FLEXIBLE MOTION ESTIMATION PROCESSORS FOR HIGH DEFINITION VIDEO CODING

Trevor Spiteri, George Vafiadis, Jose Luis Nunez-Yanez

Department of Electrical and Electronic Engineering
University of Bristol, UK
email: trevor.spiteri@bristol.ac.uk, vafiadis@ieee.org, j.l.nunez-yanez@bristol.ac.uk

ABSTRACT

This paper presents a reconfigurable motion estimation processor suitable for high definition video coding. A toolset for the design of a video coding system is presented as well. The presented tools can be used in the design and configuration of the reconfigurable processor itself. They can also be used to design user-defined block-matching motion estimation algorithms. Using the tools, the processor's design space may be explored in order to find configurations suitable for high definition video. The experiments presented show the effect of modifying the processor configuration on the performance obtained when coding high definition video sequences, and the results indicate that for high definition video, supporting sub-partitioning offers no gain for the increase in complexity.

1. INTRODUCTION

Video compression is an integral part of many multimedia applications, many of which require real-time operation and a high compression performance. New advanced coding standards, such as VC-1, AVS and H.264 [1], make use of advanced techniques to achieve high compression. Previous work [2] shows that motion estimation is the most expensive operation in the H.264 encoder, representing up to 90% of the total complexity. This makes it desirable to have specialized hardware for motion estimation.

Full search motion estimation algorithms have gained popularity in hardware implementations owing to their regularities, which make it possible to implement motion estimation hardware using systolic arrays [3]. Other approaches, such as the hexagonal search algorithm [4], the unsymmetrical multi-hexagonal (UMH) search algorithm [5], and many others, do not perform the search on full point regions. The use of these block matching algorithms can make the estimation process faster by requiring less computations than a full search. Although the full search algorithm is usually believed to yield optimal rate distortion performance, it has been shown that a well-designed fast block matching algorithm can provide better rate-distortion performance owing to its ability to track real motion more accurately [6].

There are various hardware implementations of motion estimation algorithms. Processors with instruction set architectures (ISA) similar to the proposed work, tailored for block-matching search algorithms, are presented in [7] and [8]. Xilinx have a motion estimation engine [9] that computes the sum of absolute differences (SAD) for a set of 120 search locations within a 112×128 search window in parallel. None of these cores offer the possibility of matching the hardware architecture and the search algorithm to optimize performance as the presented work does.

The motion estimation process can be performed in various different ways, and it is up to the designer to choose the strategy. Apart from the search strategy itself, other choices include whether to use multiple motion vector candidates in the search, the number of reference frames to which to compare the macroblocks, whether the macroblocks are split into partitions, whether to perform sub-pixel interpolation and search, and whether to include the cost of encoding the motion vector itself during estimation. Because of the number of design parameters and their complexity, the design space can be very large, and exploring this design space to find design parameters that are optimal can be complex and ultimately application dependent.

A toolset has been developed for the optimization and generation of configuration data for a high-performance motion estimation processor. The toolset makes the process of finding the optimal hardware configuration and software parameters faster. A cycle-accurate simulator is included, making it possible to change the parameters and test the configuration in a short time without requiring hardware access. There is already similar work on configurable generic processors, like the Xtensa configurable processor [10] from Tensilica. Designers can choose configuration parameters and generate a custom processor optimized for their needs. The options include support for 16×16-bit multiplication, a floating point unit, a barrel shifter, and others. Tensilica also provides the XPRES compiler, a tool for design space exploration.

The paper is organized as follows. Section 2 reviews the hardware architecture of our configurable motion estimation processor. Section 3 describes the integrated development

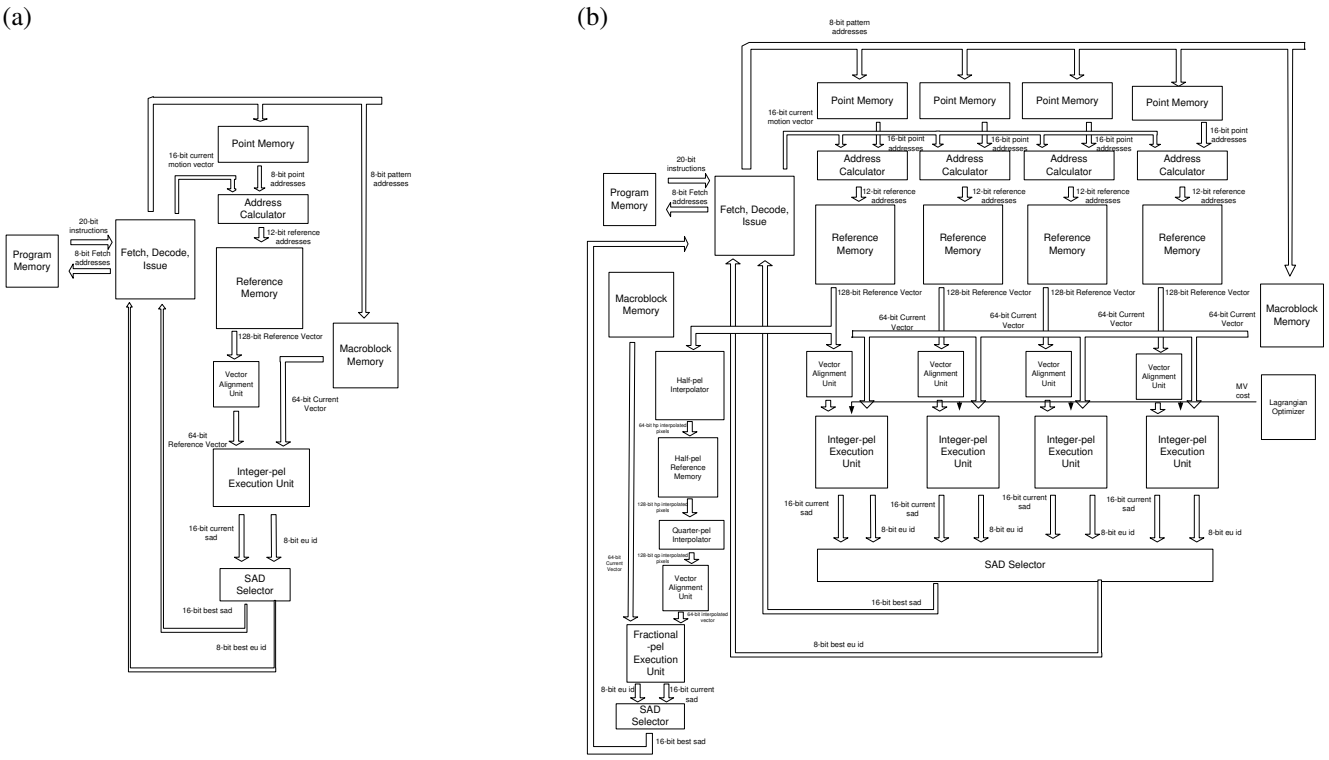

Fig. 1. (a) Base configuration and (b) complex configuration of the motion core.

environment for the design of motion estimation algorithms. Section 4 presents the cycle-accurate simulator and how it can be used to analyse and optimize motion estimation algorithms. Section 5 presents some experiments using HD sequences. Finally, section 6 draws conclusions.

2. HARDWARE OVERVIEW

The LiquidMotion processor is a reconfigurable application-specific instruction-set processor (ASIP) developed in our group. It is designed to execute user-defined block-matching motion estimation algorithms optimized for hybrid video codecs such as MPEG-2, MPEG-4, H.264/AVC and Microsoft VC-1. The core offers scalable performance dependent on the features of the chosen algorithm and the number and type of execution units implemented. Hardware configuration can typically be achieved at compile time by adapting the architecture to the chosen algorithm, and in a field-programmable gate array (FPGA) implementation, it is possible to pre-compile a range of hardware bitstreams with different configurations from which one can be chosen to match the current video processing requirements. The microarchitecture can be easily scaled to high definition (HD) video even when using low cost FPGAs such as the Xilinx Spartan-3. The ability to program the search algorithm to be used, and the ability to reconfigure the underlying hardware that it will execute on, combine to give an extremely flexible video processing platform. A base configuration consisting of a single 64-bit integer pipeline, capable of processing a hexagonal motion estimation algorithm, such as the one implemented in the x264 [11] video encoder, over a search window of 112×128 pixels in real-time for high-definition video, can be implemented in 2300 logic cells on a Xilinx FPGA. In contrast, a complex configuration supporting motion vector candidates, sub-blocks, motion vector costing using Lagrangian optimization, 4 integer-pel execution units (IPEU) and 1 fractional-pel execution unit (FPEU) plus sub-pel interpolator execution unit (SPIEU) will need around $14,600$ logic cells. A simplified diagram comparing these two configurations is shown in Fig. 1. At least 1 IPEU must always be present to generate a valid processor configuration but the other units are optional, and are configured at compile time. Each execution unit uses a 64-bit wide word and a deep pipeline to achieve a high throughput. All the accesses to reference and macroblock memory are done through 64-bit wide data buses and the SAD engine also operates on 64-bit data in parallel. The memory is organized in 64-bit words and typically all accesses are unaligned, since they refer to macroblocks that start in any position inside this word. By performing 64-bit read accesses in parallel from two memory blocks, the desired 64 bits across the two words can be selected inside the vector alignment unit.

The engine also supports half- and quarter-pel motion

Table 1. Comparison of different implementations for a diamond search pattern.

Processor impl.	Cycles per MB	FPGA slices	Virtex-II clock	Memory (BRAMS)
Intel P4 assembly	~ 3000	N/A	N/A	N/A
Dias et al. [7]	4532	2052	67 MHz	4 (external reference area)
Babionitakis et al. [8]	660	2127	50 MHz	11 (1 ref. area, 48×48 pixels)
Proposed, 1 IPEU	510	1231	125 MHz	21 (2 ref. areas, 112×128 pixels)
Proposed, 2 IPEUs	287	2051	125 MHz	38 (2 ref. areas, 112×128 pixels)

estimation, owing to an SPIEU and specifically designed FPEUs. The number of SPIEUs execution units is limited to 1 but the number of FPEUs can be configured at compile time. The SPIEU interpolates the 20×20 pixel area that contains the 16×16 macroblock corresponding to the winning integer motion vector. The interpolation hardware is cycled 3 times to calculate first the horizontal pixels, then the vertical pixels, and finally the diagonal pixels. The SPIEU calculates the half pels through a 6-tap Wiener filter as defined in the H.264 standard. The SPIEU has a total of 8 systolic one-dimensional (1-D) interpolation processors with 6 processing elements each. The objective is to balance the internal memory bandwidth with the processing power so in each cycle, 8 valid pixels are presented to one interpolator. Quarter-pel interpolation is done when required by reading the data from two of the four memories containing the half and full pel positions, and averaging according to the H.264 standard. The fractional pipeline and the integer pipeline work at the same rate and process one search point in 33 cycles. To maintain this data rate, each FPEU needs two vector alignment units so two half or integer pel 64-bit vectors are presented in each cycle to the quarter-pel interpolation unit.

Table 1 compares the complexity and performance of the proposed processor core implementation to that of other implementations. The IPEUs and FPEUs have been carefully pipelined, and all the configurations can be implemented to achieve a clock rate of 200 MHz when targeting the Virtex-4 Xilinx family. More details can be obtained in [12].

3. DESIGNING MOTION ESTIMATION ALGORITHMS

The Estimo C language is a high-level C-like language that is aimed at designing a broad range of block-matching algorithms. The code can be developed and compiled in the SharpeEye Studio [13], an integrated development environment (IDE) for motion estimation. The language contains a preprocessor for macro facilities that include conditional (*if*) and loop (*for*, *while*, *do*) statements. The language also has facilities directly related to the motion estimation processor's instruction set, such as checking the SAD of a pattern consisting of a set of points, and conditional branching depending on which point from the last pattern check command had the best SAD. The compiler converts the program to assembly and then to a program memory file containing instructions and a point memory file containing patterns.

Estimo C source code

```
s = 8; // initial step size

check(0, 0);
check(0, s);
check(0, −s);
check(s, 0);
check(−s, 0);
update;

do {
  s = s/2;
  for (i = 1 to 5 step 1) {
    check(0, s);
    check(0, −s);
    check(s, 0);
    check(−s, 0);
    update;
    #if (WINID == 0)
      #break;
  }
} while (s > 1);

for (x = −0.5 to 0.5 step 0.25)
  for (y = −0.5 to 0.5 step 0.25)
    check(x, y);
update;
```

Program memory

```
00: 0 05 00 check 5 pts, offs 00
01: 0 04 05 check 4 pts, offs 05
02: 2 00 0b if WINID is 0, goto 0b
03: 0 04 05 check 4 pts, offs 05
            ...
0b: 0 04 09 check 4 pts, offs 09
0c: 2 00 15 if WINID is 0, goto 15
            ...
15: 0 04 0d check 4 pts, offs 0d
16: 2 00 15 if WINID is 0, goto 1f
            ...
1f: 1 04 0d chk 25 frac pts, offs 11
    ↑
 opcode
    0       integer check pattern
    1       fractional check pattern
    2       conditional jump
```

Point memory

```
00: 00 00 integer (0, 0)
01: 00 08 integer (0, 8)
02: 00 f8 integer (0, −8)
          ...
11: fe fe fractional (−0.5, −0.5)
12: fe ff fractional (−0.5, −0.25)
          ...
29: 03 03 fractional (0.5, 0.5)
```

Fig. 2. The Estimo C code for a motion estimation algorithm and excerpts of the target files generated by the compiler.

Fig. 2 shows an example block-matching algorithm written in Estimo C and excerpts from the target files. The algorithm is a diamond search pattern executed for up to 5 times for a radius of 8, 4, 2, and 1 pixels, followed by a small full search at fractional pixel level. The first set of *check*() and *update*() commands create the first search pattern, which consists of 5 points. Each *check*() command adds a point to the search pattern being constructed, and the *update*() command completes the pattern. This pattern is compiled into the instruction at program address `00`, which uses the 5 points available in the point memory at addresses `00`–`04`. The preprocessor goes through the *do while* loop 3 times, with s taking the values 4, 2 and 1. Each time, a 4-point pattern is checked for up to 5 times. The *#if*(*WINID* $==$ 0) *#break* command ensures that if a pattern search does not improve the motion vector estimate, it is not repeated. The final lines create a 25-point fractional pattern search.

4. CYCLE-ACCURATE SIMULATOR

Designers may need to know how much time a particular algorithm takes to determine the motion estimation vectors. It

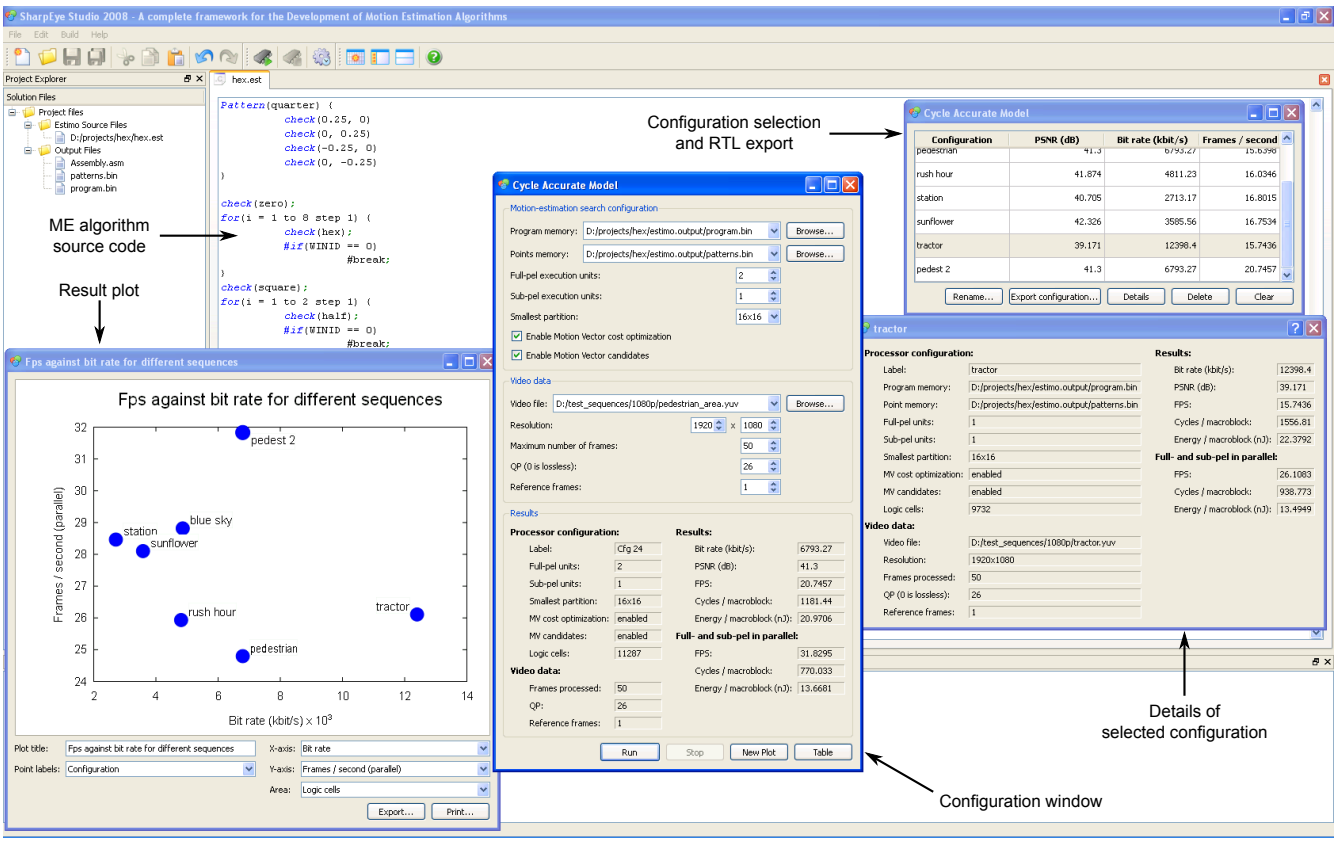

Fig. 3. Screenshot of the SharpEye IDE used to analyse motion estimation algorithms and processor configurations.

would also be very useful to be able to choose configuration parameters for the motion estimation processor depending on the particular requirements of the design.

Doing this analysis on the actual processor can be complicated and time consuming. The tasks required include synthesizing a processor with some specific configuration and measuring the time used by the processor to perform the motion estimation. A cycle-accurate simulator of the processor can speed up the development cycle significantly by reducing the number of tasks required by the designer to analyse a particular configuration. Additionally, the designer does not need access to the hardware when using the simulator.

A cycle-accurate model of the processor was developed as part of the toolset. x264 [11], a free software library for encoding H.264, was modified to use the cycle-accurate model; motion estimation in x264 was modified to use the cycle-accurate model instead of its own block searching algorithms when searching for the motion vectors.

The cycle-accurate simulator can be used directly from the SharpEye IDE described in Section 3. The designer can design a motion estimation algorithm and test it using different processor configuration parameters. Fig. 3 shows a sample session.

The simulator takes several parameters as inputs. The inputs which determine the processor configuration are: the program and point memories generated by the Estimo compiler, the number of IPEUs and FPEUs, the minimum size for block partitioning, whether to use motion vector cost optimization, and whether to use multiple motion vector candidates. The simulator takes other options which do not affect the processor configuration itself, which are: the video file to process and its resolution, the maximum number of frames to process, and the quantization parameter (QP).

The simulator will then process the video file using the selected search algorithm and processor configuration, and give the following outputs: the bit rate of the compressed video, the peak signal-to-noise ration (PSNR), the number of frames processed per second (fps) assuming a clock rate of 200 MHz, the number of clock cycles required per macroblock, and the energy requirements per macroblock.

The designer can simulate and analyse various configurations by using the simple controls in the configuration window, and then generate plots or view the results in a table. When he is satisfied with a particular configuration, he can generate a VHDL file which can be used together with the rest of the core hardware register transfer level (RTL) library to synthesize the motion estimation processor.

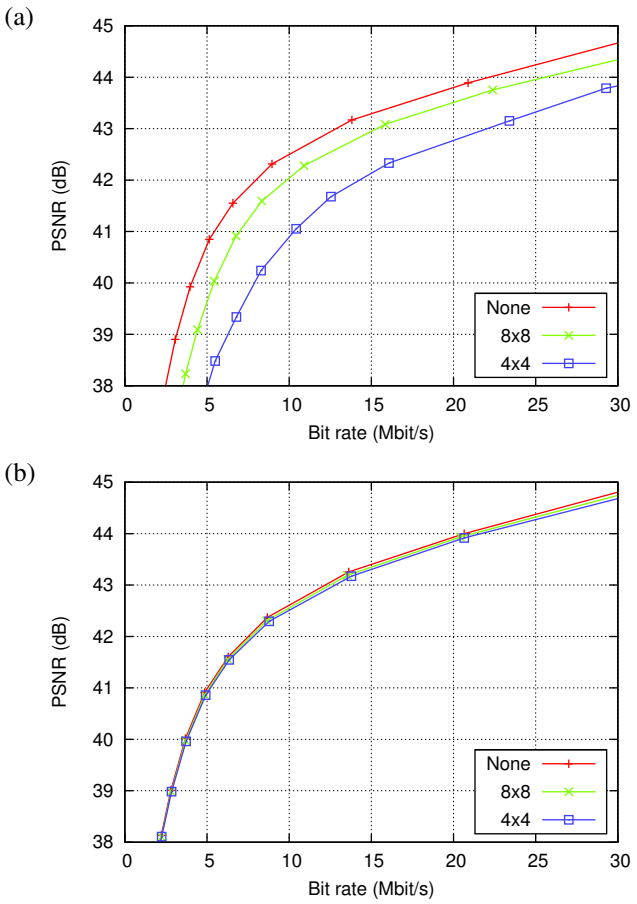

Fig. 4. Graph of PSNR against bit rate for the *pedestrian area* sequence using the hexagonal search algorithm (a) without and (b) with Lagrangian motion vector cost optimization for different sub-partition configurations.

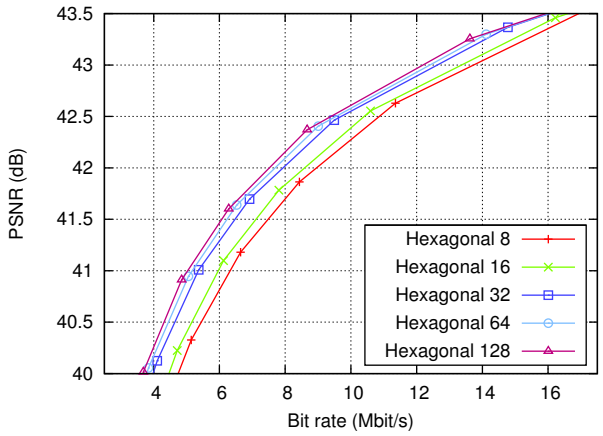

Fig. 5. Graph of PSNR against bit rate for the *pedestrian area* sequence using the hexagonal search algorithm with Lagrangian optimization for different search ranges.

5. ANALYSIS OF MOTION ESTIMATION ALGORITHMS FOR HD VIDEO SEQUENCES

A number of 1920×1080 HD test video sequences from [14] were analysed in different ways.

In H.264, it is possible to sub-partition the 16×16 macroblocks into smaller blocks, and perform a motion estimation search for each sub-partition. An experiment was performed to measure the effect of macroblock sub-partitioning when encoding HD video sequences. The experiment was performed first without motion vector costing, that is, during the motion estimation search, the cost of encoding the motion vector itself was ignored. Fig. 4(a) shows the PSNR against bit rate with no sub-partitions allowed, with sub-partitions of size $\geq 8 \times 8$ allowed, and with sub-partitions of size $\geq 4 \times 4$ allowed. The experiment was repeated with motion vector costing using Lagrangian optimization. Fig. 4(b) shows the PSNR against bit rate with the Lagrangian optimization. In both cases, the *pedestrian area* video sequence and the hexagonal search algorithm were used. Fig. 4(a) indicates that without Lagrangian optimization, partitioning the 16×16 macroblocks into smaller partitions actually gives worse results for HD sequences. For HD sequences objects are usually large in terms of pixels, so while sub-partitioning during the motion estimation search can provide lower SAD costs, there is little to be gained by splitting the macroblock into smaller partitions. There is a cost to be paid however; if for example the macroblock is split into four 8×8 sub-partitions, 4 motion vectors have to be encoded instead of only 1. Fig. 4(b) shows that if Lagrangian optimization is used, the compression performance obtained when using sub-partitioning is much nearer to that obtained when not using sub-partitioning. Unlike in the case of Fig. 4(a), the cost of encoding the motion vectors themselves is taken into consideration, so sub-partitioning does not degrade the coding performance. However, no improvement is obtained for the extra complexity. This experiment thus indicates that for HD sequences, sub-partitioning offers no gain while adding to the coding complexity. The experiment was repeated using different HD sequences with similar results.

Another experiment was performed to see what effect the range of the search has. Fig. 5 shows that when using the hexagonal search algorithm on the *pedestrian area* sequence, increasing the search range from 8 pixels to 16 pixels and then to 32 pixels will improve the coding performance considerably, while increasing the range from 32 pixels to 64 pixels and then to 128 pixels will improve the coding performance less. The improvement is more pronounced when coding for lower quality. Similar results were obtained when using other search algorithms, and when coding different video sequences. Table 2 compares the results obtained when using 5 different motion estimation search algorithms: exhaustive (full) search, UMH search [5], hexagonal search

Table 2. Bit rate and PSNR obtained for the *pedestrian area* sequence using different search algorithms with QP = 26, Lagrangian optimization, and different search ranges.

Search algorithm	Range 8		Range 32		Range 128	
	Bit rate Mbit/s	PSNR dB	Bit rate Mbit/s	PSNR dB	Bit rate Mbit/s	PSNR dB
Exhaustive	8.748	41.9	7.045	41.7	6.151	41.6
UMH	7.946	41.8	6.775	41.7	6.195	41.6
Hexagonal	8.434	41.9	6.914	41.7	6.282	41.6
Diamond	8.657	41.9	7.019	41.7	6.304	41.6
Xilinx	8.746	41.9	7.157	41.8	6.442	41.7

[4], diamond search, and the search algorithm employed by the Xilinx motion estimation engine [9]. In HD sequences, a relatively small movement by an object translates to motion by a large number of pixels because of the high resolution, so performance suffers considerably when the search range is limited. Table 3 compares the results obtained when coding differenct video sequences. Notice the results for the two most difficult sequences to encode, *tractor* and *riverbed*. For *tractor*, increasing the search range from 8 to 32 gives a very large improvement, because it has fast motion, but for *riverbed*, where the difficulty is not due to fast motion, increasing the search range does not help much.

6. CONCLUSION

The paper has presented the LiquidMotion reconfigurable motion estimation processor and the SharpEye integrated development environment for the design of algorithms and for the exploration of the processor's design space. The presented toolset is useful in producing a processor configuration and a block-matching search program efficiently without needing knowledge of the underlying microarchitecture or of the instruction set of the processor. The experiments conducted show that for motion estimation in high definition video, sub-partitioning the macroblocks, while increasing the complexity, does not improve the coding performance. Experimental results also show the effect of the search range on the coding performance for high definition videos. The toolset is available at the download section of `http://sharpeye.borelspace.com/`. The cycle-accurate simulator and full source code are also available.

7. REFERENCES

[1] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: Tools, performance and complexity," *IEEE Circuits Syst. Mag.*, vol. 4, no. 1, pp. 7–28, 2004.

[2] S. Saponara, K. Denolf, G. Lafruit, C. Blanch, and J. Bormans, "Performance and complexity co-evaluation of the advanced video coding standard for cost-effective multimedia communications," *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 1, pp. 220–235, Jan. 2004.

Table 3. Bit rate and PSNR obtained for different video sequences using the hexagonal search algorithm with QP = 26, Lagrangian optimization, and different search ranges.

Video sequence	Range 8		Range 32		Range 128	
	Bit rate Mbit/s	PSNR dB	Bit rate Mbit/s	PSNR dB	Bit rate Mbit/s	PSNR dB
pedestrian	8.434	41.9	6.914	41.7	6.282	41.6
tractor	24.057	39.6	12.339	39.3	12.195	39.3
sunflower	4.559	42.2	3.155	42.3	3.153	42.3
blue sky	5.036	41.0	4.841	41.0	4.839	41.0
riverbed	32.364	39.6	32.003	39.4	31.939	39.3
rush hour	4.763	42.1	4.569	42.1	4.541	42.1
station	2.443	40.7	2.319	40.7	2.319	40.7

[3] S.-C. Cheng and H.-M. Hang, "A comparison of block-matching algorithms mapped to systolic-array implementation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 5, pp. 741–757, Oct. 1997.

[4] A. Hamosfakidis and Y. Paker, "A novel hexagonal search algorithm for fast block matching motion estimation," *Journal on Applied Signal Processing*, vol. 2002, no. 6, pp. 595–600, June 2002.

[5] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified fast motion estimation for JM," *Joint Video Team of ISO/IEC MPEG & ITU-T VCEG, 16th Meeting, Poznan, Poland*, July 2005, JVT-P021.doc.

[6] J. Zhang, X. Yi, N. Ling, and W. Shang, "Bit rate distribution analysis for motion estimation in H.264," *Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers. International Conference on*, pp. 483–484, Jan. 2006.

[7] T. Dias, S. Momcilovic, N. Roma, and L. Sousa, "Adaptive motion estimation processor for autonomous video devices," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, pp. 41–41, Jan. 2007.

[8] K. Babionitakis, G. A. Doumenis, G. Georgakarakos, G. Lentaris, K. Nakos, D. Reisis, I. Sifnaios, and N. Vlassopoulos, "A real-time motion estimation FPGA architecture," *Journal of Real-Time Image Processing*, vol. 3, no. 1–2, pp. 3–20, Mar. 2008.

[9] H.264 motion estimation engine (DO-DI-H264-ME). [Online]. Available: http://www.xilinx.com/products/ipcenter/DO-DI-H264-ME.htm

[10] Tensilica's processor technology. [Online]. Available: http://www.tensilica.com/products/xtensa/index.htm

[11] x264. [Online]. Available: http://www.videolan.org/developers/x264.html

[12] J. L. Nunez-Yanez, E. Hung, and V. A. Chouliaras, "A configurable and programmable motion estimation processor for the H.264 video codec," in *International Conference on Field Programmable Logic and Applications*, Sept. 2008, pp. 149–154.

[13] Reconfigurable motion estimation engine for H.264. [Online]. Available: http://sharpeye.borelspace.com/

[14] Lehrstuhl für Datenverarbeitung, Technische Universität München. Test sequences 1080p. [Online]. Available: ftp://ftp.ldv.e-technik.tu-muenchen.de/dist/test_sequences/1080p/