

A Modified Spiral Search Algorithm and its Embedded Hardware Implementation

Nikolaos Kroupis, Minas Dasygenis, Dimitrios Soudris, and Antonios Thanailakis

Abstract—One of the most growing areas in the embedded community is multimedia devices. Multimedia devices incorporate a number of complicated functions for their operation, like motion estimation. A multitude of different implementations have been proposed to reduce motion estimation complexity, such as spiral search. We have studied the implementations of spiral search and identified areas of improvement. We propose a modified spiral search motion estimation algorithm, with lower computational complexity compared to the original spiral search. We have implemented our algorithm on an embedded ARM based architecture, with custom memory hierarchy. The resulting system yields lower energy consumption and higher performance, with some penalty in image quality, compared with the original spiral search algorithm.

Keywords—Spiral Search, Motion Estimation, Embedded Systems.

I. INTRODUCTION

MULTIMEDIA applications, such as teleconferencing, video transmission and video streaming, have become increasingly common nowadays. The most important factor is the increasing need for wireless systems or portable multimedia applications. H.26L is a high-performance video coding standard for video transmission into a low bit rate channel [1], widely used in portable multimedia terminals. H.26L involves computationally intensive parts of Motion Estimation (ME) kernels and requires many off-chip memory accesses. It has been reported that the ME algorithms are responsible for up to 60% of the total power consumption of an application because they require a lot of memory accesses. For this reason, the performance and energy consumption optimization of multimedia algorithms has been the subject of extensive research [2], [3], [4].

ME can be performed by various methods. One of them is Full Search (FS) which is a computationally-expensive algorithm, but guarantees finding the blocks that yield the minimum cost value. Due to the high computational complexity, alternative search methods are desirable. ME algorithms with lower computational complexity are the

Hierarchical Search (HS) [5], the 3-Step Logarithmic Search (3SLOG) [6], the Parallel Hierarchical One-Dimensional Search (PHODS) [6] and the Spiral Search (SS) [2].

The objective of our work is the SS algorithm, which is similar to the FS algorithm. Even though it is less intensive computationally than FS, it continues to have a high computational complexity, making it difficult to implement it in portable devices. We address this problem by presenting a new modified form of the SS algorithm, which improves both execution time and energy consumption. Our modified algorithm performs the search in a fashion similar to the SS algorithm, but uses a step one pixel bigger than the original SS. This results in omitting the comparison of roughly half the number of the blocks, thus decreasing the complexity of the algorithm. Furthermore, for the additional optimization of the system, the Data Transfer and Storage Exploration (DTSE) methodology of IMEC [7] and, in particular, data reuse transformations are applied. In addition, we achieve further energy and performance gains by designing a memory hierarchy that decreases the number of off-chip memory data transfers to a minimum. The algorithmic modifications involve, however, a reduction in the video quality, which nevertheless remains in the level of tolerance, as proven by experimental results.

II. SPIRAL SEARCH MOTION ESTIMATION ALGORITHM AND MODIFIED ALGORITHM

The SS algorithm compares a block of the previous frame with blocks of the current frame inside the search area. The search area is centered at the same coordinates as the previous frame block. The search process on the search space moves outward spirally (Fig. 1).

The SS algorithm consists of a number of nested loops (Fig. 2). The first two external loop indexes divide the frames into $B \times B$ size blocks. The loop with index l refers to the spiral which we are located, while the k -loop describes the location of the block within the spiral. The distance criterion SAD is computed from all pixels of the $B \times B$ block. If the value of the SAD criterion is higher than the value of the previous best matching block, there is a break in the m -loop execution, because a better match cannot be found in this block.

During the study of the motion estimation in the SS algorithm, we observed an interesting characteristic. The criterion value (SAD) decreased approaching the block with

Manuscript received July 15, 2005. This work is partially supported by a scholarship from the Public Benefit Foundation of Alexander S. Onassis.

The authors are with VLSI Design and Testing Center, Department of Electrical & Computer Engineering, Democritus University of Thrace, 67100. Xanthi, Greece ({nkroup, mdasyg, dsoudris, thanail}@ee.duth.gr).

the best match in the previous frame. Furthermore, it was set to zero or was too small when the suitable block was found. Therefore, we considered that we could achieve a reduction in the algorithm's processing time if we decreased the number of blocks for which the SAD value was calculated in each spiral. Specifically, we found out during the observations that for these two frames, as we moved away from the best match, the SAD value was bigger than 450. On the other hand, there was no big difference in values of SAD between two blocks located 1 pixel away from each other. We can explain this as follows: the luminance value of a pixel differs a little from its neighbor's pixels. In addition, the SAD value for the nearby blocks is small, such that if the SAD of (x, y) block is high ($V_{TH1} < SAD < V_{TH2}$) then also the $(x+1, y)$ will be high. Thus, we skip the calculation for this block and we compare the $(x+2, y)$ block, yielding a step equal to 2. Similarly, if the SAD of (x, y) block is much higher ($SAD > V_{TH2}$) we skip the next two blocks and we compare the $(x+3, y)$ block.

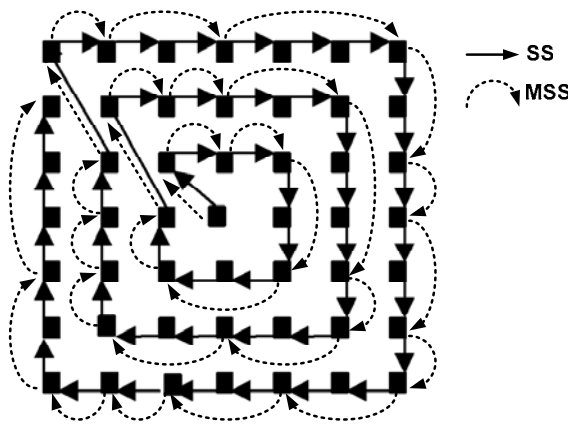


Fig. 1 MSS uses less comparison blocks compared with SS, using steps with distance one, two or three pixels

```

/* Selecting block of the frame */
for(x=0;x<N/B;x++)
for(y=0;y<M/B;y++)
{
/* Spiral index */
for(l=1; l<= MAX_Spiral_index)
/* Block index into the Spiral */
for(k=1; k<=MAX_Number_Block_in_Spiral;
{
for(m=0;m<B;m++)
for(n=0;n<B;n++) /* For all pixel of the block */
{
SAD criterion calculation
if(SAD > SAD_previous) break; /* SS only */
}
step++;
* Additional commands in MSS
* if(SAD > Vth1) step++;
* if(SAD > Vth2) step++;
Calculate next k and l
}
}

```

Fig.2 The pseudo-code of SS algorithm, the MSS algorithm has additionally the command in the box

Our goal is to reduce the number of the examined blocks in the search space. Instead of using always the 1-pixel step to move the 16×16 pixels block (as in SS), we use varying pixel

steps for the search pattern (our MSS). Pixel steps are defined according to the current value of the SAD criterion. We compute the pixel step by using two thresholds V_{TH1} and V_{TH2} ($V_{TH1} < V_{TH2}$), as follows:

- if $SAD > V_{TH2}$, we have a step equal to 3 pixels
- if $V_{TH1} < SAD < V_{TH2}$, we have a step equal to 2 pixels
- if $SAD < V_{TH1}$, we have a step equal to 1 pixel

If our current SAD value is greater than V_{TH2} , then our search is moved to the block located 3 pixels further. If the SAD value is between the two thresholds, then our search space moves to the block that is 2 pixels further. In the last case, where the SAD is lower than every threshold, we have a search movement of the block that is 1 pixel. In other words, MSS provides the possibility of moving either by 3 or 2 or 1 pixel(s) during the search for the best match. In MSS we have found out by experiments, that for $V_{TH1}=300$ and $V_{TH2}=450$ we achieve a nice video quality with low computational complexity. The pseudo-code of the MSS algorithm (Fig. 2) is similar with the SS algorithm. The additional condition checks, for the two thresholds values, are inserted after the calculation of SAD criterion. The additional commands are located at the "*" position of the pseudo-code and they implement the varied step size. Using higher threshold values the video quality will deteriorate but the complexity will be lower. This is clearly a trade-off for the designer between the video quality and performance. We should also notice that for low bit-rate applications the video quality is not the most important factor.

III. OPTIMIZATION TRANSFORMATIONS

In multimedia applications the energy consumption related to memory transfers is the dominant factor in total energy cost [7]. This motivated us to find an efficient method to reduce it. For this purpose we created a data memory hierarchy specific to every application that exploits the temporal locality of the data, by reusing them. If there is a sufficient reuse of the data, it can be advantageous to copy the data that are used frequently to a smaller memory, such that for the following usages a data element can be read from the smaller memory instead of the larger memory. The smaller memories require less energy per access and as a result they decrease the total energy consumption. However, we must first determine the specific data sets which are heavily re-used in a short period of time and therefore are appropriate to be placed in a separate memory. Consequently, we performed an exhaustive data reuse exploration of the application's data over time. In order to explore the memory hierarchy, algorithmic optimizations are required.

We optimized the SS and MSS algorithm using algorithmic transformations for energy consumption and performance, according to the DTSE methodology [7], for 176×144 QCIF input video sequences. One type of optimizations applied is the global loop transformations such as loop merging and tiling, which are used to increase the regularity of the loop structure of the application. We also applied data reuse transformations in order to reduce the number of off-chip data

memory transfers. In particular, the SS algorithm has two data arrays (*current_frame* and *previous_frame*), which account for most of the data accesses. Using profiling, we discovered that the two array signals of 176×144 pixel size have the highest number of data accesses. These blocks carry two subsequent frames of the video sequence and are used in some processing loops, resulting in many different copy candidate implementations.

Using a data reuse detection technique [8] we found out that in the SS algorithm, three arrays are possible copy candidates: (i) *prev_line* of size 176×48 , (ii) *RW* of size 48×48 and (iii) *CB* of size 16×16 . Of course different sizes of video sequences result in different *prev_line* size. Though, the size of *RW* and *CB* remain the same, because these copy candidates depend on block size only. The purpose of the three arrays is to copy part of the data from the off-chip memory of current and previous frames. Thus, a lot of different memory hierarchies can be designed combining these arrays. For this reason, an exhaustive exploration between the multitudes of different memory hierarchy implementations is required. Exploring all the possible different memory hierarchy structures, based on the three block candidates, we discovered 17 different memory hierarchies. These hierarchies were produced by all the different combinations of the three copy candidates. Every data reuse transformation corresponds to a different memory hierarchy, such that 17 transformations exist (Fig. 3).

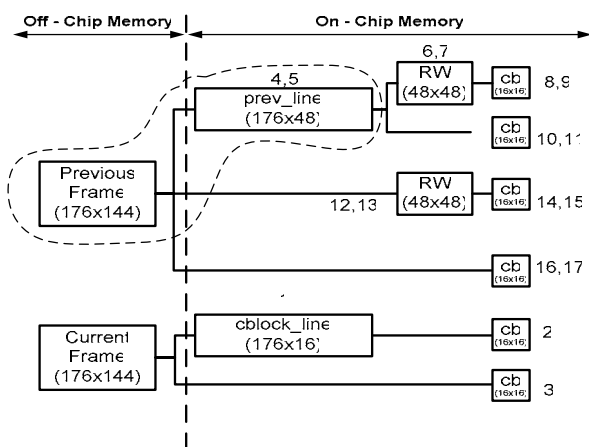


Fig.3 An exploration is required to pinpoint the optimum memory hierarchy. The numbers indicate different memory hierarchies

The number on top of every copy candidate (Fig. 3) identifies the current data reuse transformation. The first number corresponds to the transformation with data reuse from the higher level, while the second number corresponds to the transformation with data reuse between the higher level and the same block. Our goal is to move accesses from off-chip memory to on-chip layers. When the dominant part of the data memory accesses are happening to the memory blocks of small size, closer to the processor core, the energy consumption is reduced and the performance is increased. These blocks carry copies of off-chip data. Taking into consideration the on-chip size specification, we have to carefully select the on-chip

arrays in order not to exceed it. On-chip size specification depends on the selected processor. We decided to use a low power embedded processor ARM920T to implement the SS and MSS algorithms.

The ARM920T processor has a scratchpad memory of 16 Kbytes. We assigned the array signals *prev_line*, *RW* and *CB* with sizes 8448 bytes, 2304 bytes and 256 bytes respectively to the on-chip scratchpad memory. Taking into consideration the previous facts, we propose an embedded system architecture with 64 KB off-chip and 16 KB on-chip data memory (Fig. 4), to implement the SS algorithm.

The proposed architecture is optimized for the 176×144 pixels QCIF video sequence. In order to use video sequences with larger size, the *RW* and *CB* remain the same size, but the *prev_line* will have different size. This means that the on-chip memory depends on the frame size. The sizes of *previous_frame* and *current_frame* are direct related with the size of input video sequence, too.

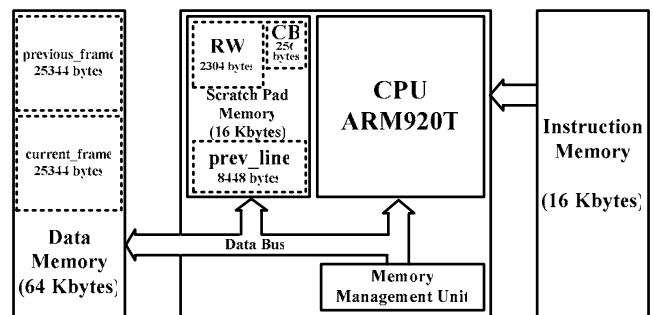


Fig. 4 The architecture of the embedded system is based on ARM core with 16KB on-chip memory

The system consists of an external data memory, instruction memory and the processor unit. Applying the DTSE Methodology, the number of memory accesses to the background memory *current_frame* and *previous_frame*, is reduced. *previous_frame*, *RW* and *CB* are placed in on-chip memory, carrying copies of off-chip data. Thus a significant number of memory accesses are shifted from off-chip memory to on-chip memory.

Since the energy consumption per access to the external (off-chip) memory is much higher compared with on-chip accesses, the proposed system architecture has very low energy consumption compared to the original architecture, as our estimations reveal.

IV. EXPERIMENTAL RESULTS

The MSS algorithm has fewer block-matching comparisons. This does not yield always the best matching and does not always result in the optimum motion vectors. For this reason, the reconstructed sequence may have lower video quality compared with the sequence using the SS algorithm. Measurements using a 200 frames video sequence, called "Akiyo", give a SNR of 36.17 dB and 31.28 dB for the SS and MSS algorithms respectively. SNR measurements, for the same video sequence, for ME algorithms have results: FS

37.62 dB, HS 33.43 dB, PHODS 30.58 dB and 3SLOG 33.71 dB. Comparing the SNR measurements of the different ME algorithms, we consider that, MSS algorithm has higher SNR than PHODS algorithm and similar to HS and 3SLOG algorithms. This also means that the video quality of MSS is acceptable, similar to other popular ME algorithms, and is well suited for low-bit rate video broadcasting (H.26L). Taking into consideration that researchers from the video processing domain indicate as acceptable the video quality of over 30 dB, we conclude that using the MSS ME algorithm we have in average a satisfactory video quality.

We used an ARM processor core emulator, called ARMulator [9] to take our measurements in terms of executed cycles and memory accesses. The simulation results for the number of cycles for the SS and MSS algorithms, show significant performance improvement (Fig. 5). As illustrated, the SS and MSS implementations have a big difference in execution times. It is obvious that the proposed algorithm requires less than half the execution time compared with the SS, for all transformations. Transformation No. 5 (pinpointed by the dashed line in Fig. 3) requires fewer cycles to implement: 69% of the original MSS cycles and merely 21% of the Original SS algorithm cycles. The average number of instructions required to implement the MSS is 31% of the average number of transformations required for the original algorithm (Fig. 6). The energy consumption of the Data and Instruction Memory is estimated by the Landman energy models [10]. Fig. 7, total energy consumption in both instruction memory and data memory hierarchy, illustrates the energy benefits of our algorithm. The best results in memory energy consumption are given by Transformation No 3.

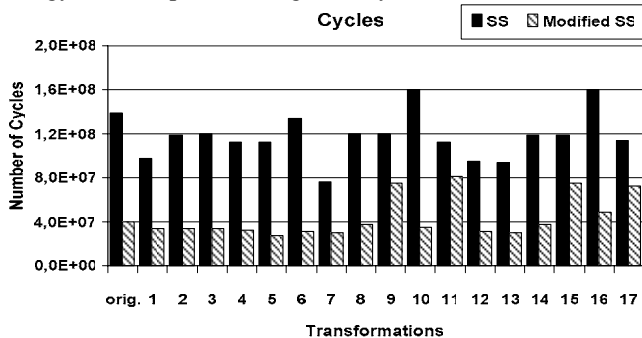


Fig. 5 MSS has lower execution time than the original SS

V. CONCLUSIONS

We have presented a promising MSS algorithm with low computational complexity. Additionally, we have proposed an embedded system with custom memory hierarchy, to implement this application. The MSS algorithm is an optimized form of SS, which increases algorithm performance by a factor of two, with minimal degradation in video quality. Applying the DTSE methodology, a number of different memory hierarchies have been produced. An exploration between the different memory hierarchies has been carried out, in order to find the optimum result. An embedded system architecture implementing the MSS and SS algorithms, based

on ARM920T processor, has been presented. Furthermore, the embedded system we have proposed for the application has high benefits in terms of performance and energy consumption.

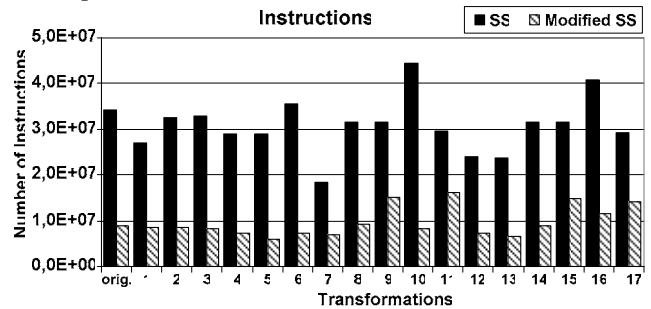


Fig. 6 MSS has lower number of executed instructions, showing a reduced complexity compared with original SS

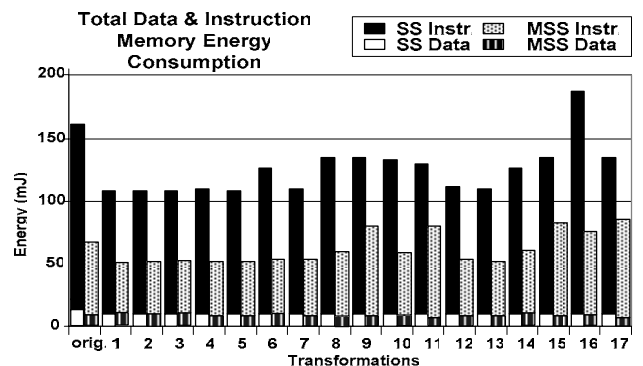


Fig. 7 MSS has lower data and instruction energy consumption, compared to the original SS

REFERENCES

- [1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra "Overview of the H.264 / AVC Video Coding Standard," IEEE Trans. on Circuits and Systems for Video Techn., July 2003, Vol. 13, Issue 7, pp. 560- 576.
- [2] C. Chok-Kwan, P. Lai-Man, "Normalized Partial Distortion Search Algorithm for Block Motion Estimation," IEEE Trans. on Circ. & Syst. for Video Tech., Vol. 10, No. 3, Apr 2000, pp.417-422.
- [3] X. Lu, T. Fernaine, Y. Wang, "Modeling Power Consumption of a H.263 Video Encoder," Proceedings of the International Symposium on Circuits and Systems, ISCAS '04, 23-26 May 2004, pp. 77- 80.
- [4] M. Dasygenis, et al, "Power and Performance Exploration of Embedded Systems Executing Multimedia Kernels," IEE Proc.-Comput. Digit. Tech., Issues "Low-power system-on-chip", Vol 149, No 4, July 2002, pp.164-172.
- [5] K. M. Nam, J.-S. Kim, R.-H. Park, Y. S. Shim, A fast hierarchical motion vector estimation algorithm using mean pyramid, IEEE Transactions on Circuits and Systems on Video Technology, Vol.5, No.4, Aug.1995, pp344-351.
- [6] V. Bhaskaran and K. Kostantinides, "Image and Video Compression Standards," Kluwer Academic Publishers, 1998.
- [7] F. Cathoor, K. Danckaert, K. Kulkarni, et al., "Data Access and Storage Management for Embedded Programmable Processors," Kluwer Academic Publishers, Boston, 2002.
- [8] I. Issenin, E. Brockmeyer, et al, "Data Reuse Analysis Technique for Software-Controlled Memory Hierarchies," in Proc. of DATE 2004, CNIT La Defese, Paris, France, 16-20 Feb. 2004, Vol. 1, pp. 202-207.
- [9] ARM L.T.D. ARM Software Development Toolkit, Version 2.50, Nov 1998.
- [10] P. Landman, 'Low Power Architectural Design Methodologies', PhD thesis, U.C. Berkeley, August 1994.