New Motion Estimation Algorithms and its VLSI Architectures for Real Time High Definition Video Coding

Gustavo Sanchez[1], Marcelo Porto[1], Diego Noble[1], Sergio Bampi[2], Luciano Agostini[1]

[1]Federal University of Pelotas, Group of Architecture and Integrated Circuits, Pelotas, Brazil
[2]Federal University of Rio Grande do Sul, Informatic Institute, Porto Alegre, Brazil
e-mail: {gfsanchez, porto, dvnoble, agostini}@inf.ufpel.edu.br; bampi@inf.ufrgs.br

ABSTRACT

This paper presents an efficient hardware design using the new Motion Estimation (ME) algorithms named: Multi-point Diamond Search (MPDS) and Dynamic Multi-Point Diamond Search (DMPDS). These algorithms are more efficient to avoid from local minima falls than traditional fast algorithms. This fact contributes to increase the quality of the motion vectors, especially in High Definition (HD) videos, were the number of local minima are considerable higher. Two versions of MPDS algorithm were proposed. The first one, focused on high performance, is capable to process videos QFHD at 30 frames per second when synthesized to Altera Stratix 4 and 90nm TSCM, with only 18mW. The second version is focused on quality enhancement and is capable to process HD 1080p videos in real time. The DMPDS architecture has been developed focusing on high performance and was synthesized to Altera stratix 4. This architecture is capable to process videos QFHD at 34 frames per second. In comparison to related works, our solutions obtained the highest processing rates, and a good trade-off among power consumption, area, memory bits and performance.

Index Terms: Motion Estimation, High Definition, VLSI Architecture.

1. INTRODUCTION

Nowadays, a vast number of applications use digital videos. Usually, these applications need to focus on performance for real time at 30 frames per second, which is necessary to show sense of continuity. With this increase in the number of applications, the video quality is also been increasing, which makes high definition videos more common. High definition videos need a lot of storage space and also a high bandwidth to be transmitted. These facts contribute to increase the effort in video coding research, and the development of new video coding standards. Current video coders, as H.264/AVC [1], for example, divide the coding process in many steps. The main steps of a current video coder are: intra frame prediction, inter frame prediction, transforms and entropy encoding. The inter frame prediction is explored by the Motion Estimation (ME) and is the focus of this work.

ME represents 80% of the total computational complexity of current video coders [2]. The ME must find the best matching in the reference frames for each block of the current frame, defining a motion vector indicating where the best matching was found. A search algorithm defines how the search is done and a similarity criterion is used to compare the candidate blocks. The search for best vectors is known to be very expensive in terms of calculations and, consequentially, in terms of processing time. The Full Search (FS) [3] algorithm must explore all possibilities in a given search area, which implies in a very high computational cost, especially for high resolution videos, which requires the use of larger search areas. Based on this fact, it is important to explore new solutions, which bring a good tradeoff between objective quality, measured in Peak-to-Signal Noise Ratio (PSNR), and computational cost.

There are many fast search algorithms in scientific literature. These algorithms deal with this complexity at different levels of impact in objective quality (PSNR). Exploring characteristic of locality among temporal correlated blocks allows these algorithms to achieve good results in terms of numbers of calculations and also allows achieving high video quality when low resolution videos are considered. However, these algorithms assume that the error function decreases monotonically on the surface of the frame, in order to speed up the algorithm. This assumption does not hold true sometimes, and the search might be trapped into a local minima, especially in high definition videos.

The majority of the published ME search algorithms only considers low resolution videos, as QCIF and CIF, in its experiments. However, the quality results of the ME algorithms can significantly change with the increase of the video resolution. For low resolution videos, the quality results for FS and many other fast algorithms are very close. The great amount of pixels in high definition videos may lead the fast algorithms to choose, more frequently, local minima as the best matching. Thus, the quality losses (in comparison with FS) are significant in this scenario. Techniques to avoid local minima must be explored to enhance the video quality, without a significant increasing in the ME computational complexity.

The FS algorithm is capable to avoid from local minima by using brute force testing all available possibilities. Another strategy to avoid from local minima is using multi-point algorithm, where the proposed algorithm should start search in multiple initial points.

This paper presents two new motion estimation search algorithm named Multi-Point Diamond Search (MPDS) and the Dynamic-Multi-Point Diamond Search (DMPDS). Hardware architectures have been developed implementing these algorithms, targeting low power and real time processing of high definition videos, as HD 1080p and QFHD (3840x2160). MPDS and DMPDS algorithms are more efficient to avoid local minima falls than traditional fast algorithms, especially in high definition videos. Software evaluation shows that MPDS and DMPDS algorithms can significantly improve the quality for high definition videos, in comparison with Diamond Search (DS) algorithm [4].

Two versions of the MPDS hardware architecture were developed, described in VHDL and synthesized. The synthesis results are presented for Altera Stratix 4 FPGA and TSMC 90nm standard cells technology. Both architectures are able to reach real time (30 fps) when processing HD 1080p sequences, with low power consumption. One of the MPDS architectures is also able to reach real time (30 fps) processing for QFHD videos. This architecture can process 30 frames per second with only 4.5mW and 9mW, when synthesized to ASIC with 90nm standard cells technology, for HD 1080p and QFHD videos, respectively. The DMPDS hardware architecture was also described in VHDL and synthesized to an Altera Stratix 4 FPGA. The architecture is able to reach real time when processing QFHD videos consuming low hardware resources.

The paper is organized as follows: Section 2 presents a study about the ME in high definition videos. Section 3 explains the MPDS and the DMDPS algorithm and presents some software evaluations of these new algorithms on high definition videos. Section 4 shows details about the architectures design and section 5 presents the synthesis results and comparisons with related works. Finally, section 6 presents the conclusions.

2. MOTION ESTIMATION IN HIGH DEFINITION VIDEOS

Hardware implementations for motion estimation are widely used to speed up the generation of the motion vectors. The use of hardware solutions is still more important when real time processing is considered, mainly for high definition video coding. There is a large amount of published works with hardware architectures for the motion estimation process, including both FPGA and ASIC solutions. Most of the solutions are based on the traditional Full Search (FS) algorithm. There is a dramatic growing in the use of high definition videos. Even mobile devices, as cell phones and digital cameras, can support videos with high definition. The growing in digital videos definition, in fact, is obtained with the growing in the number of pixels inside a frame, increasing the frame resolution. A HD 1080p frame has about 82 times more pixels than a QCIF frame, however, both must represent the same scene.

The increase in the video resolution can directly affect the ME results. The fast ME algorithms can be affected by this characteristic, generating different results, for the same video, in different resolutions. High resolution videos tend to present very similar neighboring pixels (much more than low resolution ones) and this fact contributes to increase the occurrence of local minima falls.

For this evaluation, DS and FS algorithms were applied to ten HD 1080p video sequences to demonstrate the influence of the video resolution growing in the ME process. The used video sequences were: blue_sky, man_in_car, pedestrian_area, hush_hour, station2, sunflower, riverbed, rolling_tomatoes, traffic and tractor [5]. These sequences were resized for many lower resolutions: 256x144 pixels (144p – which is equivalent to QCIF resolution in a 16:9 aspect ratio); UMD 272p (480x272); EDTV 480p (854x480); HD 720p (1280x720) and HD 1080p (1920x1080). All the experiments in this paper will use these video sequences.

Figure 1 shows the average PSNR gain curves (from the ten video sequences) of DS and FS algorithms, considering the five cited resolutions. The used block size was 16x16 pixels, and the search area grew proportionally to the video resolution growing.

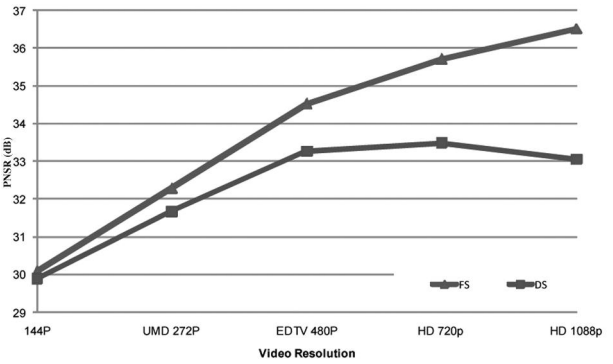

Figure 1. Average PSNR curves of DS and FS algorithms considering five resolutions.

Through analysis of Figure 1 curves it is possible to notice that the difference in PSNR results between DS and FS algorithms grows significantly with the video resolution increasing. The FS ever increases the PSNR gains with the video resolution growing. This happens because the search area also grows and the FS algorithm can explore all candidate blocks in the search area. For the DS algorithm, the growing in the search area does not provide a significant increase in PNSR because the DS gets trapped in local minima. In fact, the efficiency of the DS algorithm is reduced for high resolution videos. The great similarity of neighbor pixels (and blocks) leads the DS algorithm to more frequently fall in local minima. Then the PSNR losses of DS in relation to FS increase with the resolution increase.

These results shows that the DS algorithm is efficient for low resolution videos, since it has PSNR results which are very similar to FS results, but it presents a significant computational cost reduction. For high definition videos, the PSNR losses become significant, and the relevance of the algorithm is only related with computational cost reduction. This experiment demonstrates that quality results for fast algorithms in low resolution videos can not be extrapolated for high definition video scenario.

A. Local Minima Analysis

The efficiency loss of DS algorithm in high definition videos is mainly caused by the increase of local minima falls. With the search area growing, the ME optimum candidate block can be far away from the center of the search area. Fast algorithms, like DS, can be easily trapped in local minima, before achieving the optimal result.

Sum of Absolute Differences (SAD) maps are presented in Figure 2, to illustrate the growing of the local minima problem in high definition videos. Figure 2 presents SAD maps for a search area in the sun_flower video. Each map represents the same region of the frame, with different number of pixels. Figure 2 (a), (b), (c), (d) and (e) represents the SAD maps for the resolutions 144p, UMD 272p, EDTV 480p, HD 720p and HD 1080p, respectively. The HD 720p map is very similar to the HD 1080p map because they are high resolution images. The images represents the SAD magnitude for 16x16 blocks, where dark blue represents lower SAD values, and light orange represents higher SAD values.

In Figure 2 (a) is possible to see that good SAD results can be achieved around the center of the search area. Figure 2 shows that, for higher resolution video, new regions with good SAD results can be found, beyond the central region. This is an evidence of the increasing of local minima with the increasing of video resolution. More dark regions can be seen in Figure 2 (c), however, in Figure 2 (a), (b) and (c) the global minima can be visually identified. The SAD map for HD 720p and HD 1080p resolution, presented in Figure 2(d) and Figure 2(e), show many dark blue regions, and it is impossible to visually identify the global minima.

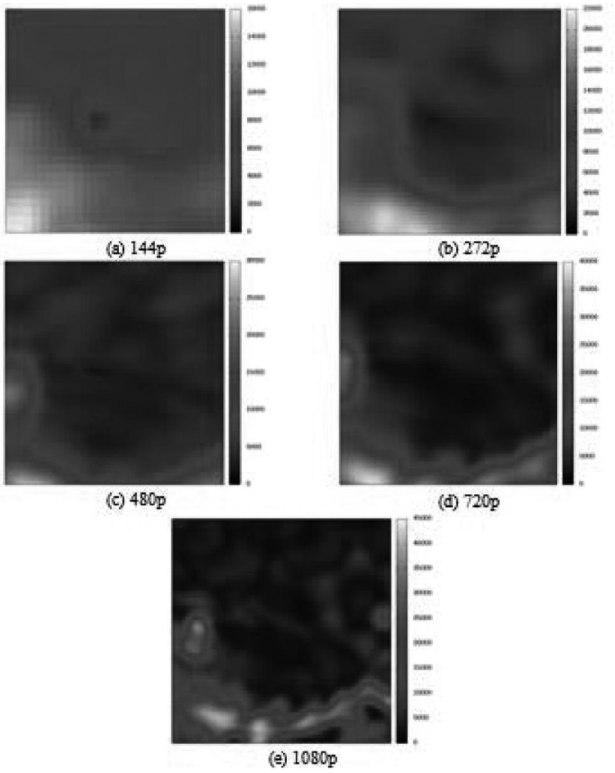

Figure 2. SAD maps for sun_flower with different resolutions.

The analysis of the images presented in Figure 2 can explain the results presented in Figure 1. Both DS and FS algorithms choose similar candidate blocks (or even the same) at low resolution videos. As presented in Figure 2 (a), there are only a few candidate blocks with good SAD results in a low resolution video, and all around the center of the search area. Even if the DS do not reach the optimal candidate block, it will choose a closer block with similar SAD value (average difference of 0.19dB). For high resolution videos, like HD 1080p, the higher number of local minima significantly increases the differences between PSNR results for DS and FS algorithms (more than 3.4dB).

For a better analysis about local minima, Figure 3 presents the Sum of Absolute Differences (SAD) for every candidate block in a search area of 128x128 pixels in the HD 1080p sun_flower sequence. This picture represents a 3D view of the SAD magnitude for blocks of 16x16 pixels, where valleys represent lower SAD values, and peaks represent higher SAD values.

In Figure 3 is also possible to see that there are many peaks and valleys in this search area. The global minimum is the valley with lower SAD. This scenario is different for low resolution videos, where the number of peaks and valleys are much lower than presented in Figure 3. Fast iterative algorithms (DS for example) are unable to transpose some peaks around the center to achieve a global minimum beyond. Fast algorithms are easily trapped in local minima around the center of the search area.

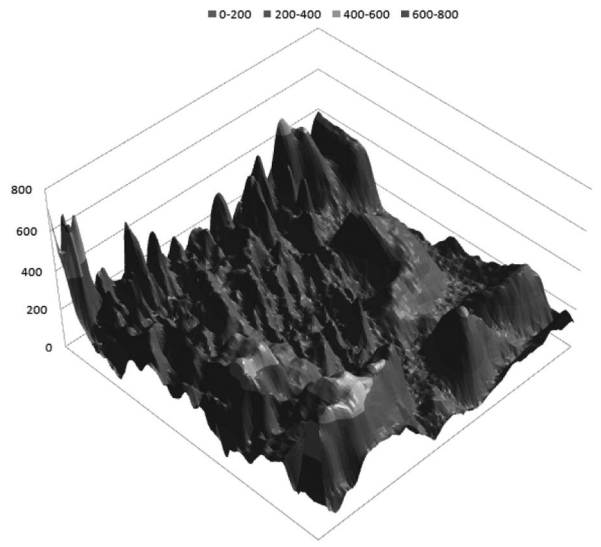

Figure 3. 3D SAD map for sun_flower sequence in HD 1080p.

Due to the FS algorithms complexity, its implementation for high definition videos is very computationally expensive. The performance requirements to achieve real time processing in this kind of video are extremely high, and hardware solutions for FS algorithms must explore massive parallelism, increasing the hardware resources utilization and also power consumption.

The development of new fast ME algorithms, focused in high definition videos, are very important to achieve good trade-off between quality and computational cost. The hardware implementation is also very important, mainly for real time applications on portable devices. Fast algorithms must be easily implemented in hardware, translating its computational cost reduction to hardware resources reduction, and also power saving.

3. MPDS AND DMPDS ALGORITHMS

The MPDS and DMPDS algorithms use a multipoint strategy that focus on avoiding from local minima falls. These algorithms start the search in multiples points in the search area, providing the algorithms the possibility to transpose a local minimum near to the center of the search area. This strategy helps the algorithm to find a result near the global minimum.

The MPDS search algorithm uses the search engine of DS algorithm. However, the search is not made only at the center of the search area. The MPDS algorithm finds the best matching in five different positions of a search area. Each position, except the central one, is defined inside of a sector (A, B, C and D), as presented in Figure 4. The MPDS is not restricted to only one start point, exactly to avoid the same local minimum, which the DS would reach. Then the MPDS defines five different start points and five independent DS cores are triggered in the same search area. In the worst case, the MPDS algorithm will reach the same result than the DS algorithm.

The main idea in MPDS algorithm is to not be restricted to local minima around the center, but it is important to keep the video motion characteristics. The central DS is responsible to achieve a high quality when low motion videos are being encoded, because the best block match should be near the center. Inserting 4 DS in a distant area around the center is the MPDS approach to deal with high motion video sequences by evading from the central local minima, when the normal DS would reach low quality results.

The pseudo code below describes the MPDS algorithm:

```
1.  Define d
2.  Frame <= 0
3.  Block <= 0
4.  Repeat
5.    Repeat
6.      SAD_zero <= Execute_DS(0, 0)
7.      SAD_A <= Execute_DS(d, d)
8.      SAD_B <= Execute_DS(-d, d)
9.      SAD_C <= Execute_DS(-d, -d)
10.     SAD_D <= Execute_DS(d, -d)
11.     Lowest_SAD <= Min(SAD_zero, SAD_A,
        SAD_B, SAD_C, SAD_D)
12.     Generate_Vector (Lowest_SAD)
13.     Block ++
14.   While (Block < Max_Blocks)
15.   Frame ++
16.   While (Frame < Max_Frames)
```

Figure 4 describes the search positions of MPDS algorithm. Each initial search point is defined by its coordinates inside the sector. The point (0,0) is the central position and it will obtain the same vector than the DS algorithm. The search in the sectors A, B, C and D will be done according the distance parameter d. The d parameter is the distance (number of samples in X and Y axis) from the central point (0,0). The sector A, B, C and D starts searching respectively at positions (*d*,*d*), (-*d*,*d*), (-*d*,-*d*) and (*d*,-*d*). When the search ends the MPDS algorithm selects the best result from the five applied diamonds.

The MPDS results are directly influenced by the value of d parameter. For low motion activity videos, the MPDS algorithm can achieve good results with low d values, since good candidate blocks can be found near to the center of the search area. On the other hand, high motion activity videos can be benefited by higher d values. Figure 5 shows the PSNR curves for MPDS algorithm with each one of the ten used test video sequences cited before. The curves were generated considering the variation of d parameter value, from zero (same result than DS algorithm) to 40, considering 8x8 block sizes.

Through the analysis of the Figure 5 it is possible to notice how the MPDS algorithm responds to the *d*

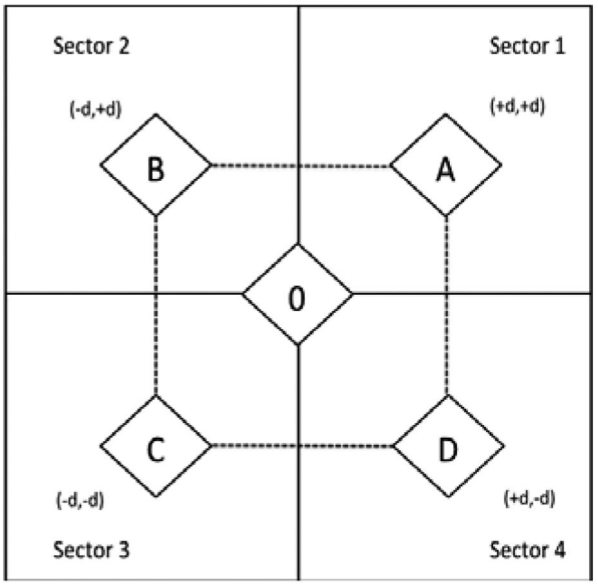

Figure 4. MPDS search in five regions of the search area.

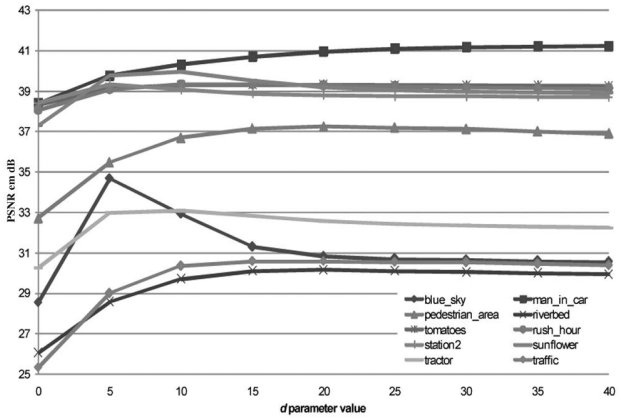

Figure 5. Variation of d parameter for MPDS algorithm.

parameter variation for different kind of motion activity videos. For low motion activity videos, as blue_sky, the MPDS algorithm can reach the best results with d=5, with more than 6dB gain in comparison to DS algorithm. For pedestrian_area sequence, for example, the MPDS reaches its best result for *d*=20, with a PSNR gain superior to 4.5dB in relation to DS.

The same experiment was made for 16x16 block sizes (focus of this work), to identify the *d* value that maximize the average results of MPDS algorithm (considering the ten sequences). Using *d*=10 the MPDS algorithm can achieve the best average gain in comparison to DS algorithm (1.69dB). This was the used value for d parameter in the rest for MPDS.

As it is possible to see from Figure 5, the optimum value of *d* depends a lot on the sequence being processed, which has a significant impact on digital video quality. However, the MPDS only allows the use of a static *d* parameter for all frames in the video sequence. The DMPDS algorithm is an evolution of the MPDS, where the *d* parameter value can be dynamically adapted. Thus, it is possible to achieve better quality with the variation of the *d* parameter value during the execution, according to the video characteristics.

The variations of the *d* parameter are influenced by the characteristic of the current scene. For a low motion activity scene, the *d* parameter value can be dynamically reduced, resulting in better quality results. If the characteristic changes, and the motion activity is increased, the *d* parameter value can be dynamically increased. This dynamic adjustment enhances the robustness of the algorithm to deal with any kind of video sequences.

The algorithm to define the *d* parameter is presented in Figure 6. Firstly, an initial *d* value and a dynamic variation (Δ) are set. The first frame is processed with original *d* parameter, the second frame with $d1 = d - \Delta$ and the third one with $d2 = d + \Delta$. The frame that obtains the lowest SAD becomes the new d and the dynamic variation (Δ) is divided by two. This process is repeated until the dynamic variation reach the value 1, when the oscillation becomes 1 until the algorithm is reset. This technique tries to dynamically adapt the optimal *d* value for each frame. The DMPDS starts with the same value as MPDS for *d* parameter ($d = 10$), and $\Delta = 5$.

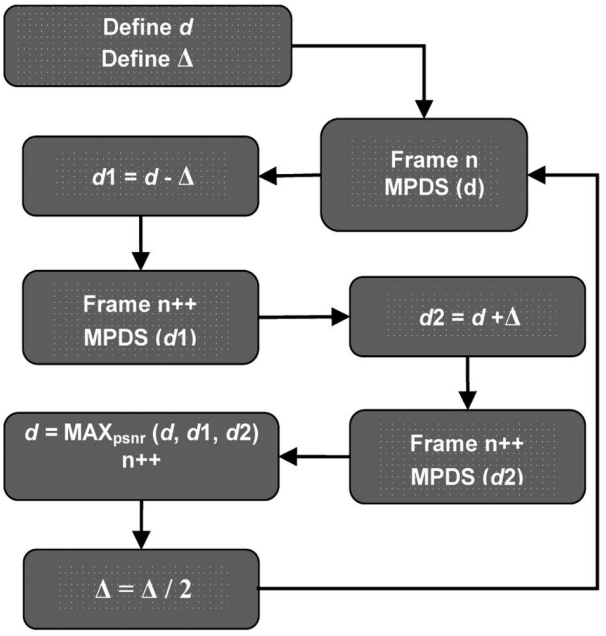

Figure 6. Dynamic control of the *d* parameter in the DMPDS algorithm.

It is important to notice that the MPDS and DMPDS algorithms could be executed in parallel or in a sequential fashion. Each sector can execute its search independently and in parallel with the other sectors. These algorithms were developed focusing on the quality of the results generated by the ME process, so both implementations, parallel or sequential, could be used since the quality results will be the same.

The MPDS and DMPDS algorithms are efficient solutions to be executed in processors with support to multi-thread or multi-core, exploring parallel programming features. It is possible because each DS core does not have any data dependencies with the others, so it is possible to perform all of them in parallel.

A. Software Evaluation

The average results (considering the first 200 frames of the ten test sequences) for quality and computational cost of MPDS and DMPDS algorithms with 16x16 block size are presented in Table I, where results for the traditional DS algorithm and the FS are also presented. All results in Table I considered the SAD as the similarity criterion. The computational cost is showed in number of Evaluated Candidate Blocks (ECB).

Comparing the MPDS algorithm to DS, it is possible to achieve an average gain of 1.73 dB, with an increase of six times in the computational cost. The best MPDS results comparing to DS is achieved in Traffic sequence. In this sequence the MPDS achieves a gain of 3.95 dB with an increase of 5.4 times in the computational cost. The worst MPDS results, in comparison to DS, is achieved in station2 sequence. In this case, it was possible to achieve a gain of 0.63 dB with an increase of 6.5 times the computational cost. This result was obtained because this video does not present much motion activity, and good vectors can be found around the center, making the DS quality near the MPDS.

Comparing to FS, the MPDS presents an average quality loss of 1.15 dB, however, it reduces the complexity by more than 47 times. The best case comparing to FS is achieved in rush_hour, when the MPDS losses is only 0.31dB with a reduction in complexity of 47.2 times.

The DMPDS achieves an average gain of 0.12 dB in comparison with MPDS, which was expected because of the dynamic model insert in the *d* parameter. The DMPDS computational cost increasing in only about 3% when compared to MPDS, which is a good trade-off between video quality and complexity. The best DMPDS result is achieved in the *blue_sky* video, with a gain of 0.5dB over the MPDS, and also with a lower number of comparisons. The best *d* value for the blue_sky video in MPDS algorithm is five, as presented in Figure 5. So, the DMPDS algorithm can reach this value in two steps, generating best results with lower comparisons, while the MPDS algorithm is using a static d=10.

The worst result for DMPDS algorithm was achieved at Traffic sequence, with a loss of 0.24 dB in relation to MPDS. This decrease in the video quality happened because the *d* parameter did not get sufficient time to adapt itself to the video characteristic, while in MPDS the *d* got a static, however, good value. Probably, for a higher number of encoding frames, the DMPDS would achieve a better quality than MPDS, because it would get adapt to the video characteristic.

The DS algorithm does not have any limitation in the number of iterations. This is a problem if we want to design this algorithm in hardware, since the number of iterations, and consequently the number of clock cycles, are non deterministic. Results targeting hardware implementation, considering the restriction in the number of iterations, are presented in Table II. In this case, the evaluation considers five and eleven iterations restriction for MPDS, DMPDS and DS algorithms. Same restriction is used in the hardware architecture design presented in next section.

Table II. Average Results for MPDS, DMPDS and DS.

Algorithm	PSNR (dB)	#ECB ($x10^9$)	SADS ($x10^9$)
DMPDS	34.87	0.32	81.55
MPDS	34.75	0.31	78.72
DS	33.02	0.05	12.33
DMPDS 4:1	34.65	0.30	19.29
MPDS 4:1	34.51	0.29	18.56
DS 4:1	32.76	0.05	2.97
DMPDS 4:1 5 iter.	33.67	0.24	15.54
MPDS 4:1 5 iter.	33.28	0.22	14.59
DS 4:1 5 iter.	31.42	0.04	2.64
MPDS 4:1 11 iter.	34.19	0.27	17.72
DS 4:1 11 iter.	32.43	0.04	2.89

Table I. Average Results for DS, MPDS, DMPDS and FS.

Video	DS		MPDS		DMPDS		FS	
	PSNR (dB)	#ECB ($x10^9$)	PSNR (dB)	#ECB ($x10^9$)	PSNR (dB)	#ECB ($x10^9$)	PSNR (dB)	#ECB ($x10^9$)
blue_sky	30.38	0.04	33.23	0.27	33.73	0.24	34.51	14.66
man_in_car	38.15	0.03	39.41	0.22	39.60	0.24	40.34	14.66
pedestrian_area	32.56	0.05	35.05	0.33	35.25	0.34	36.15	14.66
Riverbed	24.61	0.06	26.48	0.35	26.86	0.36	27.88	14.66
rolling_tomatoes	37.76	0.03	38.27	0.25	38.32	0.28	38.65	14.66
rush_hour	36.70	0.03	37.29	0.31	37.28	0.36	37.60	14.66
station2	38.00	0.04	38.39	0.26	38.50	0.22	38.80	14.66
Sunflower	37.31	0.05	38.68	0.37	38.53	0.43	39.11	14.66
Traffic	25.10	0.06	29.05	0.38	28.81	0.39	33.38	14.66
Tractor	29.65	0.07	31.69	0.35	31.85	0.33	32.54	14.66
Average	33.02	**0.05**	**34.75**	**0.31**	**34.87**	**0.32**	35.90	**14.66**

Table II also presents the evaluation considering the use of 4:1 pixel sub-sampling. This is a useful technique to reduce the number of calculations with low impact in the quality results, which can be seen in the SADs column, which presents the SADs calculations. The use of pixel sub-sampling is an interesting strategy, especially for hardware implementation, since the internal memory and processing unities can be reduced. The MPDS and DS also have been evaluated for 11 iterations.

The PSNR gain of DMPDS algorithm over DS and MPDS increases when the iterations restrictions are considered. The increase in the number of comparisons is reduced in this scenario. The DMPDS PSNR gain over DS can reach 2.25dB considering the 4:1 subsampling and the restriction of five iterations. In this case, the number of comparisons calculations is six times higher for the DMPDS when compared to DS. If no iteration restriction neither subsampling is considered, the DMPDS PSNR gain is of 1.85 dB and the number of comparison calculations is 6.5 times higher than DS.

4. DESIGNED ARCHITECTURES

The MPDS and DMPDS architectures work with 16x16 blocks, and uses 4:1 pixels sub-sampling. An architecture was designed to perform the DS algorithm and it was used as ME core to implement the MPDS and DMPDS algorithms. Each DS core is restricted to five iterations. The DMPDS architecture is strongly based on the MPDS architecture. MPDS and DMPDS architectures performance is focused on real time processing for HD 1080p and QFHD videos, with low area cost and low power consumption.

Two versions of the MPDS architecture were implemented, one considering eleven and other considering five iterations restriction. These versions are very similar, since there are differences only in the control unit. The difference in the number of iterations impacts in the architecture processing rate, since more iterations imply in more clock cycles to generate a motion vector. The first version (with 5 iterations) was modified to implement the DMPDS architecture.

Figure 7 presents the block diagram of DS core architecture. The architecture must fill the reference memory and current memory in advance to start the search process. The reference memory size is 34x34 bytes, which contains the data for the first LDSP (Large Diamond Search Pattern), 5 LDSP iterations and the data for the final refinement (SDSP – Small Diamond Search Pattern). This size was choose because the 5 iterations DS will need only to access the main memory once to fill with the data for all iterations and the 11 iterations DS must access the main memory only twice to fill with data for all iterations.

The DS core needs 34 cycles to fill its internal memory. To optimize the architecture performance, the necessary data for the first LDSP is firstly loaded from the external memory. Then, the datapath starts to work while the reference memory is fed with data for next iterations. The datapath finishes the first LDSP before the reference memory is fully fed.

When the reference memory is filled with data for the first LDSP, the 13 local memories (presented in Figure 7) will be filled, each one with a candidate block (9 for the first LDSP and 4 for the SDSP). The local memories MEM 1 to MEM 9 contain the data used for SAD calculation for the LDSP. Memories from MEM A to MEM D contain the data for the SDSP processing. When the algorithm decides to start the SDSP, the candidate blocks are still available in MEM A to MEM D. There is no additional memory latency to perform the SDSP step.

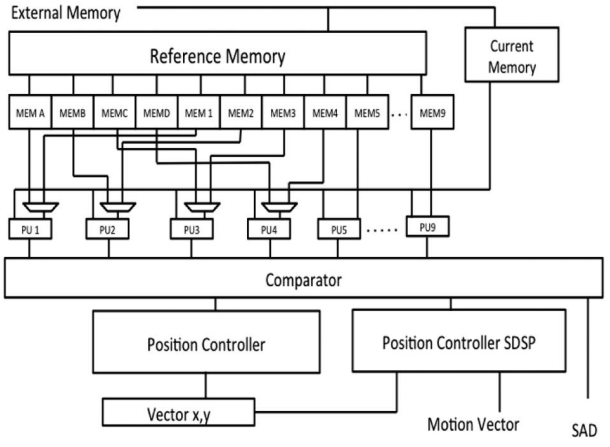

Figure 7. Block diagram of the DS core architecture.

The DS core architecture uses nine processing units (PU), one for each candidate block of the LDSP. Then, the nine blocks of the LDSP can be processed in parallel. To reduce the hardware consumption, four PUs are also used to calculate the SDSP. The PU was designed in a pipeline of five stages and it is basically an adder tree to calculate the SAD. The PU is able to process 2 lines per cycle.

The comparator sends the best block to the position controller in Figure 7. Which is responsible to update the new position of this block, since this is necessary to generate the motion vector. If the best candidate block is found in the center (PU 5), the SDSP is triggered and the position controller SDSP in Figure 7 will generate the final motion vector for this block in this DS core. When the best match is not found in the center, the architecture starts to process a new iteration. However there is no memory latency, since reference memory was previously loaded with the data for five iterations. In the worst case scenario, this core generates a motion vector in 169 clock cycles. This occurs when the core needs to perform all the five iterations. For the eleven iteration case, the architecture process the five iterations in the same way and then is fed again with data for the next 6 iterations, and process again this iterations in 169 clock cycles.

The block diagram of the MPDS and DMPDS architectures is presented in Figure 8. Both architectures use the same architectural template. Each core in Figure 8 is composed by one DS architecture presented in Figure 7. The comparator in Figure 8 is responsible to find the best SAD among the best SADs of the five cores. At the beginning, the architecture starts to fill the reference memory of core 0 (the central one). As soon as core 0 internal memory is completely filled, the architecture continues to fill the next core, while core 0 starts its search. This process is repeated until the reference memory of core D is completely filled.

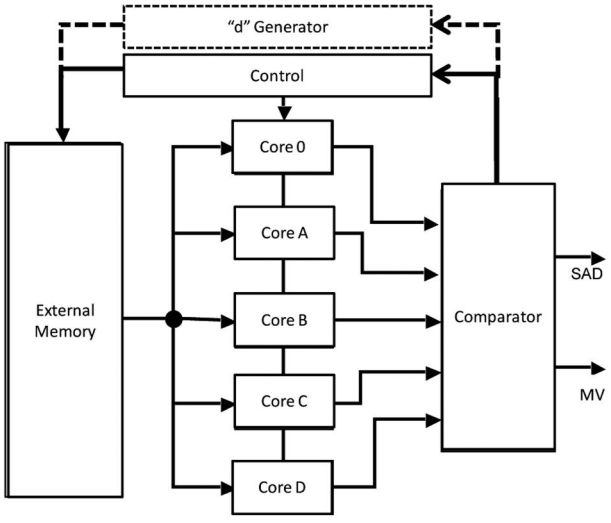

Figure 8. Block diagram of the MPDS architecture.

The first version of MPDS architecture can store the data for its five iterations in its internal memories. When the core D finished being filled, the architecture automatically starts to fill again the core 0, with data for the next motion vector generation. Both architectures have a memory latency of 170 cycles to fill all cores reference memories. More 135 cycles are necessary to the core D finishes its processing, and three more cycles are necessary to find the best result among all cores. The first vector needs 309 cycles to be generated, after that, only 170 cycles is needed to generate a new motion vector. It happens because when the first vector is ready, the cores 0, A, B and C are already working on the next block. The DMPDS architecture can generate a motion vector with the same number of clock cycles, since there is no latency for the *d* parameter generation.

The only difference from the DMPDS to MPDS architecture is in the "*d*" generator block. This block is presented only in the DMPDS architecture, and is composed of a state machine based on Figure 6. The start value of the dynamic parameter Δ is 5 and the start value of *d* is 10. The DMPDS architecture starts the process on frame one with $d = 10$. The second frame is processed with $d = 5$, and the third one is processed with $d = 15$. Then the *d* is updated with the value that obtained the lowest SAD and the dynamic parameter Δ is divided by 2. This process is repeated with the new *d* and the new dynamic parameter is equal to 1. The variation of the *d* parameter becomes ±1 pixel until the architecture is reset.

Figure 9 shows a time diagram of the proposed architectures. The second version of MPDS architecture needs to update the reference memories of all cores to perform eleven iterations. After every core finishes its processing, then the core 0 starts to be fed with data for the next six iterations. As this version needs to feed twice the internal memories cores, it generates a motion vector at every 340 cycles. The latency for the generation of the first vector is 479 cycles.

Both versions of the MPDS architecture and DMPDS architecture are able to maintain the cores operating during almost all clock cycles and almost no wasting time is necessary to allow the data fed for the architecture, each core is processing almost 99.5% of time. Then, these solutions can efficiently explore the used hardware for a high performance operation.

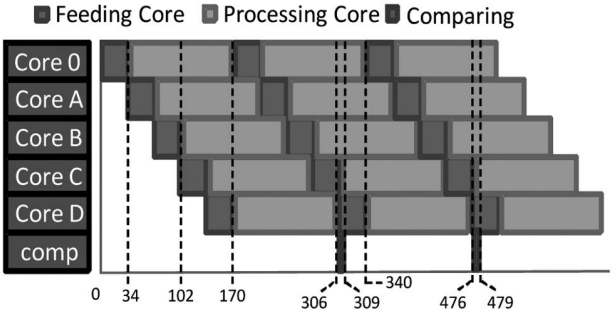

Figure. 9. Time diagram of the proposed architecture.

5. RESULTS AND COMPARISONS

Our two versions of the MPDS architectures were described in VHDL and synthesized to EP4S40G2F40I2 Altera Stratix 4 FPGA and for TSMC 90nm standard cells technology. The synthesis results of the MPDS architecture version 1 (V. 1) and version 2 (V. 2) are presented in Table III. The DMPDS architecture was described in VHDL and synthesized only to Altera Stratix 4 FPGA.

The ASIC synthesis considers the minimum operational frequency to process 30 HD 1080p frames per second, and also 30 frames QFHD per second for the MPDS V. 1 architecture.

Both versions of MPDS have very similar area resources utilization and the same memory bits usage. The main differences are in the achieved performance and power consumption. The MPDS architecture V. 1 can achieve a higher operating frequency for FPGA synthesis, and can achieve real time processing for HD 1080p videos with only 41.3MHz. The MPDS V. 2 needs the double of this frequency to process 30 HD 1080p frames per second, since more iterations are allowed in the same hardware. The performance results consider the worse case, when every core uses all the available iterations (five in V. 1 and 11 in V. 2).

The power results were obtained only from the ASIC synthesis. Synopsys tool was used to the synthesis. Due to the low operating frequency and the 90nm technology, the V. 1 of the MPDS architecture can process real time HD 1080p videos consuming only 4.5mW. The power consumption of version two is about the double, since this version needs an operating frequency twice times higher than V. 1. The MPDS V. 1 architecture has also been synthesized with a frequency capable to process QFHD videos in real time, and obtained the same area results with only 18mW of power consumption.

The DMPDS iteration limit is 5 iterations per DS. Comparing the MPDS to DMPDS results in FPGA, the DMPDS uses almost the same hardware resources and achieves a lower maximum frequency. On other hand, the DMPDS is able to achieve a better digital video quality. ASIC synthesis of DMPDS architecture would achieve similar power consummation and area utilization as MPDS V. 1.

Table III also presents results for related works [6], [7], [8], [9], [10] and [11]. The work [6] proposes fractional motion estimation (FME) algorithm with variable block size motion estimation. Comparing to [6], our solutions can achieve real time for HD 1080p videos with lower frequencies. This performance gain could be achieved with the efficiency of the proposed algorithms and a larger internal memory. Comparing the area and power results, the MPDS and DMPDS architectures use much less hardware than [6] and the power consumption is much lower.

The work [7] performs ME with the Dynamic Variable Step Search (DVSS) fast algorithm. The architecture in [7] uses the same block size of our architecture and at 130MHz it is able to process 34.3 HD 1080p frames per second. Again, our architectures need a lower frequency to process the same resolution. Comparing our FPGA synthesis results, the architecture presented in [7] needs less hardware and internal memory than MPDS and DMPDS. However, our solutions can achieve a higher operation frequency, and a considerably higher frame rate.

The architecture presented in [8] performs the Hexagon Based-Search (HEXBS), Block Based Gradient Descent Search (BBGDS) and Three Step Search (TSS) algorithms. This architecture needs 390, 437 and 680 cycles to process one block, for each algorithm, respectively. The MPDS and DMPDS architectures can process one block using fewer cycles, comparing to any configuration of the architecture presented in [8]. Comparing the power and area results, our works needs more hardware resources, however, it spends much less power.

The work [9] presents a hardware architecture for the Fast Top-Winners Search Algorithm. This architecture uses about half hardware resources and less memory bits, in comparison with our solutions. However this architecture does not reach real time for HD 1080p videos.

The architecture presented in [10] uses Multi-Resolution ME Algorithm (MMEA). This solution uses much more hardware resources, since it considers two reference frames in the ME process. It needs a much higher operational frequency to achieve similar performance than our solutions. The power results are not presented, however, it seems to be higher than ours, due to its higher frequency.

The work [11] presents the Adaptive True Motion Estimation Algorithm (ATME) and it also uses techniques for Frame Rate Up-Conversion (FRUP). It also evaluates the algorithm for HD 1080p videos. Comparing to [11], our solutions use almost the same hardware resources, but [11] processes a 16x16 block in 104 cycles, which is less cycles than our architecture. However, our architecture reaches a better operating frequency, which is responsible to increase this work processing rate in comparison with [11].

It is difficult to compare the quality results among the related works, since almost none of them except [11] evaluate their algorithms and architectures for HD 1080p videos. However, [6-10] do not mention any alternative to avoid local minima falls, so probably these architectures will process high resolution videos with a worst quality if compared with our work.

Our solutions obtained the lowest power among all presented architectures for the ASIC synthesis. The MPDS V.1 (ASIC and FPGA) and DMPDS architectures are the only ones with sufficient performance for real time processing in QFHD videos.

Table III. Synthesis results and comparisons.

Architecture	Technology	Frequency (MHz)	Cycles per Block	AREA	Memory 1080p (fps)	Power bits (K)	HD (mW)	QFHD (fps)
Kao [6]	180nm	154	631	321 KGates	9.72	374	30	7.5
Tasdizen [7]	Virtex 5	130	467	2282 KCLBs	0.51	n.a.	34	8.5
Vanne [7]	130nm	200	390 / 437 / 680	14 KGates	2.5	59	63 / 56 / 36	15.75 /14 / 9
Lai [8]	180nm	83.3	1282	26 KGates	28.7	60.84	8	2
Yin [10]	180nm	200	872	260 KGates	11.3	n.a.	28	7
Cetin [11]	FPGA 90nm	63	104 (average)	33 KLUTS	8 dual-port block RAM	n.a	74.7	18.7
MPDS V. 1	**90nm**	**41.3**	**170**	**50 KGates**	**82**	**4.5**	**30**	**-**
MPDS V. 1	**90nm**	**165.2**	**170**	**50 KGates**	**82**	**18**	**-**	**30**
MPDS V. 1	**Stratix 4**	**199.2**	**170**	**34.5 KALUTs**	**46.2**	**n.a.**	**144.6**	**36.2**
MPDS V. 2	**90nm**	**82.6**	**340**	**50 KGates**	**82**	**9**	**30**	**-**
MPDS V. 2	**Stratix 4**	**185.2**	**340**	**34.5 KALUTs**	**46.2**	**n.a.**	**67**	**16,75**
DMPDS	**Stratix 4**	**187.58**	**170**	**34.5 KALUTS**	**46.2**	**n.a**	**136**	**34**

6. CONCLUSIONS

This paper presented two new motion estimation algorithms and the respective hardware architecture for high definition videos. These architectures performs the MPDS and for the DMPDS algorithms. These algorithms were developed focused on high definition videos, and are more resilient to local minima falls than traditional fast ME algorithms, increasing the quality results. Two versions of the MPDS architecture were presented: one focusing on high performance, and other focusing in quality improvements. Both architectures were synthesized for Stratix 4 FPGA and 90nm standard cells technology. The synthesis results show that both solutions are able to process real time HD 1080p videos with low operation frequencies and low power. The MPDS architectures present the lowest power among all presented related works, and can also work with QFHD videos in real time. A DMPDS architecture was also developed and synthesized for FPGA, with sufficient performance for real time QFHD video processing. Our solutions are the only among the related works that are able to achieve real time processing QFHD videos.

REFERENCES

[1] JVT Editors (T. Wiegand, G. Sullivan, A. Luthra), Draft ITU-T Recommendation and final draft international standard of joint video specification (ITU-T Rec.H.264|ISO/IEC 14496-10 AVC), 2003.

[2] Cheng,Y. , et al. 2009. An H.264 Spatio-Temporal Hierarchical Fast Motion Estimation Algorithm for High-Definition Video. IEEE ISCAS, pp. 880-883, May 2009.

[3] I. Richardson, Video codec design: developing image and video compression systems. Wiley, 2002.

[4] ZHU, S., MA, K. 2000. A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation. IEEE Transactions on Image Processing, vol. 9, n. 2, pp. 287-290, Feb 2000.

[5] Xiph.org: Test media, available at <http://media.xiph.-org/video/derf/>, Oct, 2011.

[6] Kao, C., et al. 2009. A High-Performance Three-Engine Architecture for H.264/AVC Fractional Motion Estimation. IEEE Transactions on Very Large Scale Integration Systems, vol. 18, n. 4, pp. 662-666, April 2009.

[7] Tasdizen, O., et al. 2009. Dynamically variable step search motion estimation algorithm and a dynamically reconfigurable hardware for its implementation. IEEE Transactions on Consumer Electronics, vol. 55, n. 3, pp. 1645-1653, Aug. 2009.

[8] Vanne, J., et al. 2009. A Configurable Motion Estimation Architecture for Block-Matching Algorithms. IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, n. 4, pp. 446-476, April 2009.

[9] Lai, Y., et al. 2010. Hybrid Parallel Motion Estimation Architecture Based on Fast Top-Winners Search Algorithm. IEEE Transactions on Consumer Electronics, vol. 56, n. 3, pp. 1837-1842, Aug. 2010.

[10] Yin, H., et al. 2010. A Hardware-Efficient Multi-Resolution Block Matching Algorithm and Its VLSI Architecture for High Definition MPEG-Like Video Encoders. IEEE Transactions on Circuits and Systems for Video Technology, vol. 20, n. 9, pp. 1242-1254, Sept. 2010.

[11] Cetin, M., et al. 2011.An Adaptive True Motion Estimation Algorithm for Frame Rate Conversion of High Definition Video and Its Hardware Implementations, IEEE Transactions on Consumers Electronics, vol. 57, n. 2, May 2011.