Parallel Implementation of Arbitrary-Shaped MPEG-4 Decoder for Multiprocessor Systems

Milan Pastrnak[*,a,c], Peter H.N. de With[a,c], Sander Stuijk[c] and Jef van Meerbergen[b,c]

[a]LogicaCMG Nederland B.V., RTSE Eindhoven; [b] Philips Research Labs. Eindhoven;
[c]Eindhoven University of Technology, PO Box 513, 5600 MB, The Netherlands

ABSTRACT

MPEG-4 is the first standard that combines synthetic objects, like 2D/3D graphics objects, with natural rectangular and non-rectangular video objects. The independent access to individual synthetic video objects for further manipulation creates a large space for future applications. This paper addresses the optimization of such complex multimedia algorithms for implementation on multiprocessor platforms. It is shown that when choosing the correct granularity of processing for enhanced parallelism and splitting time-critical tasks, a substantial improvement in processing efficiency can be obtained. In our work, we focus on non-rectangular (also called arbitrary-shaped) video objects decoder. In previous work, we motivated the use of a multiprocessor System-on-Chip(SoC) setup that satisfies the requirements on the overall computation capacity. We propose the optimization of the MPEG-4 algorithm to increase the decoding throughput and a more efficient usage of the multiprocessor architecture. First, we present a modification of the Repetitive Padding to increase the pipelining at block level. We identified the part of the padding algorithm that can be executed in parallel with the DCT-coefficient decoding and modified the original algorithm into two communicating tasks. Second, we introduce a synchronization mechanism that allows the processing for the Extended Padding and postprocessing (Deblocking & Deringing) filters at block level. The first optimization results in about 58% decrease of the original Repetitive-Padding task computational requirements. By introducing the previously proposed data-level parallelism and exploiting the inherent parallelism between the separated color components (Y, Cr, Cb), the computational savings are about 72% on the average. Moreover, the proposed optimizations marginalize the processing latency from frame size to slice order-of-magnitude.

Keywords: MPEG-4, texture padding, task-level parallelism, synchronization

1. INTRODUCTION

Arbitrary-shaped video objects (AS-VO) are en essential part of the object-based video signal coding. The MPEG-4 video standard [1] extends the traditional frame-based processing towards the composition of several video objects superimposed on a background sprite image. For the proper rendering of the scene without disturbing artifacts on the border of video objects (VO), the compressed stream contains the encoded shape of the VO. The MPEG-4 standard includes two algorithms for encoding the shape information. The first algorithm is Shape-Adaptive DCT (SA-DCT), that provides the same number of transform coefficients as the number of pixels enclosed by the VO part in the pixel block. However, it has several drawbacks [2] such as the non-orthogonality and the mean weighting defect, so that it requires the modification of the DCT transformation, thereby limiting reuse of existing DCT hardware solutions. In our work, we focus on the second type of encoding, which is based on Context Arithmetic Encoding (CAE). The shape information is encoded separately by CAE and the texture part of the object is encoded with the conventional MC-DCT coding. This type of AS-VO coding is relying on the traditional block-based processing.

In previous work, we motivated the implementation of recent multimedia algorithms on a multiprocessor platform [3]. Such a multiprocessor system is attractive for several reasons, of which the primary motivation is the increase of the throughput per application. Advanced applications like the surveillance and the medical domain, are making use of arbitrary-shaped VO coding. The original algorithm for AS-VO decoding was initially designed for execution on a single CPU. Therefore, the straightforward implementation following the MPEG-4

* E-mail: m.pastrnak@tue.nl, Telephone: +31 (0)40 247 3708

standard is not optimal for direct implementation on a multiprocessor platform. The availability of a plurality of processing cores speeds-up the processing by introducing several types of parallelism, which are not envisioned for the implementation on single processor systems. For example, the split of the texture processing into individual color components which are processed on a single processor, introduces only extra communication and buffering overhead, while it does not speed-up execution. In contrast with this, the simultaneous processing of individual texture components on multiprocessor systems contributes to the overall throughput by the length of the critical path in the computation graph. For example, the most complex color-component processing in the MPEG-4 standard, that is based on 4:2:0 encoding of AS-VOs, is the processing of the luminance signal component.

The parallel execution of advanced multimedia coding is a topic of continuous study. Let us discuss two recent examples briefly. Li *et al.* [4] discuss the encoding approaches of motion estimation on a parallel bus network. They exploit the granularity of the load partitions and associated overheads for minimization of the overall processing time. An alternative coding architecture is proposed by Fang *et al.* in [5]. This system is processing all bit planes in parallel in order to minimize JPEG-2000 encoder state memories. The platform is based on a reconfigurable FIFO architecture. The common element in these studies are elegant exploitation of the granularity in processing, but the processing data itself is still based on conventional rectangular video pictures.

This paper addresses the optimization of the arbitrary-shaped VO decoder algorithm for implementation on multiprocessor platforms. It is shown that when choosing the correct granularity of processing for enhanced parallelism and splitting time-critical VO-related tasks, a substantial improvement in processing efficiency can be obtained. In our work, we focus on non-rectangular (also called arbitrary-shaped) video objects decoder. The identification of the data availability increases the system throughput by higher task-level parallelism. First, the nature of encoded data and availability of shape information prior to texture decoding leads to the concept of splitting the critical object-decoding tasks (e.g. *Repetitive Padding*). Second, we introduce a synchronization mechanism that allows the processing for the Extended Padding and postprocessing (Deblocking & Deringing) filters at block level. Third, we exploit the inherent parallelism of the individual color components in the video signal. It will be shown that a substantial reduction of computing power can be achieved, combined with a lower latency.

The sequel of this paper is divided as follows. In Section 2, we discuss three different forms of parallelism that should be considered when creating a parallel implementation of an application. Section 4 presents the modification of the straightforward implementation of Repetitive Padding. The modifications of the Extended Padding algorithm and the post-processing filters are presented in Section 5. In Section 6, we describe the application of data-level parallelism principles. The contribution of presented techniques to the overall system performance is summarized in Section 7 and concludes this paper.

2. PARALLELISM OVERVIEW

This section addresses three forms of parallelism which are all briefly discussed in the framework of multimedia video coding. It is assumed that a video coder can be described with a flow graph of computing and communicating tasks. All tasks can be executed on different processors and depending on the nature of the algorithm, parallel execution can be realized. At the end of this section, we also provide a strategy for extracting parallelism.

2.1. Task parallelism

An application is often described by a block diagram in which the relations between the different algorithms, which are composing the application, are depicted. Regularly, these blocks represent algorithms which can operate concurrently on different sets of data. For example as in Figure 1, the Video Header parsing task in an MPEG-4 decoder can already decode data of a different video-object plane (VOP) that the Shape & Texture task is processing and again different from the data that the Rendering task is using within the specific time period. In this way, the different tasks from the block diagram are executed in parallel. In practice, the block diagram can serve as a starting point for extracting so-called *task-parallelism*. Not all tasks will have the same computational requirements (i.e. require the same amount of processing time). Using profiling and analysis techniques, the most computationally-expensive tasks can be identified. To increase the amount of task-parallelism, these tasks

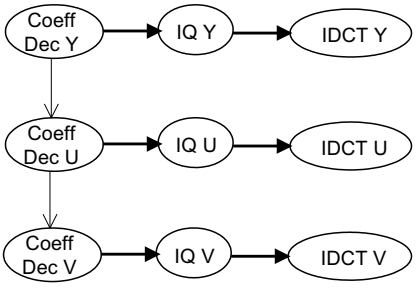
should be subdivided into smaller ones. This typically results in a chain-structured set of tasks as shown in Figure 1. As a consequence, the transformation will result in an increased throughput of the application.

Figure 1. Task-level parallelism for the texture decoding.

2.2. Data parallelism

Data-level parallelism may also be considered to increase the throughput and decrease the latency of an application. The idea behind data parallelism is to perform the same transformation on different data elements in parallel. For instance in an MPEG-4 decoder, it is possible to perform the inverse quantization in parallel for the luminance and the two chrominance planes. Figure 2(b) shows the data parallelism that can be extracted from the computation chain shown in Figure 2(a). The chrominance planes are processed in parallel in this example. It also possible to process the two chrominance planes consecutively in one chain. This will not decrease the throughput as the combined size of the two chrominance planes is smaller than the size of the luminance plane for AS-VO coding. Hence, it can be directly concluded that the computation path that processes the luminance plane is the critical path. It should be considered whether more task-parallelism can be found in this chain to increase the throughput.

(a) Original graph.



(b) Unrolled graph for each color plane component.

Figure 2. Data-level parallelism for processing of texture.

2.3. Communication granularity

A third aspect that needs to be considered when extracting parallelism from an application, is the *granularity* at which data is communicated. For example, an inverse quantization task can send data at the level of individual pixels to an inverse IDCT task, or it can send the data at the level of complete blocks or even frames. The advantage of using larger grains of data is that the communication efficiency increases due to the smaller overhead in communicating the data between the tasks. However, tasks may have to wait longer, i.e. be idle, while they are waiting for data. Choosing the correct level of granularity at which data is communicated between tasks is important to prevent tasks from waiting and avoid spending too much time on synchronization.

2.4. A strategy to extract parallelism

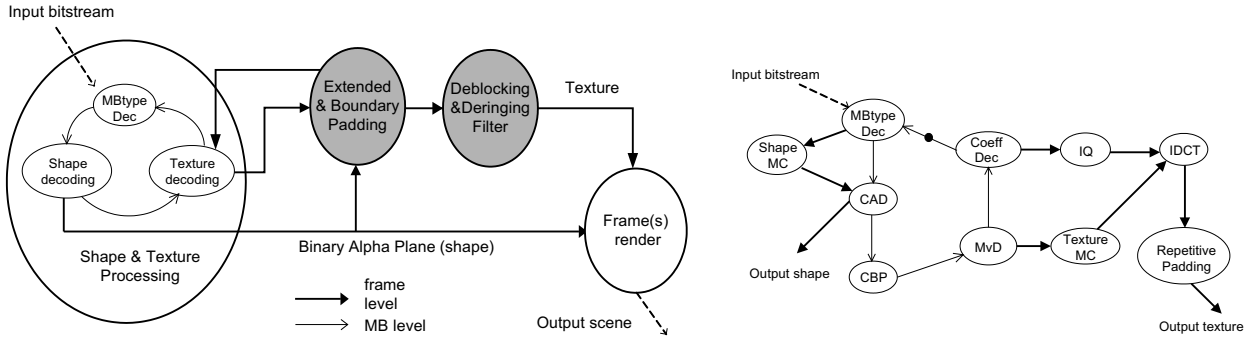
In [6], an analysis technique for identifying task-level parallelism in applications is presented. The article presents a set of concurrency measures which help a designer in making a trade-off between the three types of parallelism discussed above. Along with the concurrency measures, a strategy to extract the parallelism is now presented. As a first step, the application is profiled to identify tasks with a large execution time. These tasks are the

computational bottlenecks that should be resolved by splitting these tasks into a sequence of computationally less intensive tasks. This step introduces additional task-parallelism. The strategy continues with identifying candidate tasks for the extraction of data-parallelism. The structure of the dataflow graph is used to find these tasks. A designer can use this information to exploit data-parallelism that may be present in the application. The parallelism extraction strategy then continues with finding the right trade-off between the amount of time spent on the communication and the time spent on executing tasks (i.e. the optimal communication granularity is determined). After these steps, all potential sources of parallelism at the task-level are considered. However, the resulting dataflow graph may contain many tasks which have a low requirement for the computational resources. In other words, many tasks may be idle for large amounts of time. Dealing with a large number of this type of tasks can complicate the mapping of the dataflow graph on the computational resources. It is often preferred to recombine some of the tasks to obtain a balanced workload for the processing cores of the platform. The last two steps of the parallelism extraction strategy deal with this issue in an iterative way. Summarizing, the strategy involves the following steps:

- Identification of computational bottlenecks;
- Splitting of critical tasks;
- Exploration of data parallelism;
- Optimization of communication granularity.

3. OBJECT ORIENTED MPEG-4 DECODING

Let us first briefly discuss the details of the arbitrary-shaped video-object decoder. Figure 3 outlines a distributed version of a computation model for an AS-VO decoder. Here, we present the computation graph for the complete decoding that starts with parsing the bitstream syntax and coded data and ends with the completely reconstructed scene. The final visual scene can be composed of several VOs. In the presented graph, each task starts its execution when it has data on all inputs edges (as in Homogeneous Synchronous Dataflow Graphs). The decoding starts with the *Shape and Texture Processing* at the left side of Figure 3(a), followed by *Extended and Boundary Padding*, then applying the *Deblocking and Deringing Filter* and providing the final shape and texture data to the *Frame renderer* (the node at the right of Fig. 3(a)). The renderer is a shared task and composes the original scene from the video background-sprite image and several VOs superimposed on it.



(a) Computation graph.

(b) Shape-texture sub-graph.

Figure 3. The data-flow graph of the complete arbitrary-shaped processing at MB level.

Figure 3(b) details the shape-texture processing at the finest granularity (macroblock level). Initially, the MB type is decoded (MBtype Dec) which conditionally activates the Shape Motion-Compensation (Shape MC) task. MB decoding continues with Context Arithmetic Decoding (CAD) which provides lossless shape information. The next step in decoding is to obtain variable-length encoded Coded Block Pattern (CBP) that indicates which 8×8

luminance block(s) contain any opaque pixel. At this phase of decoding, the shape information is fully available for further processing. Texture processing consists of five conventional steps: Motion Vector Decoding (MvD), DCT Coefficients Decoding (Coeff Dec), Inverse Quantization (IQ), Texture Motion Compensation (Texture MC) and Inverse Discrete Cosine Transformation (IDCT). The new required processing for AS-VO is *Repetitive Padding*. The key property of a macroblock that contains shape information is that it contains both opaque and transparent pixels. For correct motion compensation of succeeding P- or B-VOPs, such a macroblock should assign a certain value to transparent pixels. The details of the Repetitive Padding are described in Section 4

Let us now propose an alternative for the decoding that exploits further parallelism in the decoding process. To this end, we propose a data-flow graph for a parallelism-enhanced implementation that is still compatible with the straightforward implementation of an MPEG-4 decoder. In the first step, we split the original *Repetitive Padding* into two tasks: Pre-Padding and Texture-Padding. Secondly, we propose to increase the parallelism for the *Extended Padding* by facilitating synchronized processing and thereby enabling macroblock-based pipelined processing. Thirdly, we introduce data-level parallelism to process the chrominance VO planes in parallel with the luminance VO plane. This type of parallel data processing holds for the following tasks: Inverse Quantization (IQ), Inverse Discrete Cosine Transformation (IDCT), Texture Motion Compensation (Texture MC), Repetitive Padding, Extended Padding. All three applied aspects deviate from a straightforward MPEG-4 implementation, but significantly contribute to enhanced parallel multiprocessor processing.

4. REPETITIVE PADDING

4.1. Principles of MPEG-4 repetitive padding

The transparent pixels of boundary blocks are replaced by data copies of object-boundary pixels, in order to minimize the amount of DCT coefficients after the transformation. At the decoder, the similar so-called padding steps should be performed to obtain the same reference data as in the encoder for the proper motion compensation. The boundary blocks are first padded using *Horizontal Repetitive Padding*, i.e. each sample at the boundary of the VO is replicated horizontally to the left and/or right to fill the transparent pixel row outside the VO of the boundary block. The remaining unassigned transparent horizontal samples are padded using *Vertical Repetitive Padding*, which works in a similar way, but column based. An example of repetitive padding is portrayed by Figure 4.

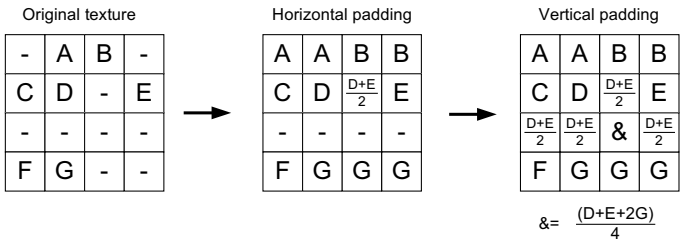


Figure 4. Principles of horizontal and vertical repetitive-padding algorithm.

4.2. Task splitting of repetitive padding

The compliant MPEG-4 standard bitstream for AS-VO has the following structure. For each time instant of the video object, the stream has encoded header information followed by shape and texture information of successive 16×16 pixel macroblocs. Due to the fact that the bitstream does not contain any markers for fast allocation of shape and texture information, the processing has to sequentially parse the original bitstream. However, we identified a possibility to increase parallelism by splitting the Repetitive-Padding task. This task should define the values of the transparent pixels for boundary macroblocks [7].

After detailed analysis of the computation flow, we found that the shape data are the most important element for repetitive padding. This hinted us to start processing of shape data immediately after the CAD task. This

results in a modified computation graph where the original Repetitive Padding is split as indicated in Figure 5. The first subtask, which is in the computation graph (Figure 5) denoted as *Pre-padding*, identifies for each pixel if its value has to be taken from the original position, or it is copied from the border of the video object. In case it is copied from one or two borders, we assign a pointer to the special buffer for padded pixel values. The padded values are computed after the texture decoding provides the texture data. This task is executed in parallel with the texture processing. The functionality of the complementary task is to provide the original functionality of *Repetitive Padding* by filling the above-mentioned pixel buffer with the real texture values and just copying the data to the output of the Repetitive-Padding task. The resulting graph of Figure **??** decreases the computation requirements and it was used for further experimental evaluation.

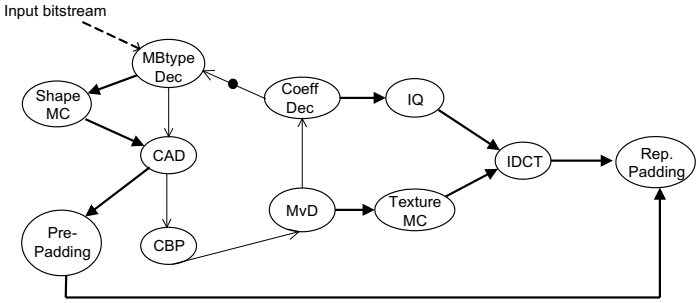
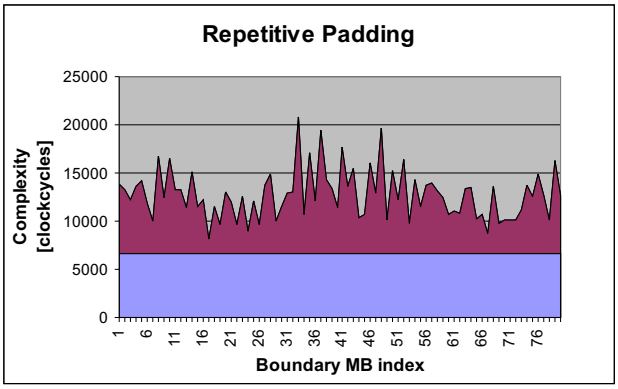


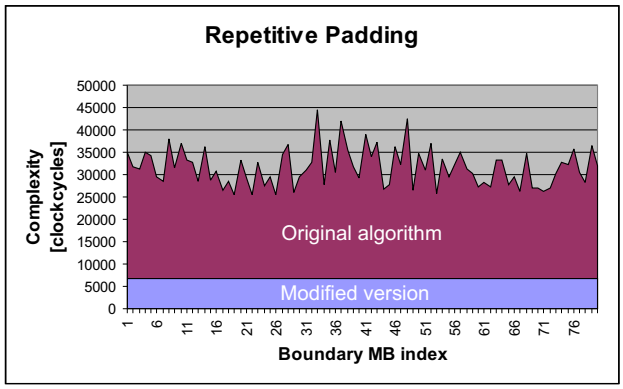
Figure 5. Modified version of shape-texture decoding with the explicit splitting of Repetitive padding into two tasks.

4.3. Evaluation of repetitive padding

Figure 6(a) compares the original computation requirements with the modified algorithm. Due to the macroblock content the decrease in computation requirements is between 11,6% and 68,5%. With the "Dancer" test sequence we achieved on the average a substantial 58.1% savings in computation effort as compared to the original algorithm. The savings become even higher by applying the data-level parallelism described in Section 6. Figure 6(b) portrays the contribution of the original sequential implementation and modified implementation with separate color-component processing. Then the optimized computation reduction varies between 64.1% and 82.7% with an average of 71.6%.



(a) Without data-level parallelism.



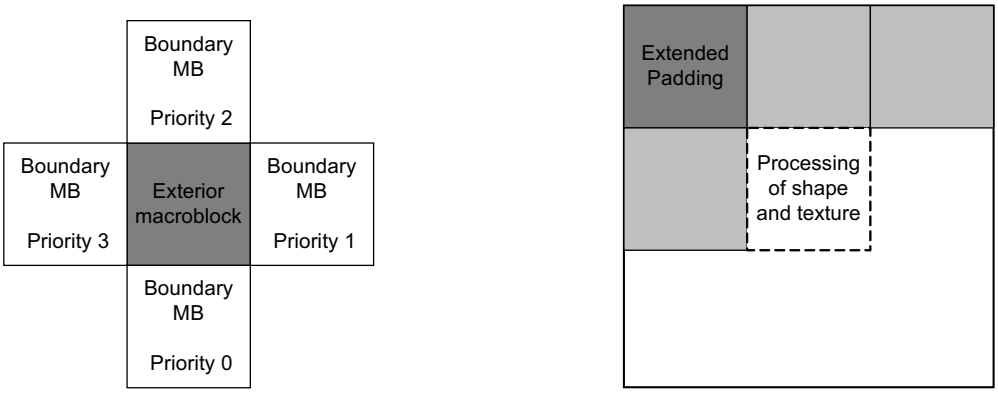
(b) With data-level parallelism.

Figure 6. Computation complexity of the original Repetitive Padding and the remaining part of the padding in the proposed modification. Data are from the execution of algorithms on ARM7 processor simulator for the "Dancer" sequence at CIF resolution.

5. BLOCK-LEVEL PIPELINING AND SYNCHRONIZATION FOR EXTENDED PADDING

5.1. Principles of MPEG-4 extended and boundary padding

Similar to Repetitive Padding, the Extended-Padding task should fill the transparent block-based parts of a VO plane with padded values. After performing the Repetitive Padding at the macroblock level, the VOP contains only fully defined MBs (Boundary and Opaque) or undefined (Transparent) MBs. The motion compensation of subsequent VOPs can require blocks from the reference VOPs that partially or fully refer to the block that has undefined texture information. Therefore, the Extended and Boundary Padding should define the complete texture information for the whole image and should be accomplished prior to starting the next motion-compensated VOP decoding.



(a) Priorities for border copying.

(b) Data dependencies for parallel processing.

Figure 7. Extended Padding principles.

The Extended Padding identifies transparent macroblocks and defines their content as follows. Exterior macroblocks adjacent to boundary macroblocks are filled by replicating the samples at the border of the boundary macroblocks. If the padded macroblock has several neighboring macroblocks, then the macroblock with the lowest priority is selected. The priority is assigned according to Figure 7(a), and the exterior macroblock is padded by replicating the horizontal or vertical border. The exterior macroblocks not having any neighboring boundary-macroblock are filled with $2^{bits/pixel-1}$ (for 8-bit luminance, for chrominance this means filing with 128). The original implementation performs this task after the complete frame was processed by previous tasks from the computation graph (see Fig. 3). Instead of processing frame by frame, the tasks can also be carried out on slices of blocks, thereby enabling smaller grains of processing.

5.2. Communication granularity optimization

By introducing synchronization tokens and a corresponding modification of the processing tasks, the granularity of these tasks can be reduced from the frame level to the macroblock level. To provide a macroblock-level task pipelining, we propose extra synchronization between the Macroblock-Type decoding task and the Extended-Padding task. The Extended-Padding task is idle until it receives the synchronization token from the Macroblock-Type decoding task. Furthermore, the extended padding can start padding of the macroblock only when the complete slice of macroblocks and the macroblock below the extended-padded macroblock is fully decoded and repetitively padded (see gray blocks following the "Extended-Padding" block in Fig. 7(b)).

With respect to the latency involved by changing the granularity of Extended-Padding processing the following can be stated. The contribution to the critical path of the whole decoding process is lowered only to performing the extended padding on the last two rows of macroblocks, as compared to the processing of the whole VO plane in the straightforward implementation.

5.3. Evaluation of the extended-padding modification

Figure 8 portrays the experimental results comparing the original algorithm with our proposed modification. We have evaluated both algorithms by executing them on an embedded ARM7 processor simulator. The simulator provides clock-cycle true information and the compiled code is identical to the one used in the final mapping. The test sequences were chosen from the MPEG group test sequences. We present in detail the results of the "Singer" test sequence.

The contribution of the original implementation of extended padding to the overall latency was on the average 537.9 kilo clock cycles per VOP containing 60 macroblocks. After the modification of the algorithm the resulting contribution was on the average only 75.3 kilo clock cycles and by introducing data-level parallelism and performing the individual color-component processing in parallel, it further decreased to 65.8 kilo clock cycles. This represents only 12.2% of the original contribution to the overall latency. This results proves the attractive potential of our approach for modification towards enhanced parallelism.

In general, the granularity depends on the fraction that the last slice occupies with the respect to the whole VOP size. The fraction is varying between 5% of the VOP size ("singer" sequence) - 50% of the VOP size ("fish" sequence) for available test sequences. The sequence can have the aforementioned fluctuation even within the same video sequence. However, for most of processed sequences it is below 25%.

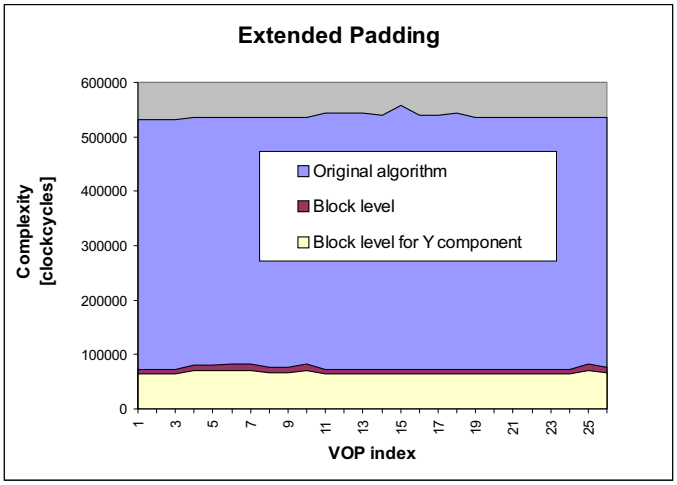


Figure 8. Experimental results of modified Extended Padding on the "Singer" test sequence.

Additionally, the effect of changing the communication granularity from the original VOP-size granularity towards the macroblock size also decreases the requirements on the internal buffer sizes. The straightforward implementation assumes that data are shared between processing tasks. Therefore, the straightforward implementation on a multiprocessor architecture requires an internal buffer for the whole frame. The parallel implementation requires the processed data to be fully stored in the internal memory of individual processors. In our new approach, the required buffer size is minimized to just one slice of macroblocks plus one macroblock to perform the MPEG-4 compliant implementation.

6. DATA-LEVEL PARALLELISM

To further increase the throughput of the decoder, we also employ data-level parallelism. For each color component, we instantiate a separate pipeline of tasks, so that the execution of the Inverse Quantization, Inverse DCT, etc., can start as soon as the decoding of coefficients for the luminance plane has finished. The subsequent processing of Cr and Cb chrominance components runs in parallel with processing of the luminance (Y) plane. Due to the smaller size of the chrominance planes, the splitting of chrominance parts to individual component

processing is not decreasing the length of the critical path of the computation graph (no latency decrease), but it may be useful for better utilization of computation resources.

The critical path contains the tasks for the original bitstream parsing for shape decoding and texture coded data. The decoding of texture information is most complex for the luminance component, due to the fact that the AS-VO MPEG-4 standard currently supports only the 4:2:0 sampling standard. Figure 9 portrays the complete graph that contains also the proposed modifications as discussed in Section 4 and Section 5. The critical path through the computation graph that effects the latency of the complete decoding has the following two parts:

- Bitstream parsing: *Context Arithmetic Decoding (CAD), Coded Block Pattern (CBP), Motion Vector decoding (MvD), DCT coefficients decoding (Coeff Dec Y)*;
- Texture processing for the luminance component : *Inverse Quantization (IQ), Inverse DCT transformation (IDCT), Repetitive Padding Y, Extended and Boundary Padding Y, Deblocking and Deringing Filter Y.*

The complexity of Pre-Padding 16×16 and Texture Motion Compensation (Texture MC Y) is significantly lower than the remaining tasks that are in the computation chain before the Repetitive Padding Y task in the critical path. It should be noted that the critical path remains as presented only in the case that the final mapping schedules the same type of the tasks for different color components on the same type of processing cores for the same clock frequency.

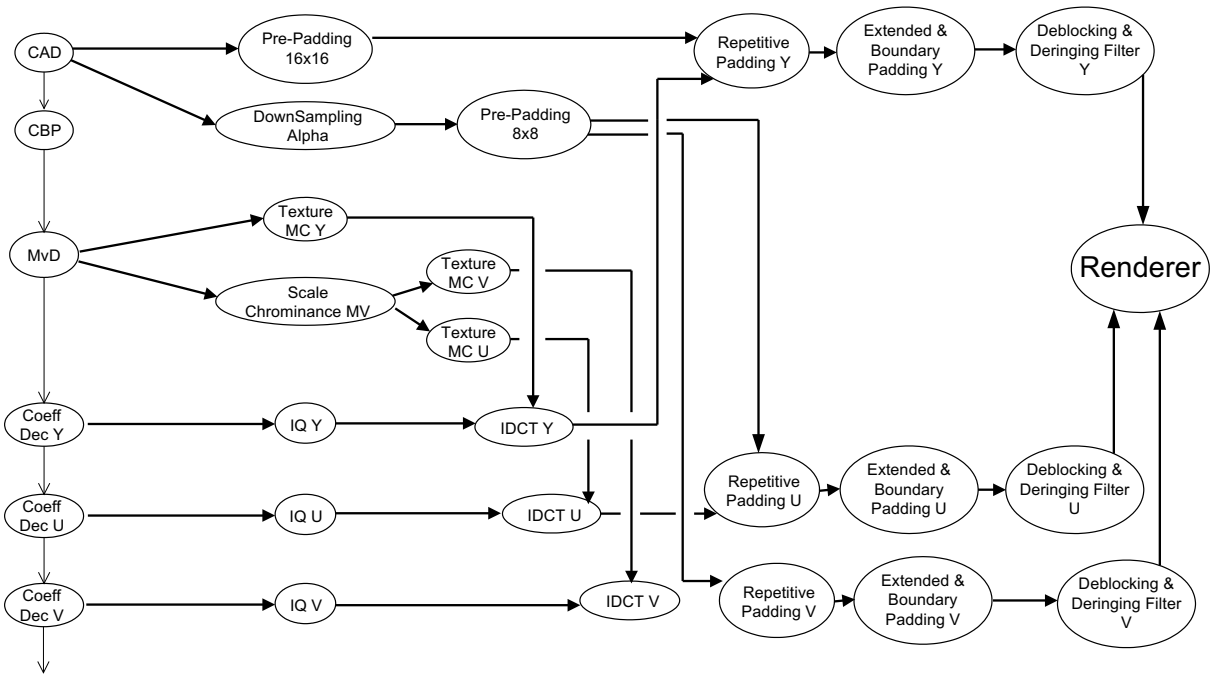


Figure 9. AS-VO compliant MPEG-4 decoding computation graph with depicted task-level and data-level parallelism.

7. CONCLUSIONS

We have shown that a parallelism-enhanced implementation of arbitrary-shaped MPEG-4 video-object coding on a multiprocessor platform offers significantly higher throughput rates than a conventional implementation. This work has presented techniques to increase the task-level and data-level parallelism in the MPEG-4 decoder. We have evaluated our algorithms on a clock-cycle true simulator of a multiprocessor architecture using the ARM7TDMI processors.

We have identified that the repetitive padding was the primary candidate for introducing task-level parallelism. We have proposed the splitting of the task to the filtering of the shape data and leaving only the copying of pixel values at the original place in the critical path. This decreases the original computational complexity of the repetitive padding to 40.9% of the original task. By applying additional parallel color-component processing it drops to only 28.4% on the average.

The modification of Extended-Padding algorithms has two major impacts. The first is in the reduction of the overall decoding process latency. The standard describes the extended padding as postprocessing after the whole VO plane is fully decoded. By changing the granularity to block level and introducing the a new synchronization mechanism, that is aware of having sufficient data for processing, we obtained only 12.2% of the original algorithm complexity. Additionally, the task-specific buffering is maximally one slice of macroblocks plus one macroblock of the image resolution. For example for CIF resolution, this involves only 5.8% of the original internal buffer.

The results in this paper can be seen as a case study for extracting parallelism in advanced multimedia applications. For this reason, we have provided a strategy to exploit the parallelism that is generally available in this class of algorithms. In the study, the identification of the critical path was carried out manually. We envision that such tasks can be conducted by automated tools for application analysis. In order to realize this, the detailed knowledge of data availability in the computation flow-graph is indispensable.

REFERENCES

1. ISO/IEC 14496-2:199/ Amd 1:2000, *"Coding of Audio-Visual Objects - Part 2:Visual, Amendement 1: Visual Extensions"*, Maui, December 1999.
2. P. Kauff and K. Schuur, "Shape-adaptive DCT with block-based DC separation and DC correction," in *IEEE Transactions on Circuits and Systems for Video Technology, No. 3, Vol. 8*, pp. 237–242, June 1998.
3. M. Pastrnak, P. Poplavko, P. de With, and D. Farin, "Data-flow timing models of dynamic multimedia applications for multiprocessor systems," in *4th IEEE Int. Workshop System-on-Chip for Real-Time Applications (SoCRT)*, 2004.
4. P. Li, B. Veeravalli, and A. Kassim, "Design and Implementation of Parallel Video Encoding Strategies Using Divisible Load Analysis," in *IEEE Transactions on Circuits and Systems for Video Technology, No. 9, Vol. 15*, pp. 1098–1112, September 2005.
5. H.-C. Fang, "Parallel Embedded Block Coding Architecture for JPEG 2000," in *IEEE Transactions on Circuits and Systems for Video Technology, No. 9, Vol. 15*, pp. 1086–1097, September 2005.
6. S. Stuijk and T. Basten, "Analyzing concurrency in computational networks.," in *MEMOCODE 2003, 1th International Conference on Formal Methods and Models for Co-Design, Proc.*, pp. 47–48, IEEE, 2003.
7. N. Brady, "MPEG-4 Standardized Methods for the Compression of Arbitrarily Shaped Video Objects," in *IEEE Transactions on Circuits and Systems for Video Technology, Ser. 8, Vol. 9*, pp. 1170–1189, December 1999.