# A Modified Spiral Search Algorithm and its Embedded System Architecture Design

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 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 energy consumption reduction up to 64% and performance increase up to 77%, with a small penalty of 2.3 dB, in average, of video quality compared with the original spiral search algorithm.

***Keywords*—**Spiral Search, Motion Estimation, Embedded Systems, Low Power.

## I. INTRODUCTION

MULTIMEDIA applications, such as teleconferencing, video broadcasting and video streaming, have become increasingly common nowadays. The most critical factor is the increasing need for wireless systems or portable multimedia applications. The portability of a device is heavily bound by its energy consumption, since energy consumption affects the battery service life and weight, packaging costs, as well as the circuit reliability [1]. All video applications are dominated by data transfers and therefore, they feature high requirements on memory size and performance, while at the same time require great amounts of energy for the data transfers from and to off-chip memory. Energy dissipation must be lowered significantly, in order to implement these applications in embedded devices. This can be reduced using algorithmic modifications and transformations, together with a suitably designed custom memory hierarchy [2].

On the other hand, some of the most popular video conference applications are based H.26L [3] protocol. H.26L is a high-performance video coding standard for video transmission into a low bit rate channel. Also, involve computationally intensive parts of Motion Estimation (ME) kernels and require many off-chip memory accesses [4]. It has been reported that the ME algorithms are responsible for up to 60% of the total power consumption [5] of an application because they require huge number of off-chip memory accesses. For this reason, the performance and energy consumption optimization of multimedia algorithms has been the subject of extensive research [6], [7], [8], [9], [10].

Nikolaos Kroupis is PhD student in Department of Electrical and Computer Engineering (ECE), Democritus University of Thrace (DUTH), Xanthi, Greece (phone: +30 2541079959, e-mail: nkroup@ee.duth.gr).

Minas Dasygenis has received PhD from ECE of DUTH in 2005 (e-mail: mdasyg@ee.duth.gr).

Dimitrios Soudris, Assocciate Professor in Department ECE of DUTH (phone: +30 2541079557, fax: +30 25410 79545, e-mail: dsoudris@ee.duth.gr).

Antonios Thanailakis, Professor in DUTH (phone: +30 2541079541, fax: +30 25410 79545, e-mail: thanail@ee.duth.gr).

Researchers commonly tackle the problem of the ME power consumption either by proposing new ME algorithms [6], [7], specially designed for low power requirements, or use transformations optimization techniques on existing algorithms [8], [9], [11] in order to reduce the execution time and the energy consumption. Kuhn et al in a detailed survey of ME kernels [10], pinpointed the high complexity of the different ME kernels under consideration and studied it together with the visual quality of the video stream. Motion estimation 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) [12], the 3-Step Logarithmic Search (3SLOG) [13], the Parallel Hierarchical One-Dimensional Search (PHODS) [13] and the Spiral Search (SS) [6]. For the implementation of the above algorithms in an embedded system, a number of algorithmic optimization techniques have to be employed. The critical point, in which system optimization research has focused, is the reduction of the number of accesses to off-chip data memory. The conclusion is that ME applications have high computation requirements that cannot be met in current multimedia terminals.

Cheung et al [6] introduced a novel fast block-matching algorithm named “normalized partial distortion search”. This algorithm reduces computations by using a halfway-stop technique in the calculation of block distortion value. In order to increase the probability of early rejection of impossible candidate motion vectors, the algorithm normalizes the accumulated partial distortion and the current minimum distortion before comparison. Even if they proposed interesting optimizations in Full Search (FS) algorithm, the computational complexity is still very high compared to the Spiral Search algorithm.

A fast ME search method, based on Spiral Search algorithm was introduced by Zahariadis et al [14]. They proposed a three step algorithm, which follows a spiral path searching outwards for candidate locations that satisfy the matching criterion. The efficiency of this algorithm arose from two facts: i) the reduction in the number of candidate locations achieved by leaving out selected zones of pixels and ii) the reduction in the number of computations since they skip some candidate locations. It is a fast motion estimation algorithm, but reduces the signal-to-noise ratio up to 3dB and uses the mean square error criterion. Thus, it has more computational complexity than Sum of Absolute Difference (SAD) but it gives a better result in output video quality. However, there is no hardware implementation taking into account performance and energy design issues.

Furthermore, in [7] researchers proposed power consumption models for the H.263 video coding, using different ME kernels. The models were developed by executing the H.263 encoder on a Pentium III PC. However, they have not designed any architecture nor optimized the system for the given applications.

In previous research works [8], [9] we had introduced data memory hierarchy optimization techniques for motion estimation algorithms, using global loop and data-reuse transformations for three different embedded processor architecture models. Even though we had managed to reduce energy consumption up to 10-60%, the algorithms had high computational complexity to execute on portable devices. In this research work we confront the computational complexity and energy consumption together. We propose a novel version of Spiral Search (SS), in which we make algorithmic optimizations to it, and design a low power memory hierarchy (achieving a 77% reduction in complexity and 64% reduction in energy consumption).

The objective of the proposed work is the study, the optimization and the implementation of SS algorithm. The original SS has a criterion which interrupts the search if it exceeds a certain threshold. Even though SS is a less intensive computationally algorithm than FS, it still has a high computational complexity, making it difficult to implement it in embedded devices. We address this problem by presenting a new modified form of the SS algorithm, which improves both execution time and energy consumption. The modified algorithm performs the search in a fashion similar to the SS algorithm, but uses a step search larger than one pixel comparing to the original SS. This results in omitting the comparisons of roughly half the number of the blocks, thus decreasing the computational complexity of the modified algorithm.

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 proposed memory hierarchy is the outcome of the exploration of all possible structures of hierarchy for the algorithm, and determining the optimal one in terms of execution time and energy consumption. The algorithmic modifications involve, however, a reduction in the video quality, which remains in the level of tolerance, as proven by experimental results (Peak Signal to Noise Ratio, PSNR measurements). Finally, the optimized version of the proposed Modified SS algorithm quadruples the performance and reduces the energy consumption by 64%, compared with the original SS algorithm.

## II. Spiral Search Motion Estimation Algorithm and Modified Algorithm

In a sequence of frames, the current frame is computed from a previous frame known as reference frame. The current frame is divided into blocks, typically 16×16 pixels in size. Each block is compared to a block in the reference frame and the best matching block is selected. The search is conducted over a predetermined search area. A vector denoting the displacement of the block in the reference frame with respect to the block in the current frame is computed. This vector is known as "motion vector". Motion estimation algorithms compute the motion vectors by minimizing a cost function, referred to as distance criterion. The most widely used distance criterion is the Sum of Absolute Differences (SAD) [1]. The SAD criterion of the block located at $(m,n)$ position of a frame is calculated using Equation 1, with $I_f$ and $I_{f-1}$ being the pixel's luminance of the current and previous frames, respectively. The motion vector $(MV_x, MV_y)$ is given for the block located at $(x,y)$, which yields the minimum SAD criterion value.

$$SAD(dx,dy) = \sum_{m=x}^{x+B-1} \sum_{n=y}^{y+B-1} \left| I_f(m,n) - I_{f-1}(m+dx, n+dy) \right| \tag{1}$$

$$(MV_x, MV_y) = \min_{(dx,dy)} SAD(dx,dy)$$

### A. Spiral Search Motion Estimation 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 searching process on the search space moves outward spirally (Fig. 1), which means that it moves in circles with increasing radius. Fig. 2 shows the pseudo-code of SS algorithm. The first two external loop indices divide the frames into $B×B$ size blocks. The $l$-indexed loop refers to the spiral which we are located, while the $k$-indexed loop describes the location of the block within the spiral. The distance criterion SAD is computed from all pixels of the $B×B$ 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 $m$-indexed loop execution (stop criterion), because a better match cannot be found in this block.

Even though the SS algorithm compares all the blocks in the search space area like the FS algorithm, SS differs in the stop criterion. The computational complexity of the SS algorithm ($CC_{SS}$) is calculated by Equation 2, where $N_{comp}$ and $CC_{block}$ are the number of the block comparisons and the computational complexity yield by a single block comparison, respectively.

$$CC_{SS} = N_{comp} \cdot CC_{block} \tag{2}$$

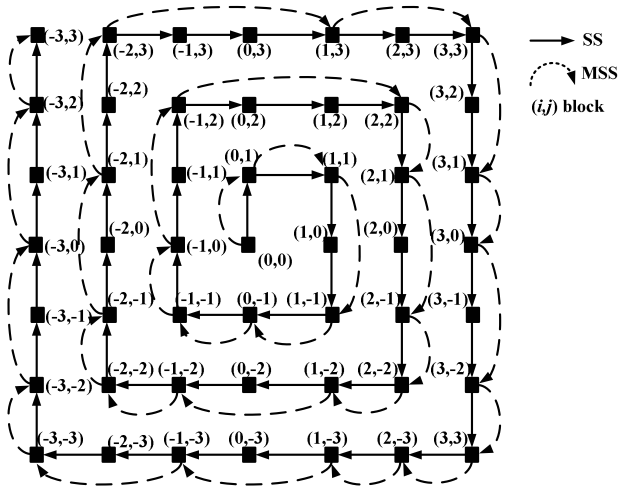

Fig. 1 MSS uses less comparison blocks compared with SS, using steps with distance one, two or three pixels

```
/* 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

### *B. Modified Spiral Search Motion Estimation Algorithm*

During the study of the motion estimation in the SS algorithm, we observed an interesting characteristic. The SAD criterion value decreases approaching to the block with the best match to 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 block, the SAD value becomes higher (>450). On the other hand, there is no large difference of SAD values between two blocks located one pixel away from each other. For example, the maximal difference is 1×16×256=4096 (1 pixel step multiplied by the 16 pixels of the one dimension and 256 of the highest possible luminance difference) in the worst case. In practice, we measured the difference to vary between 100 and 500.

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 (higher than a threshold $V_{TH1}$), then also the SAD value of ($x$+1, $y$) will be high and could not be the best matching block. 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 (higher than a threshold $V_{TH2}$) we skip the next two blocks and we compare the ($x$+3, $y$) block. Such, pixel steps are defined according to the current value of the SAD criterion. Summarizing, we compute the pixel step by using two thresholds $V_{TH1}$ and $V_{TH2}$ ($V_{TH1}$<$V_{TH2}$), as follows:

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

The pseudo-code of the MSS algorithm is similar to the SS algorithm and has additionally the new check conditions, for the two thresholds values, after the calculation of SAD criterion (Fig. 2). 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. The performance and the energy consumption (which are both related with the complexity of the algorithm) are the most important, especially in mobile multimedia terminals. The proposed MSS allows to the designer to have a pool of solutions for his/her Motion Estimation (ME) application.

Equation 3 calculates the computational complexity of the MSS algorithm. More specifically, the sum gives the total number of the 2-pixels steps, and the gives the total number of the 3-pixels steps. An overhead ($Overhead_{step}$) inside the block comparison has been added, in order to scale the step (1, 2 or 3 pixels) according to the SAD criterion value.

$$CC_{MSS} = \left(N_{comp} - \sum steps_{(2)} - 2\cdot\sum steps_{(3)}\right)\cdot\left(CC_{block} + Overhead_{step}\right) \quad (3)$$

Finally, the block comparison reduction using both the 2-pixels step and the 3-pixels step gives smaller computational complexity compared with the SS algorithm.

## III. Optimization Transformations

In multimedia applications the energy consumption related to memory transfers is the dominant factor in total energy cost [11]. This motivated us to find an efficient method to reduce it. For this purpose, we created an application specific data memory hierarchy which 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 methodologies [8], [9] and [11], for typical input video sequences benchmark QCIF (176×144). 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, SS algorithm has two data arrays (*current_frame* and *previous_frame*), which account for most of the data accesses. Using profiling, we found that the two array signals of 176×144 pixel size have the highest number of data accesses [15]. 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 [16], [17] 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).

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.

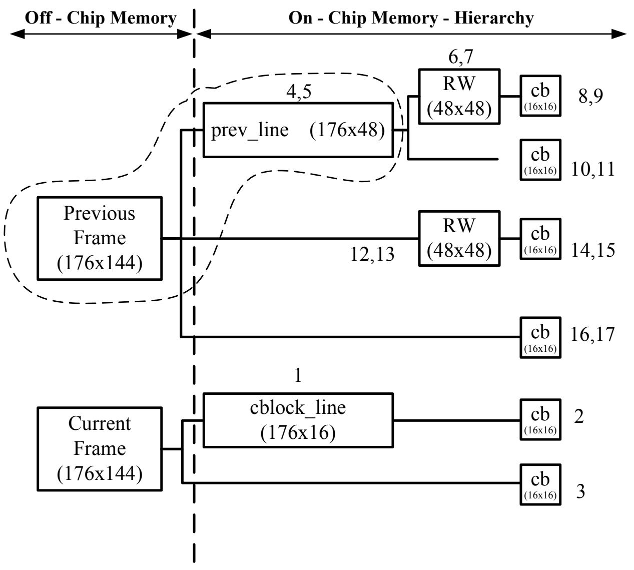

Fig. 3 An exploration is required to pinpoint the optimum memory hierarchy. The numbers indicate different memory hierarchies

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 memory. The excessively large sizes of *current_frame* and *previous_frame* did not allow these to be mapped there. 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.

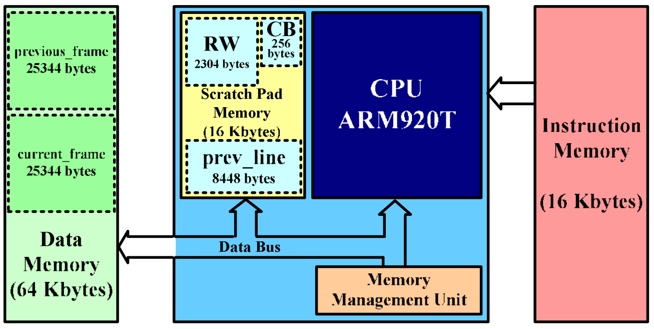
previous_frame 25344 bytes
current_frame 25344 bytes
Data Memory (64 Kbytes)
RW 2304 bytes
CB 256 bytes
Scratch Pad Memory (16 Kbytes)
prev_line 8448 bytes
CPU ARM920T
Data Bus
Memory Management Unit
Instruction Memory (16 Kbytes)

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

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.

The system consists of an external data memory, instruction memory and the processor unit. Applying the Data transfer and Storage Exploration (DTSE) Methodology, the number of memory accesses to the background memory *current_frame* and *previous_frame*, is reduced. *prev_line*, *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

To evaluate our MSS algorithm, we took measurements in video quality and made estimations in terms of cycles and energy consumption. The measurements reveal that the output video quality of the MSS is similar to the SS algorithm. Comparison measurements between SS and MSS algorithms were also performed, in order to estimate the energy savings and the performance increase in an ARM920T core implementation. Finally, the new implementation of MSS algorithm is suited to low power devices, because it achieves higher energy savings compared to the original SS algorithm.

### *A. SS and MSS Evaluation*

We used an ARM processor core emulator, called ARMulator [18] to take our measurements in terms of executed cycles and memory accesses. This tool, which features algorithm and basic system architecture simulation, estimates the number of cycles, the number of instructions, and the number of accesses to data memory. 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 23% of the original SS algorithm cycles.

The huge reduction of the number of execution cycles (which improves the performance) of the MSS algorithm enables the design of real-time hardware implementations. The average number of (micro-)instructions required to implement the MSS is 31% of the average number of transformations required for the original algorithm (Fig. 6). The number of executed commands is directly related to the energy consumption of the instruction memory. With other words reducing the number of executed instructions we are reducing the instruction memory energy consumption.

The energy consumption of the Data and Instruction Memory is estimated using the CACTI energy model [19]. 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. The data memory has very low penalty in terms of energy consumption. The proposed architecture, using the on-chip memory minimizes the off-chip memory transfers, which are more power consuming compared to the on-chip memory transfers. The total memory energy consumption in the MSS algorithm is reduced by 50% compared to the original SS algorithm.

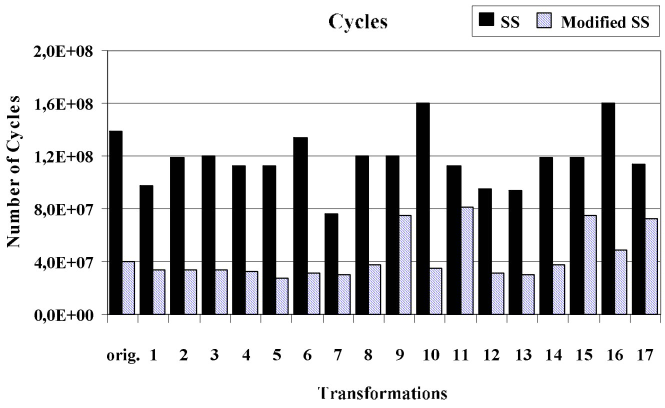

Fig. 5 Our MSS has lower execution time than the original SS

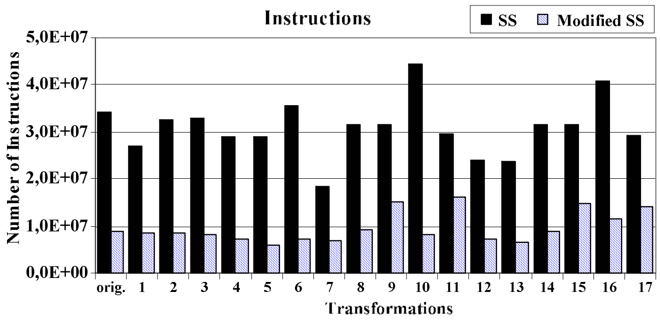

Fig. 6 Our MSS has lower number of executed instructions, showing a reduced complexity compared with original SS

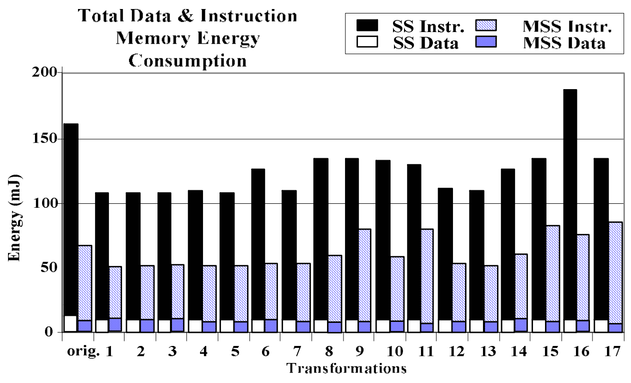

Fig. 7 Our MSS has lower data and instruction energy consumption, compared to the original SS

## B. *Video Quality Measurements*

The MSS algorithm has fewer block-matching comparisons, resulting in lower computational complexity. This does not yield always the best matching and does not always result in optimal motion vectors. For this reason, the reconstructed sequence may have lower video quality compared with the sequence using the SS algorithm. Fig. 8 presents the PSNR of different video sequences (benchmark) using different threshold values; the first point of *X*-axis corresponds to the original SS algorithm, while all the remaining values correspond to different $V_{TH1}$ values. The values of $V_{TH2}$ are related with $V_{TH1}$ according to the relation $1.5\times V_{TH1}$. Measurements using the 200 frames video sequences of "Akiyo", "Coastguard", "Container", "Foreman", "M_d", and "Silent" yield a PSNR of MSS similar with SS. For example, for "Akiyo" sequence the SS algorithm gives PSNR 37.55 dB, while the MSS gives PSNR 34.75 - 36.24 dB for variable threshold values. Fig. 8 indicates that MSS algorithm gives very good results in video quality for the remaining benchmarks, compared with the original SS. As the thresholds values increase the numbers of block comparisons increase, which means that the computational complexity increases. Thus, for higher threshold values the PSNR measurements of MSS are very close with the SS algorithm.

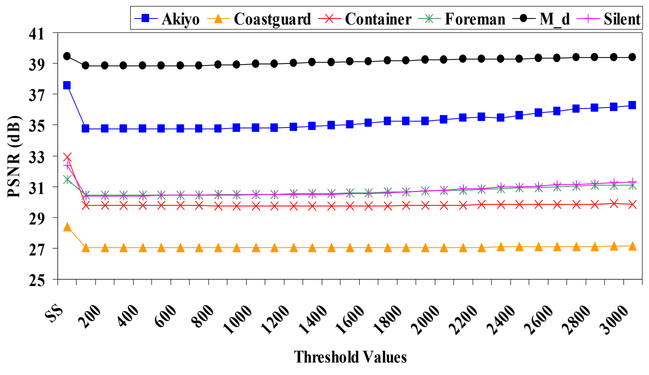

Fig. 8 The quality of different video sequences using MSS algorithm (measured in PSNR), is similar with SS algorithm.

In order to validate our estimations we examined a representative set of video sequences (Table I). Table I presents the PSNR measurements, for different established video sequence benchmarks. Comparing the PSNR measurements of the different ME algorithms, we conclude that MSS algorithm has similar PSNR with PHODS and 3SLOG algorithms. The MSS PSNR results are 2.3 dB lower, 2.6 dB lower, 2.3 dB and 1 dB lower, than the original SS, HS, 3SLOG and PHODS respectively. Taking into consideration that researchers from the video processing domain indicate as acceptable the video quality of over 30 dB [20], we conclude that using the Modified Spiral Search ME algorithm we have a satisfactory video quality.

TABLE I
PSNR MEASUREMENTS OF MSS SHOW THAT THE VIDEO QUALITY OF MSS IS SIMILAR WITH OTHER POPULAR MOTION ESTIMATION ALGORITHMS

| PSNR (dB) | Akiyo | Coastguard | Container | Foreman | M_d | Silent |
|---|---|---|---|---|---|---|
| MSS | 31.28 | 23.66 | 30.50 | 30.22 | 39.07 | 29.70 |
| SS & FS | 36.17 | 27.68 | 31.74 | 31.77 | 39.63 | 31.55 |
| HS | 34.81 | 28.50 | 33.17 | 32.02 | 39.94 | 31.87 |
| 3SLOG | 34.29 | 28.51 | 32.06 | 32.00 | 39.84 | 31.70 |
| PHODS | 31.07 | 27.93 | 31.30 | 30.52 | 39.55 | 30.27 |

In order to evaluate the modified algorithm, we computed the PSNR of the output video, together with the number of cycles (Fig. 9) versus the threshold values. Fig. 9 shows that the execution time (i.e. number of cycles) of the proposed algorithm, implemented on an ARM processor core, is about the half of the original algorithm, while, on the same time, the PSNR level is similar. The quality can be further improved using a higher threshold, incurring a small penalty in the number of processing cycles.

Concluding, the optimized MSS algorithm gives an overall optimal result in terms of performance (up to 77% increase) and energy consumption (up to 64% reduction), with some penalty in image quality (about 2 dB video degradation in the PSNR). Furthermore, MSS algorithm provides trade-offs to the designer in terms of performance, energy consumption and video quality.

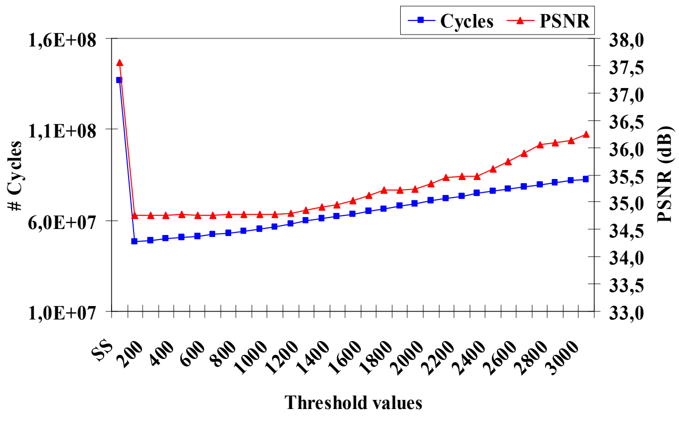

Fig. 9 Execution time vs. video quality comparison between SS and MSS with variable threshold values

## V. CONCLUSION

ME algorithms are crucial in the multimedia applications domain. ME algorithms (e.g. spiral search) are the dominant part in the system computational complexity and energy consumption of video applications. We have presented a promising Modified Spiral Search algorithm with lower computational complexity. The MSS algorithm reduces the number of block comparisons using two threshold values and steps bigger than one, depending on the previous criterion value of the block matching. The Modified Spiral Search algorithm is an optimized form of SS, which increases algorithm performance by a factor of four, with minimal degradation in video quality.

An exploration between the different memory hierarchies has been carried out, in order to find the optimal one. An embedded system architecture implementing the MSS and SS algorithms, based on ARM920T processor, has been presented.

## REFERENCES

[1] D. Soudris, C. Piguet, C. Goutis, *Designing CMOS Circuits for Low-Power*, Kluwer Academic Publisher Press, (2002)

[2] Christian Piguet, *Low-Power Electronics Design*, CRC Press, (2004)

[3] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the H.264 / AVCVideo Coding Standard", *in IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 13, Issue 7, pp. 560- 576 (2003)

[4] André Kaup and Hubert Mooshofer, "Performance and Complexity Analysis of Rate Constrained Motion Estimation in MPEG-4", *Proc. Multimedia Systems and Applications II*, Boston, Mass., (1999) 20-22 September, SPIE Vol. 3845, pp. 202-211

[5] Karl Guttag, Robert J. Gove, and Jerry R. Van Aken, "A single-chip multiprocessor for multimedia: The MVP", *IEEE Computer Graphics and Applications*, Vol. 12, No. 6, pp. 53–64 (1992)

[6] Chok-Kwan Cheung and Lai-Man Po, "Normalized Partial Distortion Search Algorithm for Block Motion Estimation", *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 10, No. 3, pp. 417-422 (2000)

[7] Xiaoan Lu, Thierry Fernaine, Yao Wang, "Modeling Power Consumption of a H.263 Video Encoder", *Proceedings of the International Symposium on Circuits and Systems, ISCAS '04*, (2004) 23-26 May, pp. 77- 80

[8] D. Soudris, N. Zervas, A. Argyriou, M. Dasygenis, K. Tatas, C. Goutis and A. Thanailakis, "Data-reuse and parallel embedded architectures for lowpower, real-time multimedia applications", *Proceedings of 10th Int. Workshop PATMOS*, Gottigen, Germany, (2000) September, pp. 243–254

[9] M. Dasygenis, N. Kroupis, K. Tatas, A. Argyriou, D. Soudris and A. Thanailakis, 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, pp.164-172, (2002)

[10] P. Kuhn, G. Diebel, S. Herrmann, A. Kaup, A. Keil, R. Mayer, H. Mooshofer, W. Stechele, "Complexity and PSNR-Comparison of several Fast Motion Estimation Algorithms for MPEG-4", vol. *SPIE 3460 Applications of Digital Image Processing XXI*, San Diego, July, (1998)

[11] F. Catthoor, K. Danckaert, K. Kulkarni, E. Brockmeyer, P.G Kjeldsberg, T. van Achteren and T. Omnes, *Data Access and Storage Management for Embedded Programmable Processors*, Kluwer Academic Publishers, Boston, (2002)

[12] Kwon Moon Nam, Joon-Seek Kim, Rae-Hong Park, Young Serk Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid", *IEEE Transactions on Circuits and Systems on Video Technology*, Vol.5, No.4, pp 344-351, (1995)

[13] Bhaskaran and K. Kostantinides, *Image and Video Compression Standards*, Kluwer Academic Publishers, (1998)

[14] Th. Zahariadis, D. Kalivas, "A Spiral Search Algorithm For Fast Estimation Of Block Motion Vectors", *Proceedings of the EUSIPCO 96, Eighth European Signal Processing Conference*, Trieste, Italy, (1996), September, pp 1079-82.

[15] Francky Catthoor, Frank Franssen, Sven Wuytack, Lode Nachtergaele, and Hugo De Man, "Global communication and memory optimizing transformations for low power signal processing systems", *Proceedings IEEE/ACM Int. Workshop on Low Power Design*, Napa Valley, CA, Apr. (1994), pp. 203-208

[16] Ilya Issenin, Erik Brockmeyer, Miguel Miranda and Nikil Dutt, "Data Reuse Analysis Technique for Software-Controlled Memory Hierarchies", *Proc. of Design, Automation and Test in Europe, DATE 2004*, CNIT La Defese, Paris, France, (2004), 16-20 February, Vol. 1, pp. 202-207

[17] N. D. Zervas, K. Masselos, C.E. Goutis, "Data-reuse exploration for low-power realization of multimedia applications on embedded cores", *Proceedings of 9th International Workshop on Power and Timing Modeling, Optimization and Simulation* (PATMOS'99), (1999), October, pp. 71-80

[18] ARM Software Development Toolkit, ARM L.T.D., Version 2.50, Nov (1998)

[19] P. Shivakumar and N. Jouppi, *CACTI 3.0: An Integrated Cache Timing*, Power, and Area Model, WRL Research Report 2001/2, Aug. (2001).

[20] Prashant J. Shenoy, Harrick M. Vin, "Efficient support for scan operations in video servers", *International Multimedia Conference*, San Francisco, California, United States, (1995), pp. 131-140