ADAPTIVE DECODING OF MPEG-4 SPRITES FOR MEMORY-CONSTRAINED EMBEDDED SYSTEMS

Milan Pastrnak[a,b], Dirk Farin[b], Peter H. N. de With[a,b]

[a]LogicaCMG Nederland B.V., PO Box 7089, 5605JB Eindhoven, The Netherlands
[b]University Technol. Eindhoven, PO Box 513, 5600MB Eindhoven, The Netherlands
Email: M.Pastrnak@tue.nl

Abstract

Background sprite decoding is an essential part of object-based video coding. The composition and rendering of a final scene involves the placing of individual video objects in a predefined way superimposed on the decoded background image. The MPEG-4 standard includes the decoding algorithm for background image decoding, but this algorithm is not suitable for implementation on a memory-constrained platform. In this paper we present a modification of the decoding algorithm that decodes MPEG-4 sequences while fulfilling the requirements of a memory-constrained multiprocessor system with only 17% extra overhead of computation. Our algorithm reduces the memory cost of such decoding with a factor of four. Additionally, our algorithm offers the possibility of high level data parallelism and consequently contributes to an increase of throughput rate.

1. INTRODUCTION

Object-oriented video coding enables new features for portable multimedia devices. This type of video processing requires a considerable amount of memory to render the final scene. The memory requirement contrasts with the minimization of resource usage in portable devices. Recent studies revealed that a multiprocessor system-on-chip (MP-SoC) [1] or a cell-processor system [2] provide high computational resources with low power consumption. The proposed CELL processor implementation poses a limitation on other types of resources such as memories and communication resources.

Multiprocessor architectures are described as a set of processing tiles connected together via a bus or a network. The efficient implementation of streaming applications on such platforms intrinsically ask for partitioning of complex algorithms into smaller subtasks, which can fit to the resources of a processing tile.

Pipelined execution of individual subtasks results in complete functionality of the original algorithm. The CELL processor meets computational requirements of complex state-of-the-art applications such as MPEG-4 coding [3]. However, the limited size of local memory requires a modification of the MPEG-4 sprite-decoding algorithm. The implementation of a first-generation CELL processor consists of a 64-bit power processor element (PPE) and its L2 cache, multiple *synergistic processor elements* (SPE). Each SPE has its own local memory (LS) [4]. Synergistic aspects in processing are that individual processing tiles are directly connected and perform pipelined execution of distributed algorithms. As shown in Figure 1, an SPE unit contains four 64 kByte memory blocks.

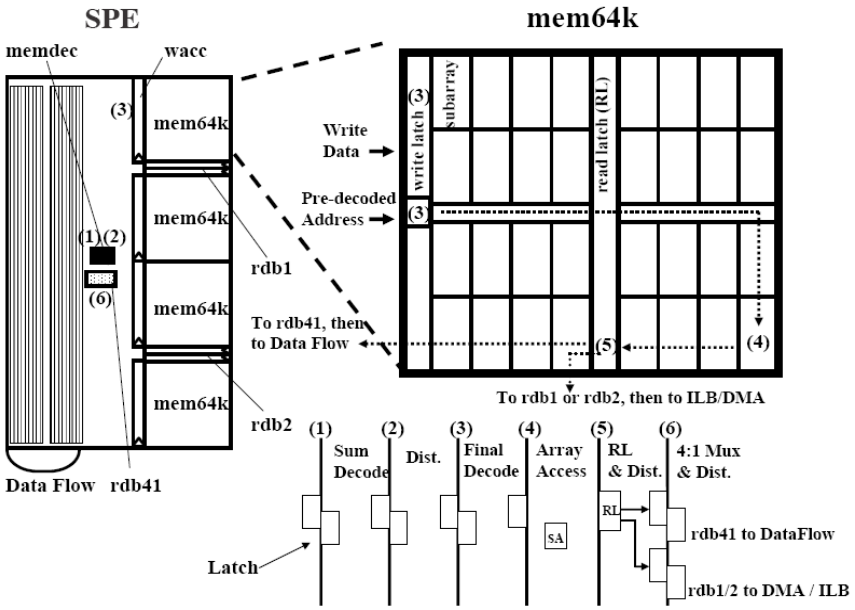

Figure 1: Local memory organization in SPE (taken from [4]).

A major cost problem of the MPEG-4 decoding algorithm is the buffering of the decoded scene background. The Main Visual Profile Level 2 (MP@L2) of the MPEG-4 standard bounds the maximum size of reference image for the sprite reconstruction to 1584 MacroBlocks (MBs) at CIF resolution (a single video picture is 396 MBs) [5]. The target processing-tile memory limitation of 256 kByte can handle decoding of rectangular video pictures or arbitrary-shaped video object decoding, but it cannot internally buffer the complete reference sprite image. In this paper we present a new algorithm that decodes MPEG-4 compliant sequences which satisfies the target platform constraints on memory (MP@L2).

Furthermore, it introduces the data-level parallelism.

The paper is organized as follows. Section 2 presents the sprite reconstruction principle and addresses MPEG-4 standard decoding. Section 3 gives our new decoding algorithm while Section 4 provides details on the corresponding data structures. Section 5 describes the experiments and Section 6 concludes the paper.

2. SPRITE RECONSTRUCTION AND MPEG-4 DECODING

Object-oriented coding in the MPEG-4 standard enables individual processing of foreground objects and the scene background (also called sprite). The background sprite image is used for the reconstruction of the scene background for a set of succeeding frames. The reference sprite is constructed by merging several views into one large image that is further encoded and transmitted.

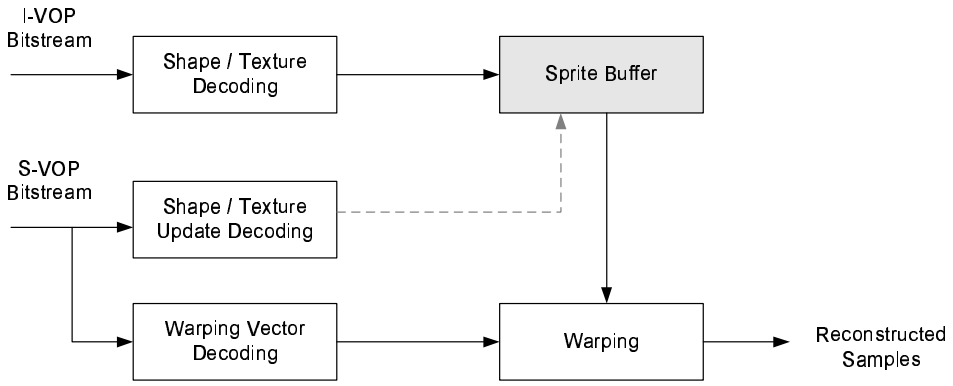

Figure 2: MPEG-4 decoding of background sprite.

The sprite warping (see Figure 2) generates the actual camera view that is used in the renderer for the scene composition. The warping transformation is modeled as a mapping between the decoded reference sprite plane and the actual view plane. This mapping is described by the following MPEG-4 GMC formulas:

$$x' = \frac{m_{00} \cdot x + m_{01} \cdot y + m_{02}}{p_x \cdot x + p_y \cdot y + 1}, y' = \frac{m_{10} \cdot x + m_{11} \cdot y + m_{12}}{m_{20} \cdot x + m_{21} \cdot y + 1}, \quad (1)$$

where x', y' are the transformed display coordinates and x, y denote the original coordinates for the video texture in the sprite buffer. The global motion parameters m_{ij} in the above equations are calculated during encoding.

The MPEG-4 static sprite coding provides the reconstructed scene background for the actual video scene and may require substantial memory. In the

MPEG-4 standard, the sprite decoding consists of four major steps: shape/texture decoding, buffering of the complete sprite, decoding of the warping vector and geometrical warping (Fig. 2). The I-VOP (Intra-coded Video Object Plane) contains coded data for shape and texture of the referenced sprite. An S-VOP (Sprite Video Object Plane) contains coded warping vectors.

The luminance, chrominance and grayscale alpha data of a sprite are stored in two-dimensional arrays. The width and height of the luminance array are specified by the syntax parameters `sprite_width` and `sprite_height`, respectively. These planes are stored in the so-called *Sprite Buffer* and are used as references for the actual camera view reconstruction. The decoding of the current view background applies to the previously described warping process.

3. NEW SPRITE DECODING ALGORITHM

In the new algorithm, we focus on the partitioning and optimal buffering of sprite data, because it is intrinsically a costly function due to the high amount of involved MBs. The large latency of accessing data in a processor network system, specifically for an off-tile memory location, requires a complete modification of the original MPEG-4 standard decoding process. The primary difference is in the way how MB data are stored. The original approach keeps the whole reference image in an uncompressed form. We propose to keep the reference sprite image in compressed form and decode only the part that is required for the reconstruction. As a consequence, the decoding process is decomposed into four steps: (1) the

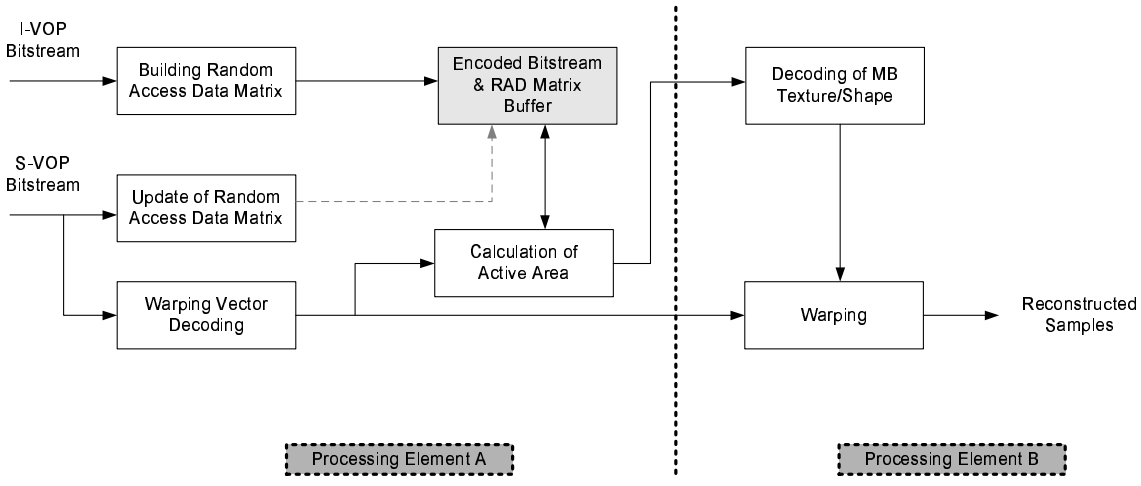

Figure 3: New decoding of background sprites and corresponding task-to-processor assignment.

construction of an information matrix for random access of MB data, (2) decoding of warping vectors, (3) decoding of the required MB texture data, and (4) warping of the actual sprite.

Figure 3 gives the block scheme of the new approach. Let us simplify the decoding problem to the situation when the bitstream contains only one fully encoded sprite image and it is followed by a number of S-VOPs without updates of texture/shape information. The decoding of the actual background involves the following steps:

1. Parsing of the I-VOP bitstream and construction of the *Random Access Data Matrix* (details are given in the next section).
2. Decoding of warping vectors.
3. Calculating of the bounding box that defines the actual referenced sprite view.
4. Fetching and decoding of MB data that was not available in the previous referenced image for the actual bounding box.
5. Recalculating of the image origin.
6. Warping of the background image.

In Figure 3, we indicate by a vertical dotted line, that the algorithm has to be split at least into two parts and mapped onto two processing elements. This split fulfills the SPE constraints on the memory size (256 kBytes). More analysis and proof that the split ensures the decoding without problems is given in Section 5.

4. CONSTRUCTION OF MB DATA MATRIX FOR RANDOM ACCESS

The MPEG-4 compressed bitstream does not contain markers for accessing image data at a macroblock granularity level. At the first stage, we construct the *MB access matrix* for access to the bitstream of MB compressed data. The texture processing is performed in three steps: DC/AC prediction based on the previous neighboring blocks, decoding of DCT coefficients and Inverse DCT transformation. To provide random access to the MB data, two approaches are available. First, storing data in matrices of 8×8 DCT coefficients. Second, buffering for each MB DC/AC predictors and postponing the decoding of DCT coefficients until the moment that the MB is required for the warping process. Since the target matrix for storing DCT coefficients has the same size as the complete sprite, we have to adopt the second approach to save memory.

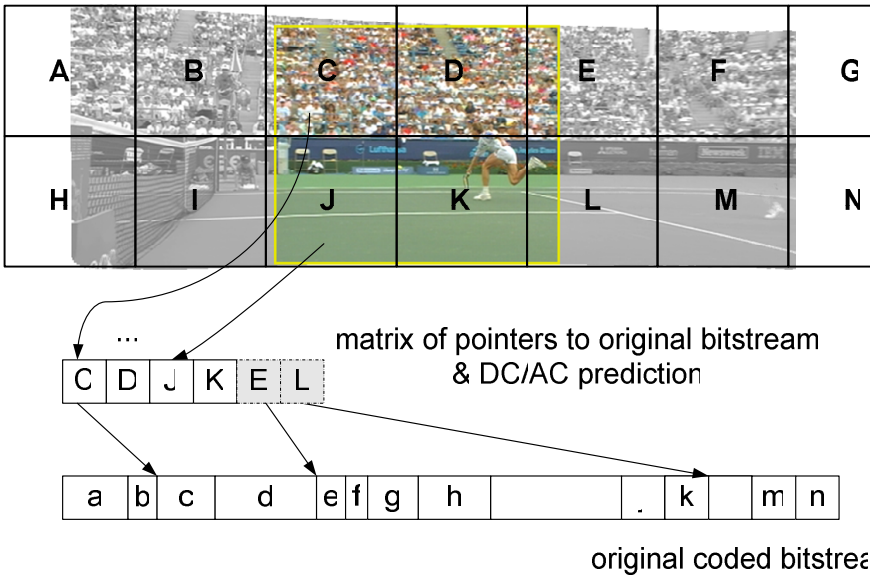

Figure 4: Data organization for the new sprite decoding algorithm.

Figure 4 portrays the data organization for the new sprite decoding algorithm. Note that memory blocks E, L are dotted, because these blocks require the decompression of encoded e, l MB data. The calculation of the active area (Step 5 in the algorithmic description) indicates the needs for these MBs in the further warping process. The algorithm looks to the matrix of pointers to identify the positions in the original coded bitstream buffer.

A major gain in the use of pointers for the decoding process is achieved during the reconstruction of the first frame(I-VOP). The MPEG-4 standard algorithm performs DC/AC prediction, DCT coefficients decoding and IDCT on the complete reference image. Instead, we perform first two simple processing stages (without buffering of DCT coefficients) on the whole image and only the last step performs actual full decoding but on a restricted image size. This last step introduces an extra overhead caused by the redundant decoding of DCT coefficients for the first image. However, this overhead can be removed if we obtain the warping vectors for the first image *a priori* to texture parsing. When using this modification, the reduction of the number of MBs for the first image IDCT transformation is 76%. Additionally, we can speedup the texture processing by instantiating subsequent DCT coefficients decoding and IDCT transformation with parallel MB decoding.

5. EXPERIMENTS

We have implemented the new version of the MPEG-4 sprite decoder. Figure 5 shows the results of decoding the well-known MPEG "tennis player" sequence. The amount of required MBs for decoding processing is illustrated at the left of Figure 5 by the (noisy) bold line at the bottom. The decoding of the background reference sprite requested for the first image processing of the complete reference image (in our case 225 MBs) is shown at the top of the same figure. For this test sequence, the maximum number of uploaded MBs for the decoding of the active sprite area was 41 MBs.

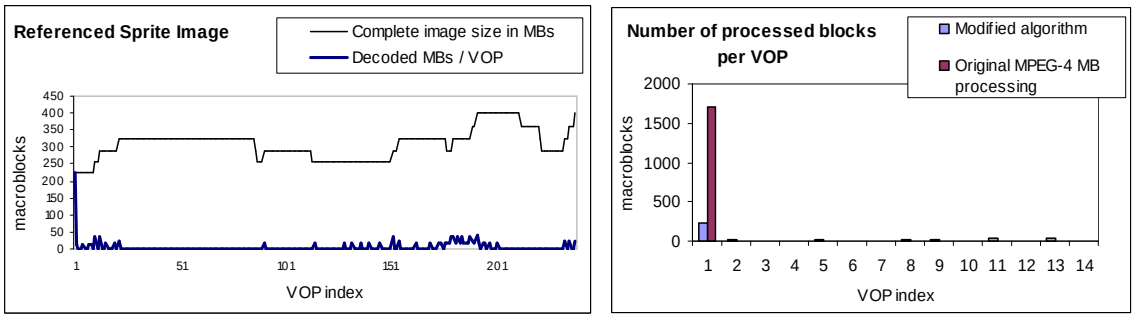

Figure 5: Left: Processing of "tennis player" sequence by the new algorithm. Right: Comparison of the original and modified size of the buffer.

At the right of Figure 5, we show the required buffer sizes in terms of MBs, and compare the original and the new algorithm. The original decoding requires 1701 MBs stored in an uncompressed way (1584 is the maximum, but multiple 16-pixel MB grid alignments increases this number to 1701). The maximum size of the reference sprite was never higher than 400 MBs. It was found that the original algorithm requires 4.25 times more memory than our new algorithm. The figure also shows that our algorithm requires small bursts of blocks for decoding during the scene, but these bursts do not accumulate to a significant number, so that the total remains much smaller than the big burst required for the original algorithm.

However, compared to the original algorithm, we found that our proposal has the disadvantage of occasional extra decoding of MBs for special background movements. For example, this occurs when the active window leads to the release of decoded MBs and after some frames requires the same picture data leading to repetitive decoding of already released MBs. This disadvantage depends on the camera motion in the background. During our experiment, we observed that the repetitive decoding involved 17% of extra MBs decoding. The amount of memory

to store the original bitstream was 37.440 kBytes and the resulting random-access data matrix needed 7 kBytes of memory (fixed small memory space).

6. CONCLUSIONS

We have proposed and implemented a new MPEG-4 background sprite decoding algorithm. The new algorithm features minimum use of local memory in the processing elements of the target processor network. A second feature is that it is based on constructing a special information matrix for random access to MBs data, which enables independent MB processing. This potentially increases the level of parallelism to perform the decoding on more processing elements at a higher speed (for a VOP). It was shown that the required memory reduces with the factor of 4.25, with only 17% computation overhead due to repetitive MB decoding for the complete sequence.

REFERENCES

[1] M. Berekovic, H.-J. Stollberg, and P. Pirsch, "Multicore System-On-Chip Architecture for MPEG-4 Streaming Video," in *IEEE Transactions on Circuits and Systems for Video Technology*, August 2002, vol. 12 of *8*, pp. 688–699.

[2] B. Flachs et al., "A Streaming Processing Unit for a CELL Processor," in *Proceedings of IEEE International Solid-State Circuits Conference*, February 2005, pp. 134–135.

[3] M. Pastrnak, P. Poplavko, P.H.N. de With, and J. van Meerbergen, "On Resource Estimation of MPEG-4 Video Decoding for a Multiprocessor Architecture," in *Proceedings of PROGRESS 2003*, October 2003, pp. 185–193.

[4] S.H. Dhong et al., "A 4.8GHz Fully Pipelined Embedded SRAM in the Streaming Processor of a CELL processor," in *Proceedings of IEEE International Solid-State Circuits Conference*, February 2005, pp. 134–135.

[5] Dirk Farin, Peter H. N. de With, and Wolfgang Effelsberg, "Minimizing MPEG-4 Sprite Coding-Cost Using Multi-Sprites," in *SPIE Visual Communications and Image Processing, Vol. 5308/1*, January 2004, pp. 234–245.