# Cache Optimization for Mobile Devices Running Multimedia Applications

Abu Asaduzzaman, Imad Mahgoub, Praveen Sanigepalli, Hari Kalva, Ravi Shankar, and Borko Furht
Department of Computer Science & Engineering
Florida Atlantic University
777 Glades Road, Boca Raton, Florida 33431, USA
Tel: (561) 297-3855, Fax: (561) 297-2800
aasaduzz@fau.edu, imad@cse.fau.edu, spraveen@fau.edu, hari@cse.fau.edu, ravi@cse.fau.edu, and borko@cse.fau.edu

## Abstract

*The popularity of mobile/wireless embedded systems running multimedia applications is growing. MPEG4 is an important and demanding multimedia application. With improved CPU, memory subsystem deficiency is the major barrier to improving the system performance. Studies show that there is sufficient reuse of values for caching to significantly reduce the raw required memory bandwidth for video data. Decoding MPEG4 video data in software generates many times more cache-memory traffic than required. Proper understanding of the decoding algorithm and the composition of its data set is obvious to improve the performance of such a system. The focus of this paper is to enhance MPEG4 decoding performance through cache optimization of a mobile device. The architecture we simulate includes a digital signal processor (DSP) to run the decoding algorithm and a two-level cache system. Level-1 cache is split into Data (D1) and Instruction (I1) caches and level-2 (CL2) is a unified cache. We use Cachegrind and VisualSim simulation tools to optimize cache size, line size, associativity, and levels of caches for a wireless device decoding MPEG4 video.*

## 1. Introduction

Due to the growing demands in functionalities the size and complexity of multimedia application are increasing. For time critical applications, these systems must react to system changes and must compute certain results in real time. To cope with these issues, more computational power is required in their implementation. Increased computational power implies more traffic from CPU to Memory. The bandwidth to off-chip memories is not increasing as fast as the increase in speed of computation power, leading to a significant processor/memory speed gap. An approach to deal with memory bandwidth bottlenecks is to use cache(s). Cache is intended to give memory speed approaching that of the fastest memories in the system. Figure 1 shows the memory hierarchy with level-1 (CL1) and level-2 (CL2) caches. Cache improves performance by reducing the data access time. Data between CPU and cache is transferred as data object and between cache and main memory as block [1, 2, 3, 4].

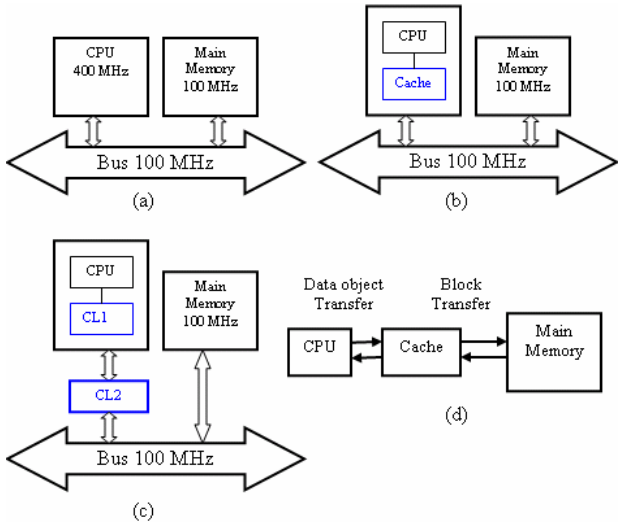


**Figure 1:** Memory hierarchy; (a) CPU, Main Memory, and Bus; (b) Processor cache (CL1); (c) CL2; (d) Data transfer among CPU, Cache, and Main Memory

Multimedia computing has become a practical reality, and computer architectures are changing in response. MPEG processing is a significant challenge for memory subsystems, which have become the primary performance bottleneck. The high data rate, large sizes, and distinctive memory access patterns of MPEG exert a particular strain on caches. While miss rates are acceptable, generate significant excess cache-memory traffic. Multimedia applications seriously suffer due to

Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering (ISMSE'04)
0-7695-2217-3/04 $20.00 © 2004 **IEEE**

cache inefficiency from dropped frames, blocking, or other annoying artifacts. For mobile devices, where power and bandwidth are limited, cache inefficiency can have a direct cost impact, requiring the use of higher capacity components that can drive up system cost [5, 6, 7].

The paper is organized as follows. Section 2 discusses related work. Section 3 presents a short overview of multimedia applications. Section 4 explains the simulated architecture. Simulation details are presented in Section 5. In Section 6, the simulation results are analyzed. Finally, we conclude our work in Section 7. At the end, VisualSim Block Diagram and Simulation Cockpit are attached as Appendix A.

## 2. Related Work

A number of studies have been done on cache optimization for mobile devices running multimedia applications. In this section, we include only those that are very relevant to the work presented in this paper.

A general-purpose computing platform running MPEG-2 application is studied in [4]. Most of the data transferred is concentrated on the main system bus. At the very least, there are two streams of encoded and decoded video being concurrently transferred in and out of main memory. Any excess memory traffic generated by cache inefficiency will further exacerbate this situation. It is found that there is sufficient reuse of values for caching to significantly reduce the raw required memory bandwidth for video data. The addition of a larger second level cache to a small first level cache can reduce the memory bandwidth significantly.

The problem related to improving memory hierarchy performance at system level for multitasking data intensive application is addressed in [3]. They propose compiler-like method for intra-task and analytical method to find a static task execution order for inter-task data cache misses by using cache partitioning. Due to the lack of freedom in reordering task execution, this method can optimize the caches more.

Cache behavior of multimedia and traditional applications is studied in [4]. The analysis show that multimedia applications exhibit higher data miss rate and comparable lower instruction miss rate. The study indicates larger data cache line sizes than are currently used would be beneficial in case of multimedia applications.

In [1] we explore the architecture of a multiprocessor mobile system running MPEG4 application. We develop a simulation program to evaluate the system performance in terms of utilization, delay, and total transactions for various CL1 sizes.

In this work, we focus on the impact of various cache design parameters namely, cache size, line size, associativity, and cache levels on the performance of MPEG4 decoding algorithm running on a single processor system.

## 3. Multimedia Applications

Multimedia is a combination of various data types including audio, graphics, and video. A multimedia application is one which operates on data to be presented visually and/or aurally. At its most basic level, compression is performed when an input video stream is analyzed and information that is less significant to the viewer is discarded. Each event is then assigned a code - commonly occurring events are assigned few bits and rare events will have more bits. The transmitter encodes and transmits the encoded video streams; the receiver decodes the encoded video streams and plays them back [9]. This section is an overview of MPEG4 video algorithm.

Moving Picture Experts Group (MPEG), the working group within the International Organization for Standardization (ISO), defined MPEG4, the next-generation global multimedia standard. MPEG4 delivers professional-quality audio and video streams over a wide range of bandwidths, from cell phone to broadband and beyond. MPEG4 considers both the spatial and temporal redundancy of video signals to achieve compression. Video data is broken down into 8 by 8 pixel Blocks and passed through a discrete cosine transform (DCT). The resulting spatial frequency coefficients are quantized, run-length encoded, and then further compressed with an entropy coding algorithm. To exploit temporal redundancy, MPEG4 encoding uses motion compensation with three different types of frames. I (intra) frames contain a complete image, compressed for spatial redundancy only. P (predicted) MPEG4 frames are built from 16 by 16 fragments known as macro-blocks. These consist primarily of pixels from the closest previous I or P frame (the reference frame), translated as a group from their location in the source. This information is stored as a vector representing the translation, and a DCT-encoded difference term, requiring far fewer bits than the original image fragment. B (bidirectional) frames can use the closest two I or P pictures - one before and one after in temporal order - as reference frames. Information not present in reference frames is encoded spatially on a block-by-block basis. All of data in P and B frames is also subject to run-length and entropy coding [10].

Consider a group of picture (GOP) that has 7 picture frames as shown in Figure 2. For decoding, these frames must be processed in the non-temporal order, which is a result of these dependencies. It is important that for a GOP the encoding, transmission, and decoding order is the same. Structure (at the encoder) is usually specified using two parameters, M and N. An I frame is decoded every N frames and a P frame every M frames, the rest are B frames with the consideration that the prediction error does not exceed a certain threshold. In this example, N = 7 and M = 3.

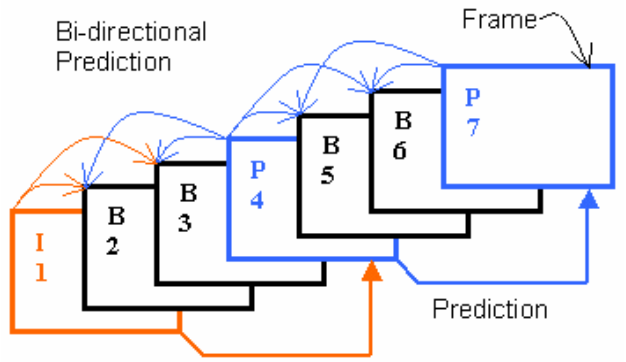


**Figure 2:** Sample MPEG4 picture frames

The decoder reads the MPEG data as a stream of bits. Unique bit patterns (start-codes) mark the division between different sections of the data. The simplified bit stream hierarchical structure is shown in Figure 3.

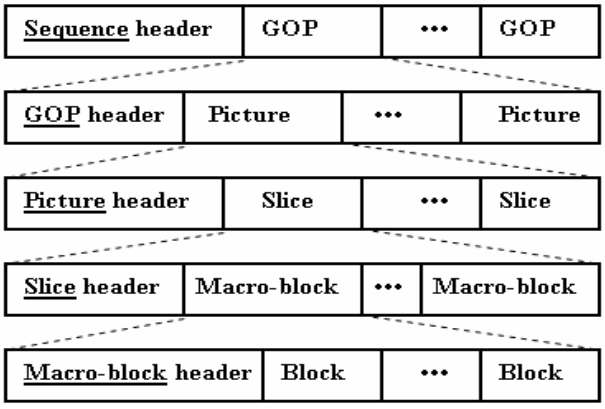


**Figure 3:** MPEG bit-stream structure

A Sequence (video clip) consists of groups of pictures (GOP). A GOP contains at least one I frame (picture) and typically a number of dependent P and B frames. Pictures consist of collections of macro-blocks called slices.

During MPEG4 encoding, both spatial and temporal redundancy is considered in order to achieve compression. Due to the fact that there are dependencies among frames while decoding encoded video, selection of right cache parameters may improve cache performance significantly.

## 4. Architecture

### 4.1 Cache Design Parameters

Following cache design parameters are examined in this work – cache size, line size, associativity, and cache levels.

**Cache size:** The first most significant design parameter is cache size. Cache size is usually increased by factors of two. For MPEG4 decoding, the cache-memory traffic is a function of cache size and increasing sizes show improvement, but may not be significant. Cache memory has cost and space constraints, so the decision of how large a cache to implement in a system is critical.

**Line size:** Sub-block placement can help decouple the size of cache lines and that of the memory bus. Low miss rates call for larger lines. Larger lines tend to provide superior spatial locality, but require more data to be read and possibly written back on a miss. For this reason, minimal memory traffic occurs with the smaller lines.

**Associativity:** Better performance can be achieved by increasing the level of associativity of smaller caches. Changing from a direct-mapped cache to a 2-way set-associative may reduce memory traffic by as much as 50% for small caches. Set sizes of greater than 4, however, show minimal benefit across all cache sizes [4].

**Multi-level caches:** CL2 cache between CL1 and main memory may significantly improve the CPU and overall performance. In general, addition of CL2 decreases the bus traffic and latency.

### 4.2 Simulated Architecture

We study cache optimization for a mobile device running multimedia applications. Our focus is on MPEG4 decoding algorithm. The simulation program is developed using VisualSim to evaluate the system performance in terms of utilization and total number of transactions processed by the different system components for various cache sizes, line sizes, associativity, and cache levels. We collect D1, I1, and CL2 references and misses for MPEG4

application using Cachegrind to drive our simulation model. The system architecture is shown in Figure 4.

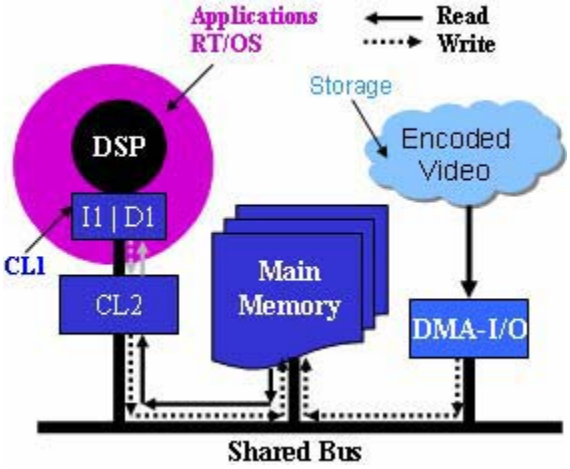


**Figure 4:** Simulated architecture for mobile devices decoding MPEG4 video

Digital signal processor (DSP) decodes the encoded video streams. DSP has CL1 and CL2. CL1 is a split cache for data (D1) and instruction (I1) caches and CL2 is a unified cache. We are interested to investigate the impacts of various CL1 and CL2 sizes on system performance. DSP and main memory are connected via a shared bus. DMA-I/O transfers and buffers encoded video data from the storage to the main memory. DSP decodes and writes the video streams into the main memory. The CPU reads the encoded data from and writes the decoded video into the main memory through its cache hierarchy. This paper focuses on cache optimization for video decoding and video playback is not considered.

## 5. Simulation

### 5.1 Simulation Tools

In this work, we use two simulation tools – Cachegrind from Valgrind and VisualSim from Mirabilis Design [11, 12].

Cachegrind is a simulation package, also known as a cache profiler [11]. Cachegrind performs detailed simulation of the D1, I1, and CL2 caches on an x86 machine. Total references, misses, and miss rates for D1, I1, and CL2 caches can be collected using Cachegrind.

VisualSim is an effective tool to simulate system level architecture [12]. VisualSim provides block libraries for various system components including CPU, caches, bus, and main memory. VisualSim simulation model is developed by selecting right blocks and making appropriate connections among them. VisualSim simulation cockpit provides functionalities to run the model and to collect simulation results. Detailed simulation block diagram and simulation cockpit are shown in Appendix A.

### 5.2 MPEG4 Workload

The workload defines all possible scenarios and environmental conditions that the system-under-study will be operating under. The quality of the workload used in the simulation is important for the accuracy and completeness of the simulation results [13, 14]. MPEG4 is an important and demanding multimedia application. In our simulation, we use cache (D1, I1, and CL2) hit ratios to model the system.

Different combinations of D1, I1, and CL2 sizes are used. Simulation results indicate that about 33% references are data (D1) and 67% references are instructions (I1) as shown in Table 1.

**Table 1:** Level-1 Data and Instruction references

| Ch. Sizes | D1 Refs (K) | | I1 Refs (K) | | CL1 Refs | |
|---|---|---|---|---|---|---|
| | Total | Miss | Total | Miss | D1% | I1% |
| **1*** | 18,782 | 521 | 38,758 | 512 | 33 | 67 |
| **2**** | 18,782 | 430 | 38,758 | 106 | 33 | 67 |
| **3***** | 18,782 | 403 | 38,758 | 39 | 33 | 67 |

**1*** D1+I1/CL2 – 8+8/128 K, Line Size – 16 B
**2**** D1+I1/CL2 – 16+16/512 K, Line Size – 32 B
**3***** D1+I1/CL2 – 32+32/2048 K, Line Size – 64 B

We calculate hit rates for D1, I1, and CL2 with various combinations of CL1 and CL2 sizes as summarized in Table 2. Line size and associativity is fixed at 16 B and 4-way, respectively. Hit rates increase with the increase of cache sizes.

**Table 2:** D1, I1, and CL2 hit ratios

| Cache Sizes D1+I1/L2 (KB) | Line Size | CL1 Hits | | CL2 Hits (%) |
|---|---|---|---|---|
| | | D1 (%) | I1 (%) | |
| 8+8/128 | 16 B | 95.0 | 98.0 | 99.3 |
| 16+16/512 | 32 B | 96.4 | 98.6 | 99.9 |
| 32+32/2048 | 64 B | 98.0 | 99.5 | 100.0 |

Table 3 shows the total amount of reads and writes of the data references. About 67% references are reads and 33% references are writes.

**Table 3:** Read and Write references

| CL2 Size (KB) | D1 References Read (K) | Write (K) | D1 References R (%) | W (%) |
|---|---|---|---|---|
| 32 | 12,391 | 6,391 | 67 | 33 |
| 128 | 12,391 | 6,391 | 67 | 33 |
| 512 | 12,391 | 6,391 | 67 | 33 |
| 2048 | 12,391 | 6,391 | 67 | 33 |

The numbers presented in Tables 1 – 3 are collected from Cachegrind and are used in the VisualSim simulation model.

### 5.3 Input Parameters

Cache sizes, line size, associativity, and levels of caches are varied as input to the VisualSim simulation model. System parameters are shown in Table 4.

**Table 4:** System parameters

| Item | Value |
|---|---|
| CL1 Cache sizes | 8+8 to 32+32 KB |
| CL2 Cache sizes | 32 to 4096 KB |
| Line size | 16 to 256 B |
| Associativity | 2-way to 16-way |
| Cache levels | L1 and L2 |
| Simulation time | 2000.0 simulation time units |
| Task time | 1.0 simulation time units |
| Task rate | Task time * 0.4 |
| CPU time | Task time * 0.4 |
| MEM time | Task time * 0.6 |
| Bus time | MEM time * 0.4 |
| CL1 Cache time | MEM time * 0.2 |
| CL2 Cache time | MEM time * 0.4 |
| Main memory time | Task time |
| Bus queue length | 300 |

### 5.4 Assumptions

The following assumptions are made for the VisualSim simulation model.

1. The dedicated bus that connects CL1 and CL2 introduces negligible delay compared to the delay introduced by the system bus which connects CL2 and main memory.
2. Write-back update policy is implemented. According to this policy the CPU is released immediately after CL1 is updated.
3. Task time has been divided among CPU, main memory, bus, level-1 and level-2 cache proportionally [12].

### 5.5 Performance Metrics

Using VisualSim, we measure the following two performance metrics – utilization and transactions.

**Utilization:** The CPU utilization is defined as the ratio of the time that CPU spent computing to the time that CPU spent transferring bits and performing un-tarring and tarring functions [12].

**Transactions:** Total number of transactions processed is the total number of tasks performed (entered and existed) by a component during the simulation [12].

## 6. Results and Discussion

In this research work, we use Cachegrind and VisualSim tools to investigate the impacts of various cache design parameters on the performance of MPEG4 decoding algorithm. Using Cachegrind, we obtain miss rates for D1+I1 cache sizes from 8+8 to 32+32 KB by a factor of 2, CL2 size from 32 to 4096 KB by a factor of 2, line size 16 to 256 B by a factor of 2, and associativity from 2-way to 16-way by a factor of 2. In this Section, we discuss the effects of level-1 cache sizes, line size, and associativity variation on miss rates. We, also, present the influence of the presence of a level-2 cache on utilization and total transactions.

### 6.1 Cache Size

Using Cachegrind, we change D1+I1 (CL1) from 8+8 KB to 64+64 KB by a factor of 2. We keep CL2 fixed at 1024 KB, line size 16 B, and 4-way associativity. The miss rates due to the variation of CL1 size are shown in Figure 5. It is noticed that the miss rates remain almost unchanged and using a CL1 size greater than 8+8 KB does not offer any benefit.

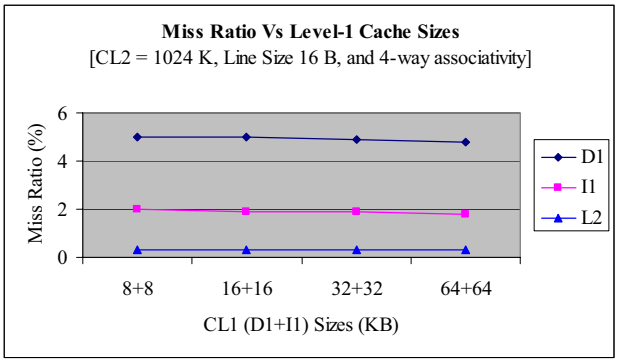


**Figure 5:** Miss Ratio versus CL1 Size

Secondly, we keep CL1 fixed at 8+8 KB, line size 16 B, and 4-way associativity. The miss rates due to the variation of CL2 size is shown in Figure 6.

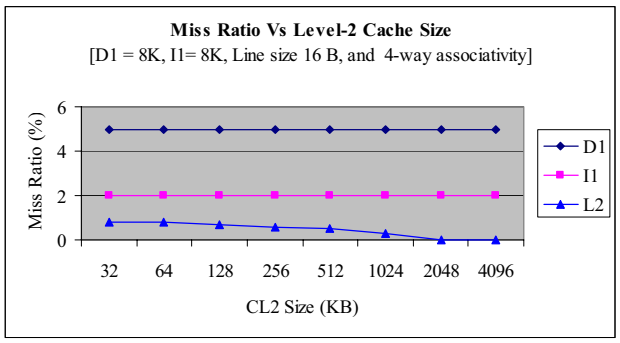


**Figure 6:** Miss Ratio versus CL2 Size

It is observed that for CL2 size from 32 to 512 KB the miss rates decrease slowly, from 512 KB to 2 MB the miss rates decrease sharply, and from 2 to 4 MB the miss rates remain almost unchanged. From cost, space, and complexity standpoints, larger CL2 may provide no significant benefit.

## 6.2 Line Size

For some applications, up to a certain point, increasing the line size may improve cache hit rates and performance as shown in Figure 7.

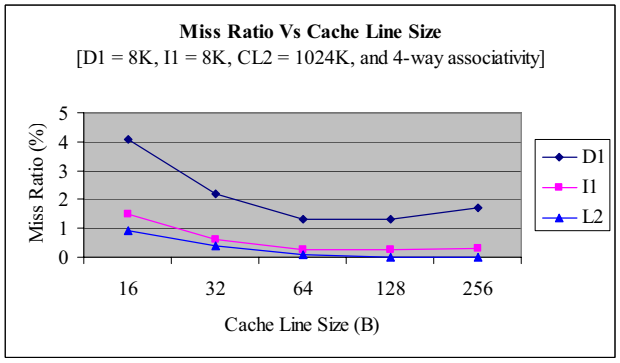


**Figure 7:** Miss Ratio versus Line Size

Using Cachegrind, we collect the miss rates for different line sizes while level-1 cache (D1+I1) size is fixed at 8+8 KB, level-2 cache size is fixed at 1024 KB, and associativity at 4-way. For a small cache like D1, miss rates start decreasing (in other words, hit rates start increasing) with the increase of line sizes. After a certain point, termed as cache pollution point, miss rates start increasing. For line size between 16 and 64 B, larger line size provides better spatial locality. For line size 128 B or higher, it requires more data to be read and written (in case of a miss).

## 6.3 Associativity

Using Cachegrind, we collect miss rates by varying associativity for D1+I1 cache sizes 16+16 KB, CL2 size 1024 KB, and line size 32 B. The miss rates for different associativity are shown in Figure 8.

The miss rates significantly decrease when we go from 2-way to 4-way associativity. Going to 8-way or higher, the changes are not significant.

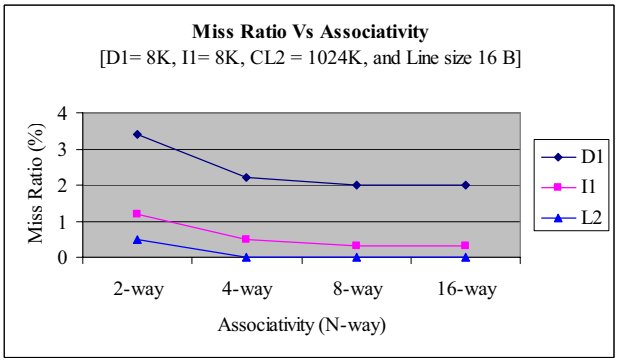


**Figure 8:** Miss Ratio versus Associativity

## 6.4 Cache Levels

Using VisualSim, we investigate the impact of the presence of a level-2 cache. CL2 size is varied and performance metrics, namely CPU utilization and total number of transaction, are collected. We keep D1+I1 sizes fixed at 8+8 KB, line size at 32 B, and associativity at 4-way. We change CL2 size from 32 KB to 4 MB. Total simulation time is 2000.0 and queue length is 300.

Table 5 shows transactions through different components. Memory requests are initiated by the CPU and are referred to D1 and I1; if not satisfied, be referred to CL2; finally, unsuccessful requests are satisfied from the main memory (MM).

**Table 5:** Total transactions for different CL2 sizes

| | 32K | 128K | 256K | 512K | 1M | 2M |
|---|---|---|---|---|---|---|
| **CPU** | 10K | 10K | 10K | 10K | 10K | 10K |
| **CL1** | 10K | 10K | 10K | 10K | 10K | 10K |
| **CL2** | 303 | 303 | 303 | 303 | 303 | 303 |
| **Bus** | 3 | 3 | 2 | 2 | 1 | 0 |
| **MM** | 3 | 3 | 2 | 2 | 1 | 0 |

MM transactions decrease with the increase of CL2 size. For a total of 10,000 tasks, 3,333 are data and 6,667 are instructions. All tasks are initiated at CPU and referred to CL1 (D1+I1). For D1 hit ratio 5.0% and I1 hit ratio 2.0%, 168 + 135 = 303 task requests go to CL2. For CL2 size 32 KB (miss ratio is 0.9%), only 3 requests go to MM via the shared bus. For CL2 size 2 MB and higher (miss ratio is 0%), no requests go to main memory.

Figure 9 shows the impact of CL2 size variation on CPU utilization (without and with CL1). CPU utilization decreases with the increase of CL2 size. Between CL2 sizes 512 KB and 2 MB, this decrement is significant. For CL2 size 128 KB or smaller and 4 MB or bigger this change is not significant.

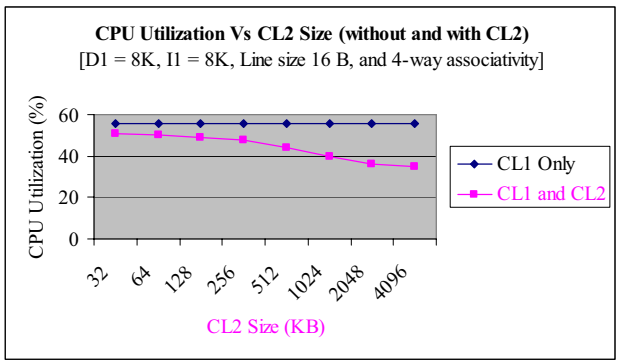


**Figure 9:** CPU Utilization versus CL2 Size

## 7. Conclusion

In this paper, we focus on enhancing MPEG4 decoding performance through cache optimization of a mobile device. We use Cachegrind and VisualSim simulation tools to optimize cache sizes, line size, associativity, and levels of caches for the system. The architecture we simulate includes a DSP to run the decoding algorithm and a two-level cache system. We collect total number of references and miss rates for D1, I1, and CL2 using Cachegrind to drive the VisualSim simulation model.

Techniques like selective caching, cache locking, scratch memory, and data reordering should improve the system performance, something that we will investigate in the future.

## 8. References


[1] A. Asaduzzaman and I. Mahgoub, "Evaluation of Application-Specific Multiprocessor Mobile System", *Proceedings of the 2004 Symposium on Performance Evaluation of Computer Telecommunication Systems*, pp. 751-758, San Jose, CA, July 2004.

[2] W. Stallings, "Computer Organization & Architecture Designing For Performance", *Prentice Hall*, Upper Saddle River, NJ, 6th edition, 2003

[3] A.M. Molnos, M.J.M. Heijligers, S.D. Cotofana, J.T.J. van Eijndhoven, and B. Mesman, "Data Cache Optimization in Multimedia Applications", *Proceedings of the 14th Annual Workshop on Circuits, Systems and Signal Processing*, ProRISC 2003, pp. 529-532, Veldhoven, The Netherlands, November 2003

[4] P. Soderquist and M. Leeser, "Optimizing the Data Cache Performance of a Software MPEG-2 Video Decoder", *ACM Multimedia 97 – Electronic Proceedings*, Seattle, WA, Nov. 1997

[5] N.T. Slingerland and A.J. Smith, "Cache Performance for Multimedia Applications" portal.acm.org/ft_gateway.cfm?id=377833&type=pdf

[6] C. Kulkarni, F. Catthoor, H. DeMan, "Hardware cache optimization for parallel multimedia applications", *Proceedings of the 4th International Euro-Par Conference on Parallel Processing table of contents*, pp. 923 – 932, 1998

[7] N.T. Slingerland and A.J. Smith, "Design and characterization of the Berkeley multimedia workload", *Multimedia Systems*, pp. 315-327, Springer-Verlag 2002

[8] F. Vahid and T. Givargis, "Embedded System Design – A Unified Hardware/Software Introduction", *John Wiley & Sons*, New York, NY, 2002

[9] R. Schaphorst, "Videoconferencing and Videotelephony – Techonology and Standards", *Artech House*, Norwood, MA, 2nd edition, 1999

[10] S.R. Ely, "MPEG video coding - A simple introduction", *EBU Technical Review Winter 1995*

[11] Cachegrind - a cache profiler: Valgrind http://valgrind.kde.org/index.html

[12] VisualSim – system-level simulator: Mirabilis Design, Inc. http://www.mirabilisdesign.com/

[13] A. Maxiaguine, S. Kunzli, and L. Thiele, "Workload Characterization Model for Tasks with Variable Execution Demand", *Project supported in part by KTI/CTI*, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland

[14] A. Avritzer, J. Kondek, D. Liu, and E.J. Weyuker, "Software Performance Testing Based on Workload Characterization", *WOSP '02*, July 24-26, 2002 Rome, Italy, AT&T Labs, ACM ISBN 1-1-58113-563-7 02/07, 2002

## Appendix A: VisualSim Block Diagram and Simulation Cockpit

Simulation tool used is VisualSim from Mirabilis Design, Inc. [http://www.mirabilisdesign.com/].

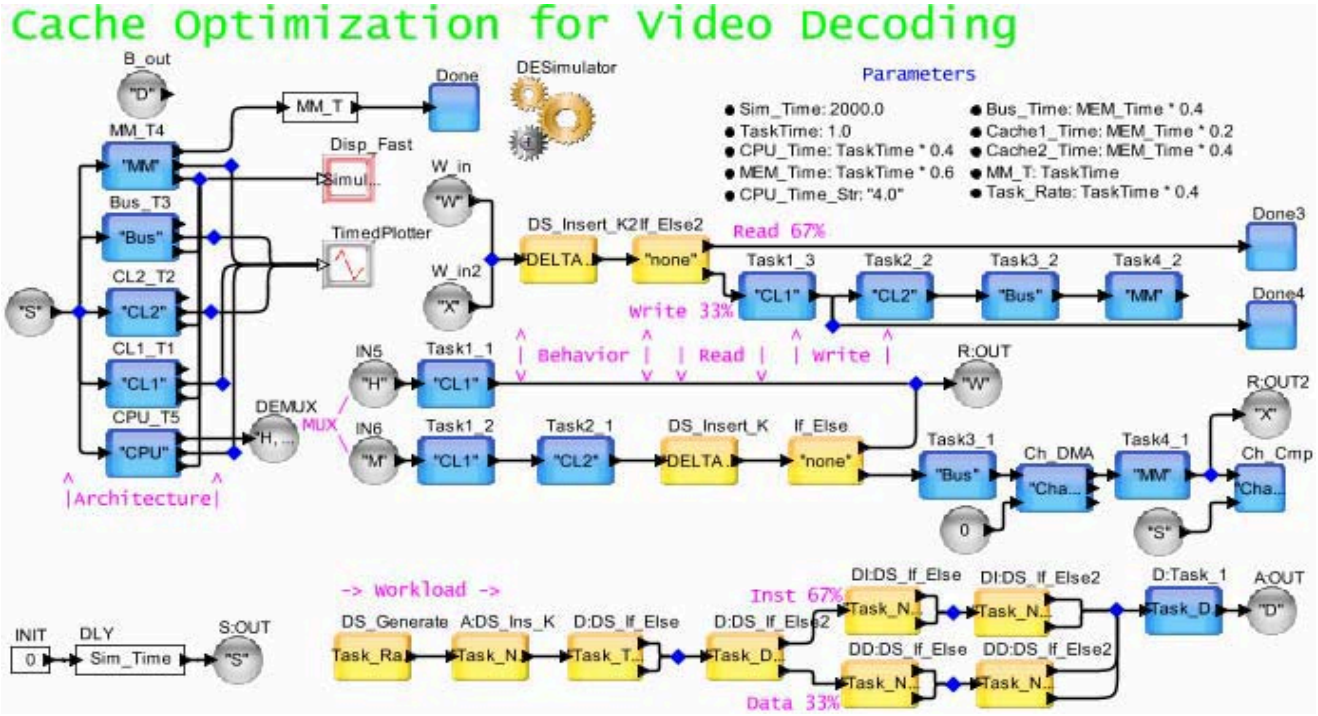


**Figure A.** VisualSim Block Diagram. The system to be evaluated can be described in three parts – Architecture, Behavior, and Workload. Architecture: Elements such as CPU, cache (CL1/CL2), bus, and main memory are specified here. Behavior: This describes the actions performed on the system. Examples include network traffic shaping. Workload: Transactions that traverse the system such as network traffic. Mapping between behavior and architecture is performed using Virtual Execution. Connection can be dedicated or Virtual. The virtual execution capability makes re-mapping from hardware to software by just changing a parameter. The output of a block can be displayed or plotted [Figure B].

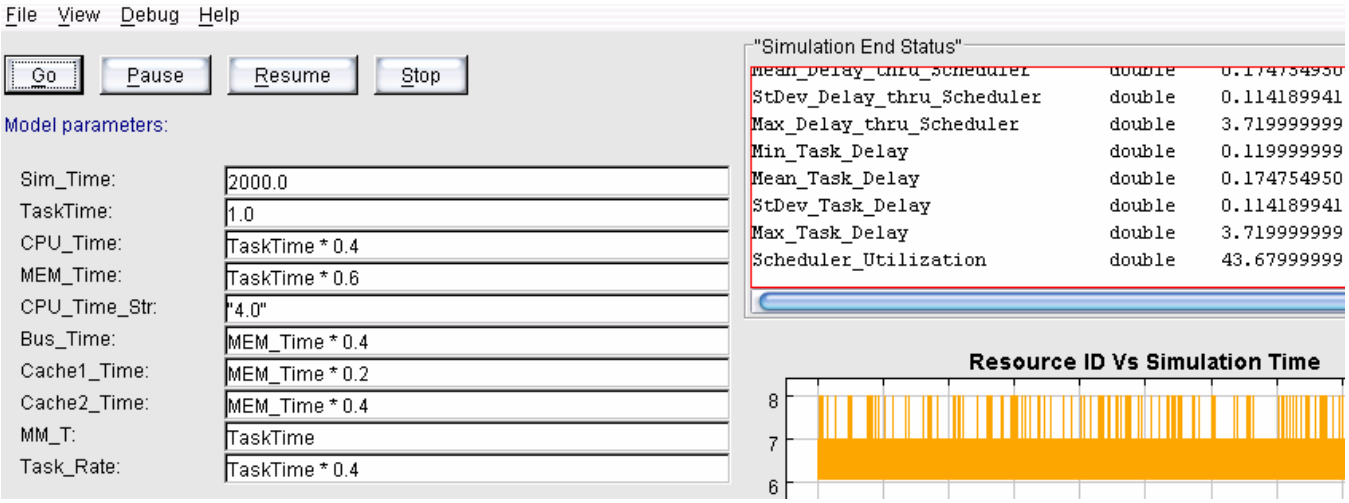


**Figure B.** VisualSim Simulation Cockpit (partial). The Simulation Cockpit provides functionalities (left-top) to run the model (block diagram) and to collect simulation results (right-top). Parameters can be changed before running the simulation without modifying the block diagram. The final results can be saved into a file and/or printed for further analysis.