for each machine, and each job must access the machines in the same order. In our application, we have also used *machine skipping*, which means that for some jobs, the execution time of certain operations may be zero: i.e. nothing is performed on that machine. By the PFS definitions, the execution times of operations are deterministic. This is not a severe limitation, since the accelerated functions are very predictable. Also, the assumption about deterministic execution times has been made previously in similar contexts.[8,9] From this

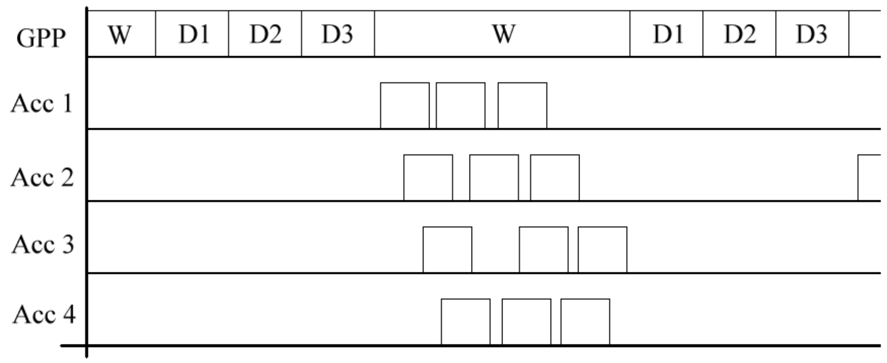


Figure 1. Sequential control code processing and parallel accelerator operation.

description, it is evident that PFS can be applied only to some scheduling problems, as the one presented in this paper.

The benefit of such a restricted scheduling problem is that the scheduling algorithm can be very straightforward and since the scheduler is called with a high frequency, a low overhead will be the most important characteristic of the scheduler. Thus, of the different scheduler implementations described in,[4] the "no job ordering, no-wait timetabling" was selected for computing schedules in this environment. No-wait timetabling (Figure 2) means that within the same job, the next operation is started immediately after the previous one finishes. In our scheduling problem this is essential, because it ensures that buffers between processing units are not overwritten too early.

2.2 Hardware Accelerators

The parts of the XViD code to be hardware accelerated, were selected manually. Evidently, it is most beneficial to use acceleration for compact parts of the code that are invoked often, e.g. nested loops. In MPEG-4 video decoding this part is found from macroblock decoding and especially in block decoding (in our case each macroblock consists of six blocks).

Besides being often invoked, it is desirable that the accelerated code parts should have a minimal amount of inputs and outputs. Based on these reasons, the hardware accelerators were created from the block decoding functions, that are depicted in Figure 3. The figure consists of boxes (accelerators), circles (buffers) and arrows that indicate the dataflow between the entities. It can be seen that there are several different dataflows, of which only some are used for each block, depending on the coding scheme.

When choosing the accelerated functions, some compromises had to be made with the modularity (amount of inputs and outputs) of accelerators. This does not affect the results between the two hardware accelerated systems, since both of them use the same accelerator units. However, it causes some extra overhead to the hardware accelerated systems when they are compared against the full-software based approach.

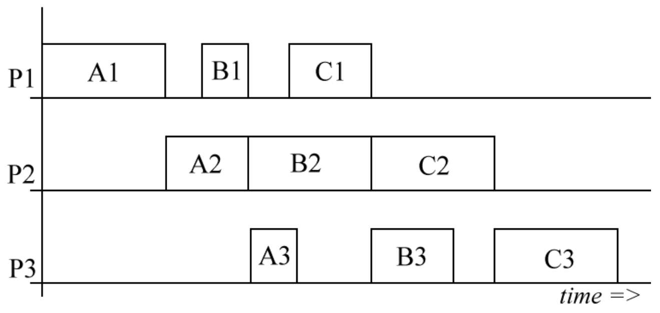


Figure 2. No-wait timetabling of three jobs (A,B,C) on three processors.

Table 1. Hardware accelerator latencies in clock cycles

Accelerator 1	Accelerator 2	Accelerator 3	Accelerator 4	Accelerator 5
16	25	13	8	200

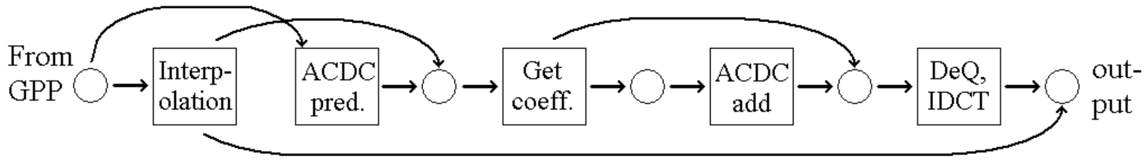


Figure 3. Data flow between accelerators.

2.3 Hardware Platform

The two hardware-accelerated decoding systems are based on the hardware platform presented in Figure 4. It consists of six PEs: a general purpose processor and five dedicated hardware accelerators. The PEs are triggered according to the polling-based approach or by the scheduling solution, that has been described previously. Both solutions trigger the accelerators in a ”waterflow” manner, where each accelerator passes its computed results to the following accelerator via a shared local memory. The full-software decoding system is running on the GPP of the same platform and does not have any use for the dedicated hardware accelerators. The used hardware accelerator latencies can be seen in Table 1.

3. SIMULATION FRAMEWORK

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

3.1 Processor simulator

The Sim-Panalyzer[10] processor simulator was used for this study. Sim-Panalyzer is based on the SimpleScalar[11] processor simulator, and performs cycle accurate simulation of a strongARM SA-1100 processor. At every simulated cycle it computes the energy consumption of each module within the ARM core (clock, ALU, cache, etc.). The processor simulator allows running an ARM-based operating system on top of it.

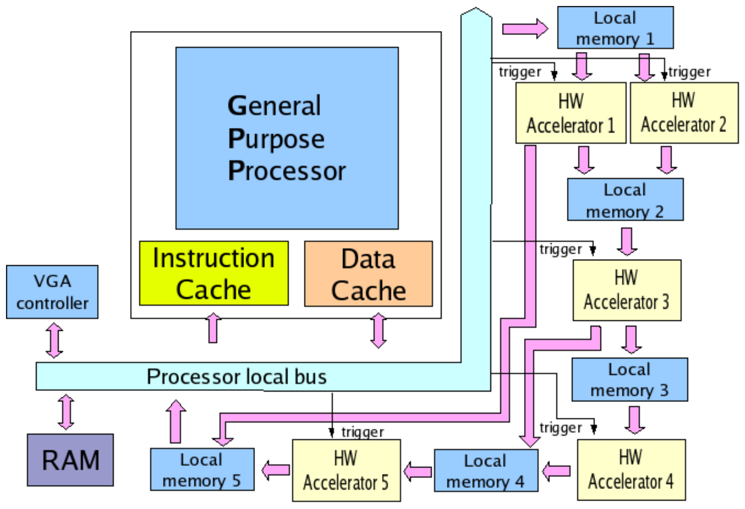


Figure 4. Hardware platform.

3.2 Operating system

The SimpleScalar port of the real-time operating system RTEMS (v. 4.6.2) was used in this study.[12] 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 the execution of a set of applications as independent tasks in a pre-emptive multitasking environment, which enabled us to conduct the measurements with multiple independent software decoders.

3.3 Accelerators

The accelerators are implemented within the Sim-Panalyzer framework and the applications can trigger the accelerator via dedicated system calls. It is the responsibility of the application to move the input data into the local memory 1 feeding the hardware accelerators 1 and 2 on Figure 4. In the same way, it is the responsibility of the application to read the results from local memory 5 when needed. For the rest of the accelerators reading input parameters is automatically done when the accelerator is triggered.

The following pseudo code illustrates how the communication with the hardware accelerators is handled for the polling-based and the scheduled systems:

```
Polling-based accelerator access
> repeat:
  > if(accelerator is free)
    > read input data to local memory
    > trigger/call hardware accelerator
  > endif
> goto repeat

Implementation using the scheduler
> compute schedule
> repeat:
  > sleep until designated time
  > read input data to local memory
  > trigger/call hardware accelerator
> goto repeat
```

In order to maintain the cache coherency the Sim-Panalyzer handles the memory accesses to the accelerator local memories as uncached memory regions. It is important to note that the execution of the hardware accelerators is performed outside the simulated platform. The energy consumption of the hardware accelerator is therefore not directly measured by the system. Instead, based of the difference in power dissipation between the full-software decoder system and hardware-accelerated systems presented in the results-section, it is possible to calculate an energy budget that tells us how much energy the accelerators can use to still provide a better energy efficiency than the full-software decoder.

Table 2. Memory latencies in cycles

	IL1	DL1	UL2	Local acc. memory	Main memory *first chunk access*	Main memory *inter chunk access*
Latency in cycles	1	1	4	4	30	4

Table 3. Configuration of the caches

Caches	Associativity	Size	# of 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

3.4 Simulation Parameters

This subsection defines the constant and variable parameters and the corresponding values used in the simulation framework. 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 and data cache and the unified secondary cache is presented in Table 3. Table 2 shows the different latencies for the caches and the local accelerator memories. All other parameters used by the Sim-Panalyzer were set to their default values. The used input data for all systems consisted of four compressed 320x240 pixel video streams, that had 45 frames each and a nominal speed of 15 fps. For the full-software implementation RTEMS uses time slices of 50 ms, which implies 20 task switches per second. This configuration tries to model an average embedded system that could be used in a multimedia handheld device.

4. RESULTS

For clarity all graphs presenting measurement results are given at the end of the paper in Appendix A. In the abbreviations FS stands for "full-software-based", HA for "hardware accelerated", PH for "polling, hardware accelerated" and SH for "scheduler, hardware-accelerated". Figures 5 to 9 present the cost differences for initializing the MPEG-4 video decoder with the full-software and hardware accelerated decoding systems. As the two hardware-accelerated decoding systems share the same piece of code for initializing the decoder, the initialization costs are common for both systems.

Figure 5(a) presents the execution time in clock cycles and Figure 5(b) presents the power dissipated by the microarchitecture for initializing the full-software and the hardware accelerated decoding systems. The better results for the hardware accelerated decoding systems presented in these two graphs can be explained by the task switching overhead in the multi-tasking environment for the full-software system. The numbers in Figure 7 can also be explained by the task switching overhead.However, Figure 8(b) presents an increase of misses in the level one data cache for the hardware accelerated decoding systems, which according to figures 9(a) and 9(b) led to an increase of hits in the unified level 2 cache. As shown on Figure 6, this affect the power dissipated by the unified level 2 cache. These variations in cache activity are explained by the modifications done to the XViD codec, to make it suitable for hardware acceleration. As stated before, the full-software implementation uses the original XViD.

Figures 10 to 14 present the average costs for decoding one frame with the full-software and the two hardware accelerated decoding systems. For each measurement the average costs for decoding one frame is obtained by applying the following formula:

$$\text{Average cost per frame} = \frac{\text{Total cost - Initialization cost}}{\text{Total number of decoded frames}} \tag{1}$$

Figure 10(a) shows the average speed of execution for decoding one frame on each system. The system using the hardware accelerator and the scheduler is more than twice faster than the polling based system and about 40% faster than the full-software system. Thus, if we take an average power dissipation of 550 mW for a strongARM SA-1100 processor,[13] the maximum average energy budget for all accelerators must stay below 220 mW, if we want to get the scheduler based system to have a better energy efficiency than the full-software system. On the other hand, figures 10 and 11 clearly show the inefficiency of the polling based system compared to the two others. As the graphs in figures 13 and 14 show, this inefficiency in execution time and power dissipation is mainly due to a huge data access increase in the polling based system. However, it must be pointed out that the software-based polling solution used here is clearly very inefficient and could be implemented in a much more efficient way by using some kind of hardware support. Finally, with the used simulation parameters, we can see that only the scheduler-based system is able to decode the four video streams in real time at 15 frames per second.

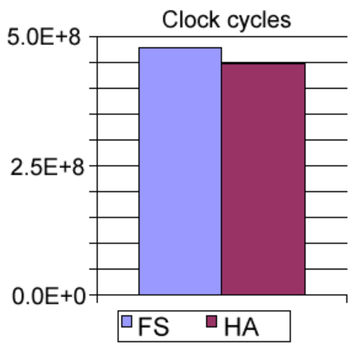
5. CONCLUSION

We have presented a comparison between three different multi-stream MPEG-4 video decoding systems. The comparison was made based on measurements of execution time, power dissipation and cache behaviours. The compared systems consisted of one fully software-based and two hardware accelerated solutions. One of the hardware-accelerated solutions used polling to do synchronization between processing elements, whereas the other one used a new scheduling-based synchronization approach. The measurement results showed that the hardware accelerated, scheduling-based solution provided the best energy efficiency of these three, if the hardware accelerators do not consume too much power.

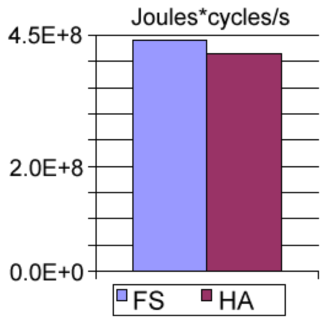
REFERENCES

1. W. Wolf, *High-Performance Embedded Computing*, Morgan Kaufmann, 2006.
2. O. P. Gangwal, A. Nieuwland, and P. Lippens, "A scalable and flexible data synchronization scheme for embedded hw-sw shared-memory systems," in *ISSS '01: Proceedings of the 14th international symposium on Systems synthesis*, pp. 1–6, ACM, (New York, NY, USA), 2001.
3. T. Rintaluoma, O. Silven, and J. Raekallio, "Interface overheads in embedded multimedia software," *Embedded Computer Systems: Architectures, Modeling, and Simulation* , pp. 5–14, 2006.
4. J. Boutellier, S. S. Bhattacharyya, and O. Silven, "Low-overhead run-time scheduling for fine-grained acceleration of signal processing systems," *Signal Processing Systems, 2007 IEEE Workshop on* , pp. 457–462, 17-19 Oct. 2007.
5. XViD-codec, "http://www.xvid.org."
6. C. Lucarz, M. Mattavelli, J. Thomas-Kerr, and J. Janneck, "Reconfigurable media coding: a new specification model for multemedia coders," in *Proceeding of the IEEE 2007 Workshop on Signal Processing Systems (SiPS)*, 2007.
7. S. French, *Sequencing and Scheduling*, Mathematics and its applications, Ellis Horwood Limited, 1982.
8. Y.-S. Chen, C.-S. Shih, and T.-W. Kuo, "Dynamic task scheduling and processing element allocation for multi-function socs," in *Proc. 2007 Real Time and Embedded Technology and Applications Symposium*, pp. 81–90, (Bellevue, WA), April 2007.
9. Y.-J. Kim and T. Kim, "A hw/sw partitioner for multi-mode multi-task embedded applications," *The Journal of VLSI Signal Processing* **44**, pp. 269–283, 2006.
10. Sim-Panalyzer, "http://www.eecs.umich.edu/~panalyzer."
11. D. C. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," Tech. Rep. CS-TR-1997-1342, 1997.
12. RTEMS, "http://www.jwhitham.org.uk/simplescalar/."
13. Intel-Corporation, "Intel strongarm sa-1100 microprocessor for embeddedapplications, brief datasheet, 1999."

APPENDIX A.



(a) Execution time



(b) Microarchitecture power dissipation

Figure 5. Execution time and microarchitecture power dissipation during initialization stage.

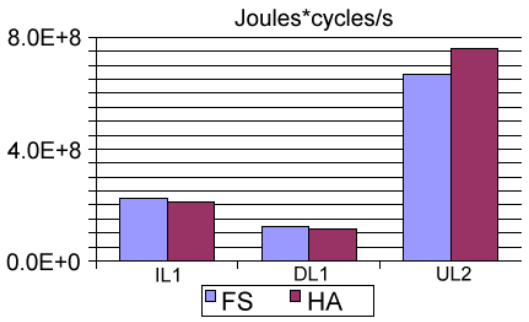
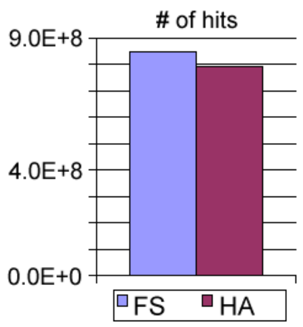
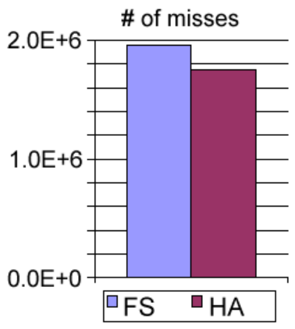


Figure 6. Power dissipation in level 1 and 2 caches during initialization stage.

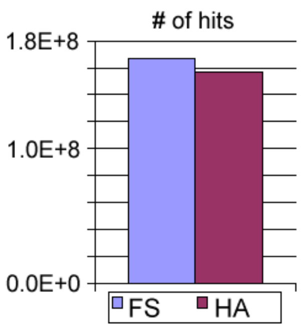


(a) IL1 hits

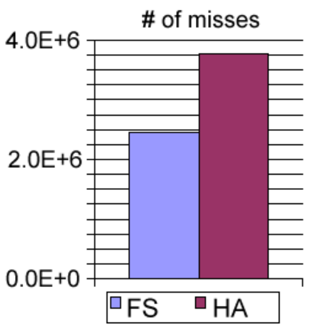


(b) IL1 misses

Figure 7. Instruction level 1 cache hits and misses during initialization stage.

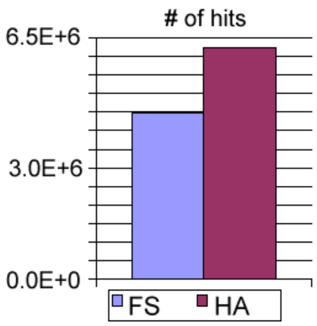


(a) DL1 hits

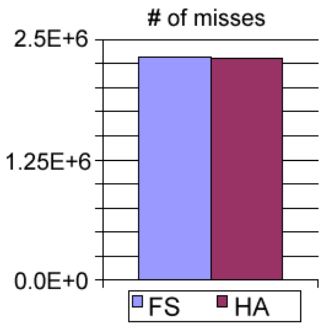


(b) DL1 misses

Figure 8. Date level 1 cache hits and misses during initialization stage.

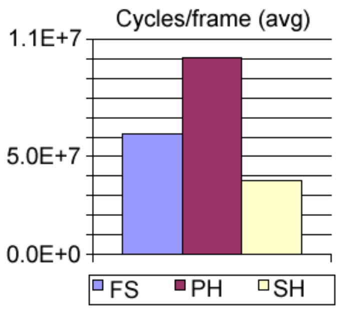


(a) UL2 hits

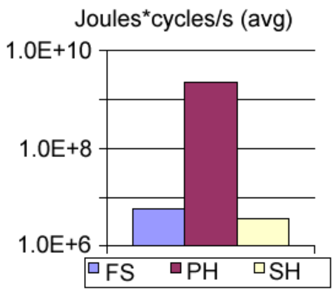


(b) UL2 misses

Figure 9. Unified level 2 cache hits and misses during initialization stage.



(a) Execution time



(b) Microarchitecture power dissipation

Figure 10. Execution time and microarchitecture power dissipation on average for decoding one frame.

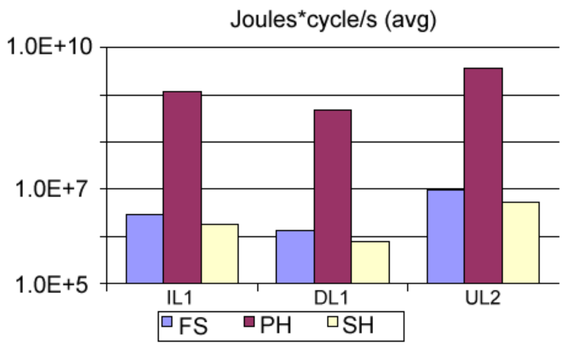
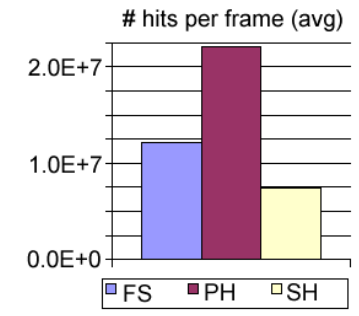
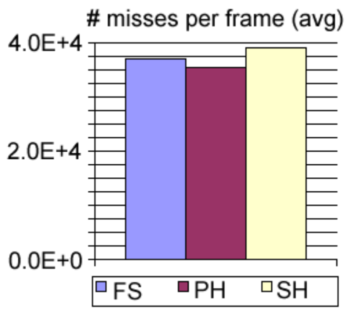


Figure 11. Power dissipation in level 1 and 2 caches on average for decoding one frame.

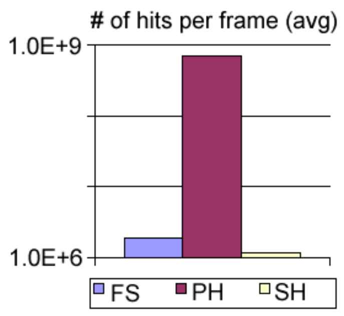


(a) IL1 hits

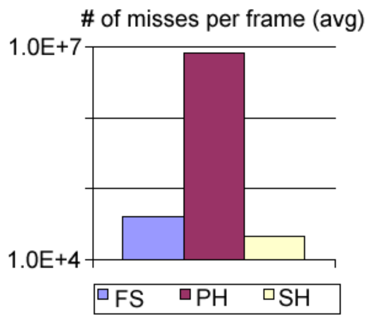


(b) IL1 misses

Figure 12. Instruction level 1 cache hits and misses on average for decoding one frame.

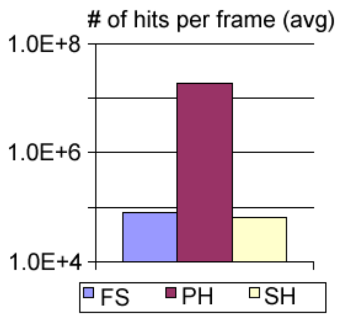


(a) DL1 hits

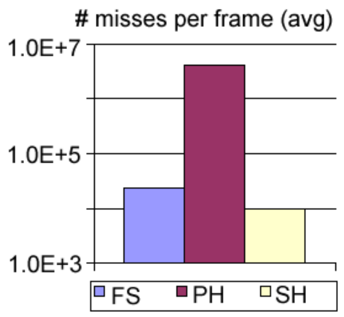


(b) DL1 misses

Figure 13. Date level 1 cache hits and misses on average for decoding one frame.



(a) UL2 hits



(b) UL2 misses

Figure 14. Unified level 2 cache hits and misses on average for decoding one frame.