MAPPING AND OPTIMIZATION OF THE AVS VIDEO DECODER ON A HIGH PERFORMANCE CHIP MULTIPROCESSOR

Konstantinos Krommydas, George Tsoublekas, Christos D. Antonopoulos, Nikolaos Bellas

Department of Computer and Communications Engineering
University of Thessaly
Volos, Greece
{kokrommi, getsoubl}@inf.uth.gr, {cda, nbellas}@uth.gr

ABSTRACT

Modern multimedia workloads provide increased levels of quality and compression efficiency at the expense of substantially increased computational complexity. It is important to leverage the off-the-shelf emerging multi-core processor architectures and exploit all levels of parallelism of such workloads in order to achieve real time functionality at a reasonable cost.

This paper presents the implementation, optimization and characterization of the AVS video decoder on Intel Core i7, a quad-core, hyper-threaded, chip multiprocessor (CMP). AVS (Audio Video Standard), a new compression standard from China, is competing with H.264 to potentially replace MPEG-2, mainly in the Chinese market. We show that it is necessary to perform a series of software optimizations and exploit parallelism at different levels in order to achieve FullHD real time functionality. The input dependent variability of execution time per work chunk is addressed using dynamic scheduling to allocate work to each thread.

Moreover, we evaluate the interaction of the application with the i7 CMP architecture using both high- and low-level performance metrics. Finally, we evaluate a new feature of Intel's i7 micro-architecture called Turbo Boost, which dynamically varies the frequencies of non-idling cores to optimize performance.

Keywords— Video decoding, AVS, Chip multiprocessor

1. INTRODUCTION

Video compression and decompression applications have become a very important and taxing workload for computing environments, especially due to the trend towards high quality, high definition and even ultra high definition video systems [8].

On the processor architecture front, difficulties in scaling single-thread performance and limiting the power envelope has forced CPU vendors to introduce general purpose multi-core units in a single die. The current trend in multi-core design shows that the number of cores in CMPs will double in every new processor generation resulting in tens and even hundreds cores per die in the near future [4]. In addition, multimedia ISA extensions like the SSE series for x86 or Altivec for PowerPC feature vector operations to exploit the data-level parallelism which is abundant in video codecs (section 2). Modern compilers or run-time systems offer little or no help in extracting higher granularity parallelism from the application; thus, the burden of optimizing the application to achieve a target performance falls squarely to the developer.

This paper details the mapping and optimization strategy of AVS (Audio Video Standard), on Intel's i7 multi-core processor (section 2), and explains how we can achieve real time AVS decoding for full HD resolution (section 3). The most promising technique is thread-level parallelization at the macroblock level, in which each thread processes one macroblock (MB) at a time. This type of parallelization is scalable with the number of cores provided that at any time there are enough independent macroblocks to keep the cores busy. Besides scalability, this approach favors load balancing if dependence-driven, dynamic scheduling is applied. Video codecs also exhibit high data-level parallelism, exploitable through the vector instructions of modern CPUs. However techniques such as software predication were required to overcome the complex control flow of the code and facilitate efficient vectorization. Such optimizations allowed for a cumulative speed-up from 3.37x (for lower resolution frames) to 5.63x (for higher resolution frames) with respect to the initial, AVS group reference implementation. The optimized code achieved real-time AVS decoding for FullHD videos.

This paper also presents a detailed characterization of macroscopic performance after each optimization step (section 3) as well as sensitivity metrics when varying input benchmark parameters (section 4).

Dynamic and static scheduling techniques at various levels of granularity have been studied extensively for H.264 decoding in [1, 2] and [7], respectively. Previous work includes scheduling on homogeneous multicore systems [1], the Cell BE processor [3], embedded systems [2] or simulators [7]. After scheduling threads at the frame, slice, row or MB levels authors agree that dynamic MB-level scheduling offers the best load balance and scalability at the expense of higher implementation complexity and often more power dissipation.

This paper is the first work to optimize the AVS decoder on a modern CMP system. It is also the first work to combine a complete repertoire of software and hardware optimizations such as MB-level dynamic scheduling, SIMDization and Turbo Boost [5] to achieve real time FullHD decoding on Intel's Core i7 processor.

2. AVS DECODER AND CMP TARGET PLATFORM

The AVS standard was drafted by the A/V work group of China to reduce royalty/licensing payments for the MPEG standards to non-

978-1-4244-7493-6/10/$26.00 ©2010 IEEE

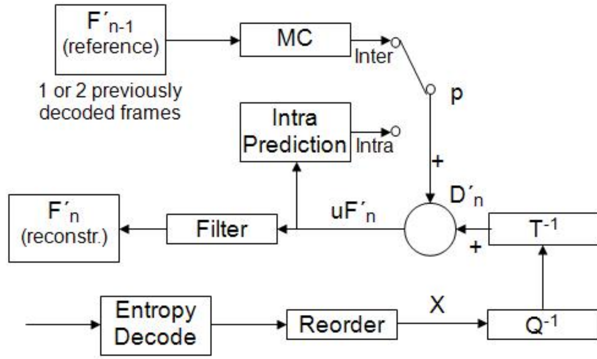


Figure 1. Block diagram of the AVS decoder

Chinese companies [6]. The AVS decoder functional diagram, shown in Figure 1, is similar to older standards such as MPEG-4 and H.264 and it uses similar modules. It has similar compression efficiency to H.264/AVC and as much as twice that of MPEG-2. However, it has lower computation and memory requirements than H.264 [6].

Video decoding is performed for each frame and its basic element is a *macroblock* (a portion of the video frame that is 16x16 pixels). It is based on the same block-based intra and inter prediction and transform-based coding framework of previous video standards. The intra predictions are derived from neighboring pixels in the top, top-left, top-right and left macroblocks. AVS supports four sizes of blocks (8x8, 8x16, 16x8 and 16x616) for inter prediction, as opposed to H.264 which supports block sizes as small as 4x4. The Motion Vector precision is quarter pixel. The AVS standard uses a deblocking filter on the reconstructed frame to eliminate blocking artifacts at the edges of blocks, mainly on areas with low spatial frequency. The deblocking filter is automatically adjusted for different scene activities and values of the quantizer value Q_p of each block.

Table 1 outlines the characteristics of the set of 6 videos used for the experimental evaluation of the AVS decoder. The source videos were coded, using the reference AVS encoder, into bitstreams at different target bitrates, ranging from 4 to 30Mbps depending on the frame resolution. We experiment with two frame resolutions: FullHD 1080p (1920x1080) and SD 576p (720x576). The frame sequence follows the pattern $\{I(BBP)_5\}_n$ and the YUV 4:2:0 format.

Our experimental platform is an Intel-based workstation, using an Intel E5520 i7 processor clocked at 2.27 GHz and 4GB of RAM. The processor integrates four identical cores. It supports Turbo Boost, a newly introduced feature that allows the chip to automatically and dynamically self-overclock the working cores, whenever some cores on the chip are idling and provided that the current and dissipated heat will not exceed the chip's working envelope. At the same time, the processor implements HyperThreading (a form of simultaneous multithreading). Each core integrates 12 execution units. It can simultaneously execute instructions from up to two threads, provided that they do not contend for the same execution units. Whenever contention for core resources arises, the execution of the contending threads on the specific core is serialized.

Throughout the paper we present results using as input the "rush hour" video at a FullHD resolution, encoded at 30Mbps. In all cases Turbo Boost is enabled on the processor, as it results to

Table 1. Benchmark videos

Videos	Description
Pedestrian area	Low camera position, people pass by very close to the camera. High depth of field. Static camera.
Rush hour	Many cars moving slowly, high depth of focus. Fixed camera.
Station2	Evening shot. Long zoom out. Many details, regular structures (tracks).
Mobcal	Moving text and a detailed photo. Moving train with colorful toys. Background with two types of wallpaper. Very detailed and demanding.
Park run	Long shot. Man running in a park with an umbrella in his hand. Trees, snow and water in the background. Very detailed and demanding.
Shields	Man with beard and a speckled jacket walking in front of a wall of detailed knight shields.

measurable performance benefits. The full set of videos/resolutions/bitrates is used in section 4 to perform a sensitivity analysis of the optimized decoder to input characteristics. In section 3 we study the effects of Turbo Boost and HyperThreading on performance.

All binaries have been compiled using version 11.1 of Intel C/C++ compiler with the same set of optimizations, including profile-driven ones.

To estimate the relative computational weight of the various modules of AVS decoder, we profiled the reference implementation. Fig. 2 (1st column) shows that Motion Compensation (MC) contributes more than 50% of the total execution time, primarily due to quarter pixel interpolation. Inverse transform and deblocking filter amount for more than 20% of the execution time. These three modules are clearly targets for

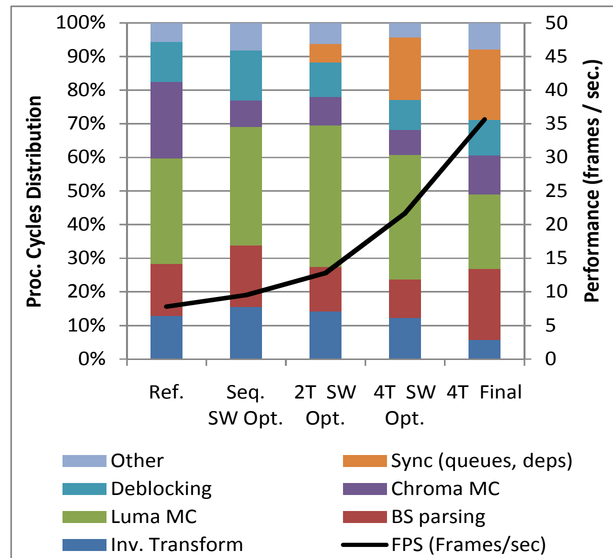


Fig. 2. Execution time breakdown and performance (in fps) for 5 different versions of the AVS decoder: Initial sequential reference code, sequential code after software optimizations, multithreaded version (2 threads), multithreaded version (4 threads) and final, vectorized, multithreaded code (4 threads).

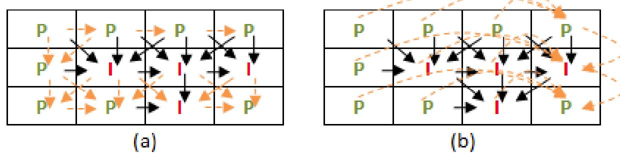


Figure 3. Macroblock dependencies in AVS decoder for intra (I) and inter (P) MBs. (a) Solid black arrows denote the dependencies due to intra prediction and deblocking filter. Dotted arrows denote the dependencies due to the deblocking filter only. (b) By performing filtering separately in a whole row, we can significantly reduce dependencies on P MBs and increase thread-level parallelism.

optimization. The unoptimized sequential reference code decodes FullHD frames at 7.2 frames/sec.

The challenge of improving the frame rate to achieve real time functionality (at least 30 fps) in a multi-core engine is to detect and exploit parallelism at all levels of granularity.

3. MAPPING AND OPTIMIZATIONS

The optimization and mapping to the CMP target platform has been performed in three steps: (i) sequential code optimizations, (ii) introduction of multithreading to exploit multiple cores and (iii) vectorization to exploit the SIMD capabilities of the processor. Fig. 2 depicts the processor cycles breakdown and performance in the gradually optimized decoder versions.

3.1. Sequential Code Optimizations

The first step of optimizing the reference code was to evaluate limited-scale algorithmic optimizations with the potential of significant performance impact.

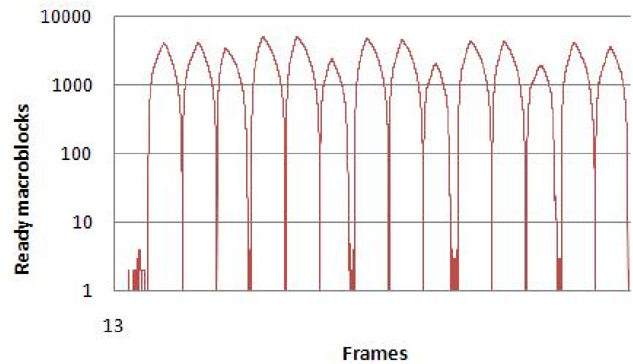
In AVS, motion vectors can reach out of the reference picture. The reference code makes extensive use of min/max macros – eventually translated to conditional branches with practically unpredictable behavior – to ensure that the pixels used for the interpolation are within the limits of the reference frame. In case a pixel outside the frame is needed, its value is approximated by that of the closest real pixel at the frame boundary.

We applied frame expansion to reduce non-predictable branches and the associated overhead of branch mispredictions. Each frame is transferred to a scratch buffer and extended – by replicating the boundary rows and columns – by as many rows and columns as necessary according to the sampling method used (up to 20 and up to 8 in each direction for luminance and chrominance samples, respectively). The extension ensures that all pixels required for the interpolation will be present in the extended frame.

The performance gain due to the elimination of min/max macros and due to data prefetching during the creation of the scratch buffer significantly outweighs the overhead of frame extension. More specifically the sequential frame extended version of the code is 1.32x faster than the reference implementation, yielding 9.52 FullHD frames per second. Fig. 2 indicates that chroma MC is the module that mostly benefits from this optimization.

3.2. Multithreading

In sequential AVS decoding and deblocking, frames are decoded sequentially and macroblocks (MBs) of each frame are processed



No.	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Type	I	B	B	P	B	B	P	B	B	P	B	B	P	B	B
#I MBs	8160	246	247	2537	278	229	3058	256	379	3224	536	880	2747	884	1018
#P MBs	0	7914	7913	5623	7882	7931	5102	7904	7781	4936	7624	7280	5413	7276	7142
Max Pop.	4	4113	4216	3510	5089	5194	2468	4946	4635	2068	4479	4399	1923	4230	3650

Figure 4. Ready queue population during the processing of a series of 15 frames. For each frame the table outlines its type (intra- (I) or inter-predicted (P/B)), the number of intra- (I) or inter-predicted (P) macroblocks in the frame and the maximum population of the queue.

in raster scan order. In order to exploit parallelism the programmer has to clearly identify dependencies between consecutive frames and MBs of the same frame. Intra-predicted MBs use data from neighboring MBs (upper left, upper, upper-right, left). The same set of data dependencies must be satisfied during the application of deblocking filter as well. Inter-predicted macroblocks, on the other hand, depend solely on macroblocks from the previous frame. This set of dependencies is characteristic of a 3D wavefront computation [2].

We opted to parallelize the decoder using a dependence-driven self-scheduling scheme at the MB level. All threads of the application are organized as a thread pool, getting work from a shared ready queue. Whenever each thread finishes the luminance and chrominance MC for a MB, it atomically updates (decrements) the dependencies of all dependent MBs. Ready MBs, namely MBs that have all their data-dependencies satisfied, are enqueued to the ready queue by the thread that satisfied their last dependence. That thread may opt to reserve one ready MB for itself, bypassing the ready queue.

Deblocking is significantly less computationally intensive than MC. We thus parallelized it at the granularity of a row. Threads that finish MC on the last MB of each row, deblock the row immediately above. Dependencies have been added from each MB in each row to the last MB of that row, in order to ensure that all MBs in row i have been motion compensated before deblocking MBs in row i-1. Non-deblocked luma and chroma samples of MBs in row i-1 may be needed during MC at row i, should row i contain any intra MBs. A dependence has thus been added from the last MB of row i-1 to the last MB of row i, to ensure that MC has finished on row i-1 before deblocking it. Figure 3 depicts dependencies before and after changing the granularity of deblocking to a full row of MBs. Apart from reducing the management and synchronization overhead, this strategy significantly simplifies the pattern and reduces the number of

dependencies. This, in turn, increases the degree of exploitable parallelism.

Figure 4 depicts the population of the ready queue for a representative window of 15 frames. For each frame we also provide the number of intra (I) and inter (P) macroblocks, as well as the maximum population of the queue. It is clear that application exposes ample parallelism during a vast percentage of its execution life. Parallelism is limited only in two cases: (i) at the beginning and end of processing of each frame, due to the fact that we do not exploit parallelism across frames, and (ii) whenever there are a lot of intra-predicted MBs in the frame. For example, I frames consist solely of intra-predicted macroblocks, thus the dependence pattern of Figure 3a is necessarily enforced, significantly limiting parallelism. This is an indication of the positive performance effect of data dependencies simplification through the granularity coarsening of deblocking. Exploiting MB-level parallelism across frames will potentially provide additional speed up in next generation many-cores systems. For an 8-thread processor inter-frame parallelism is not worth the extra complexity.

Other modules of the AVS encoder, such as bitstream parsing and entropy decoding, are either inherently sequential or notoriously difficult to parallelize. We therefore opted to overlap the bitstream parsing and entropy decoding modules with MC and deblocking. The first thread to observe that processing on the current frame has finished will start bitstream parsing and entropy decoding for the next frame. Bitstream parsing and entropy decoding proceed MB-by-MB. Each MB can be placed in the ready queue as soon as it has been bitstream parsed and entropy decoded, provided also that all its dependencies have been satisfied. When the thread finishes bitstream parsing and entropy decoding for the whole frame, it joins the other threads in decoding MBs.

The non-trivial dependencies pattern, as well as operations on the ready queue introduce a non-negligible overhead of 5% and 16% during the execution with 2 and 4 threads respectively (Fig. 2), despite the fact that synchronization has been implemented with fast, non-blocking operations. However, multithreaded AVS decoding with 2 and 4 threads results to a frame rate of 12.79 and 21.67 fps for the FullHD video, namely a speedup of 1.77x and 3x respectively over the initial, reference code.

3.3. Vectorization

Vector operations have been used for the interpolation process during luma and chroma motion compensation, for luma and chroma deblocking, as well as for the inverse transform.

Taking luma motion compensation as an example, as luma and chroma samples at sub-sample positions do not exist, it is necessary to generate them from nearby samples. Most of the complexity of the MC module, approximately 50% of the total execution time in the reference code, is due to the quarter pixel interpolation.

The predictive value at half sample position can be obtained with horizontal or vertical interpolation using the four-tap filter F1 (-1, 5, 5, -1). Similarly, the predictive value at a quarter sample position can be obtained with interpolation using the four-tap filter F2 (1, 7, 7, 1). For example, the interpolation of half sample b in Figure 5 is given by:

$$b' = -C + 5D + 5E - F \text{ and } b = clip((b'+4) >> 3).$$

The interpolation at quarter pixels requires integer and half sample values. For example, the quarter pixel value a is given by:

$$\alpha' = ee + 7D' + 7b' + E \text{ and } \alpha = clip((\alpha'+64) >> 7).$$

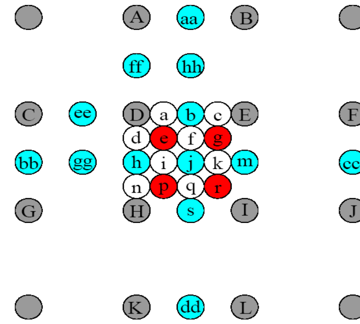


Figure 5. Interpolation of Luma components

The 128-bit wide SSE vector operations of Intel i7 allow the simultaneous calculation of up to a 8 samples of an interpolated luma block. Since input data are typically unaligned, they need to be "manually" loaded to the SSE registers, even if the interpolation is performed horizontaly. Shift operations are used whenever possible to reduce the number of costly, unaligned loads. Note the significant amount of reuse when a row/column of pixels is interpolated: to compute the next quarter pixel to the right of pixel E, we reuse the values of b, E and F. All these 8 samples per block can be combined as a vector and computed in a single step.

Chrominance is sampled at half the frequency of luminance. It is thus possible to vectorize chroma interpolation using 64-bit vector instructions (MMX), which perform significantly better than the corresponding 128-bit ones (SSE).

Inverse transform and deblocking are 2D operations, implemented as a series of 1D operations, one in each direction. In fact, vertical 1D operations are substituted by horizontal ones, preceded by block transposition. The 1D inverse transform follows a butterfly pattern that maximizes reuse of intermediate results.

Luma and chroma deblocking are characterized by a complex control flow (series of if/else instructions). Such codes are difficult to vectorize and their performance is limited due to mispredicted branches. In order to overcome this problem we apply a form of "software predication": we use vector operations to redundantly calculate the potential results for all possible code paths, for a row of 8 output samples. We then select and store the correct result for each sample, according to the path that would actually be executed, given the input data.

The vectorized version of the AVS decoder reaches, a 35.65 fps frame rate, when executed with 4 threads, exceeding real-time requirements (30 fps). This is a 4.95x improvement over the initial reference code.

3.4. Analysis

Figure 7 depicts the execution time breakdown for the first 58 frames of "rush hour". Intra frames (such as 13, 28 and 43) are decoded faster due to the smaller prediction component (although the deblocking filter is slightly more expensive than in inter frames). The bitstream parsing component shows higher variations across frames and can be a serious bottleneck because it occupies one thread and often causes the remaining threads to spin idle for potentially large amounts of time waiting for work.

Turbo Boost proves beneficial for performance, without requiring any effort from the compiler or the programmer. The average performance improvement ranges between 3-4% in the reference code. As expected, Turbo Boost is more effective when executing on fewer cores. Using the hardware performance counters on the system we identified the average operating

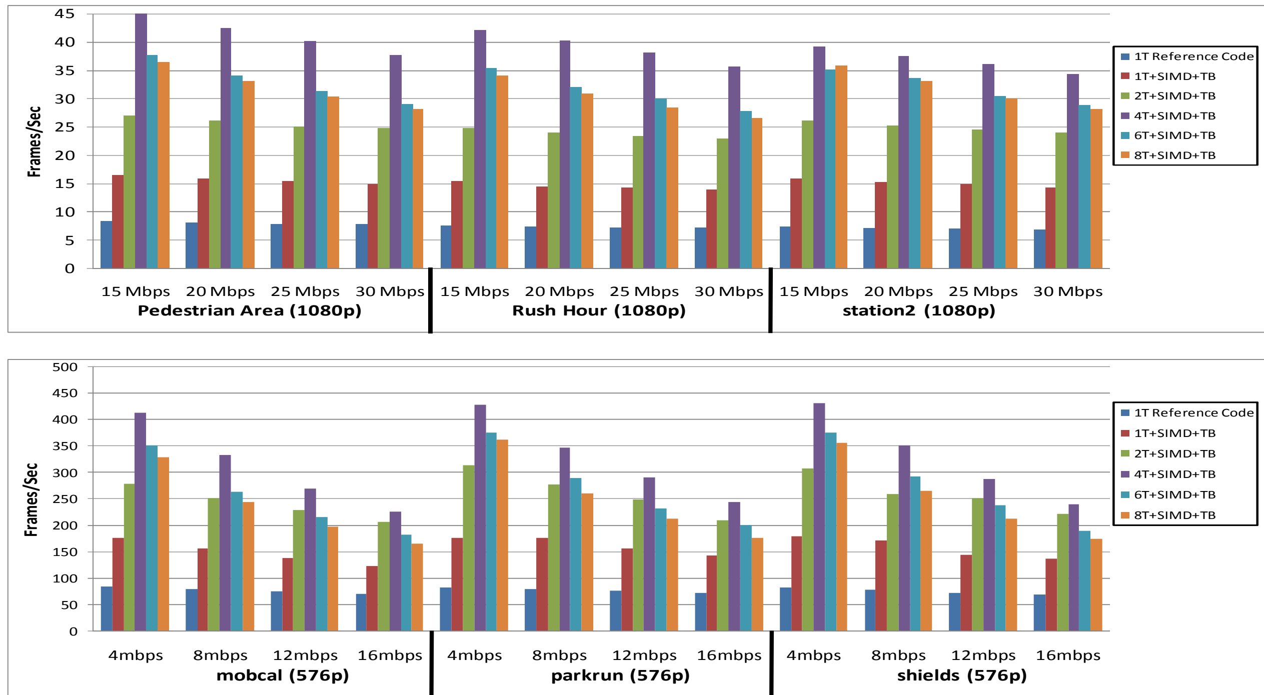


Figure 6. Performance (in frames / sec) for FullHD (top) and SD (bottom) resolution across, different input videos and compression bitrates

frequency during the execution of the fully optimized version of the code to 2.52, 2.50 and 2.39 GHz when 1, 2 and 4 cores were active respectively. Even when all cores were active, Turbo Boost managed to dynamically overclock the processor on average 120MHz (5.3%) over the base operating frequency.

As the reader can observe in Figure 6, the HyperThreading feature of Core i7 had an adverse impact on performance. The extensive use of vector instructions exercises resource pressure on the cores. When the number of threads exceeds the number of available cores, the extra threads request already busy execution units, due to the fact that all threads execute the same code. At the same time, HyperThreaded execution reduces the opportunities for the activation of Turbo Boost. Performance counters pinpointed the data & instructions load unit of cores as the bottleneck in the case of AVS. A pipelined parallelization strategy might offer more benefits from HyperThreading, provided that it would be possible to pair thread with qualitatively different instruction mixes on each core.

4. SENSITIVITY ANALYSIS

The next set of experiments investigates the robustness of our optimizations across different experiments that vary the input benchmarks (Table 1), frame resolutions, and compression bit rates.

Figure 6 shows that the fully optimized implementation of the AVS decoder always achieves real-time functionality at slightly less than 35 fps in the worst case (station2, 30 Mbps bitrate). Performance and scalability are relatively insensitive to the input video, however they are sensitive to its bitrate and resolution. The worst case scenario occurs in higher bitrates and lower resolution (16Mbps, 720x576). In this case, there is less potential for parallelism due to a smaller percentage of P MBs, and additionally work per macroblock is not enough to outweigh management overhead. This limits the speed up to 3.37x compared with the reference unoptimized code.

As expected, the performance of the optimized code scales with higher frame resolution (Figure 4). For FullHD resolution, the speed up varies from 4.83x (higher bitrate, higher number of intra-MBs) to 5.63x (lower bitrate, higher number of inter-MBs).

5. CONCLUSIONS

In this paper, we have presented a complete methodology for mapping and optimization of the AVS video decoder on a modern chip multiprocessor, which leads to real-time FullHD video decoding.

Thread-level parallelism will be the prevalent source of performance improvement for video decoders as technology moves towards platforms with hundreds of cores. This is due to the plentiful MB-level parallelism within and across frames, with the latter becoming more and more important as the number of available cores increases. Since the processing time per MB and the number of dependencies between MBs are input dependent, dynamic scheduling at the MB level, together with sophisticated granularity control will probably remain the key to efficient utilization of next generation parallel platforms.

Vectorization is an integral part of the optimization of video applications. In this paper, we show how vector instructions of the x86 architecture complement thread-level parallelism for an additional 1.65x speed-up on average. Sequential code optimizations will always be critical for performance improvement

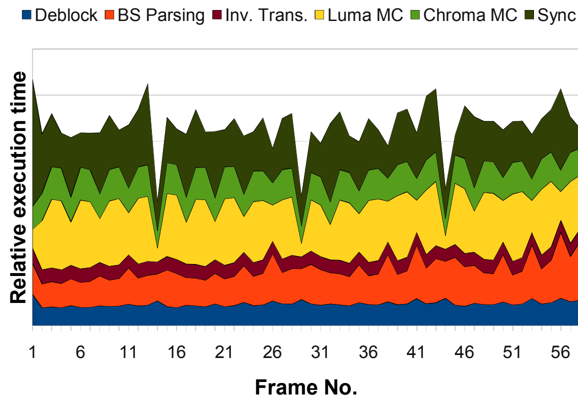


Figure 7. Relative execution time of main AVS modules for a frame sequence.

in video applications. Since there is a significant non-parallelizable portion in every video standard (parsing and entropy decoding), Amdahl's law will severely curtail speed up if the programmer does not optimize for single-threaded performance.

Acknowledgements
This project is partially supported by the EC Marie Curie International Reintegration Grant (IRG) 223819.

6. REFERENCES

[1] M. Alvarez, A. Ramirez, M. Valero, A. Azevedo, C.H. Meenderinck, B.H.H. Juurlink, "Performance Evaluation of Macroblock-level Parallelization of H.264 Decoding on a cc-NUMA Multiprocessor Architecture," Avances en Sistemas e Informática, June 2009, Volume 6, Number 1, ISSN 1657-7663.

[2] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J. Hoogerbrugge, M. Alvarez, A. Ramirez, M. Valero, "Parallel H.264 Decoding on an Embedded Multicore processor," Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC), pp. 404 – 418, 2008, Paphos, Cyprus.

[3] M. Baker, P. Dalale, K. S. Chatha, S. B. K. Vrudhula, "A Scalable Parallel H.264 Decoder on the Cell Broadband Engine Architecture," CODES+ISS, October 11-16, 2009, Grenoble.

[4] S. Borkar, "Thousand core chips: a technology perspective," Proceedings of the 44th Design Automation Conference (DAC), 2007, pp. 746-749.

[5] Intel Corporation, "Intel Turbo Boost Technology in Intel Core Microarchitecture (Nehalem) Based Processors", White Paper, November 2008.

[6] Jose Lau, "MPEG-4, AVS deliver better video compression more flexible format," *Electronic Times Asia,* June 1st, 2006.

[7] F. H. Seitner, M. Bleyer, R. M. Schreier, M. Gelautz, "Evaluation of data-parallel splitting approaches for H.264 decoding," 6th International Conference on Advances in Mobile Computing & Multimedia, pp. 40-49, November 24-26 2008, Linz, Austria.

[8] M. Sugawara, "Super hi-vision – research on a future ultra HDTV system, European Broadcasting Union, Tech. Report, 2008. *www.ebu.ch/fr/technical/trev/trev_2008-Q2_nhk-ultra-hd.pdf*