
Peer reviewed version

Link to published version (if available): 10.1109/FPL.2009.5272247

Link to publication record in Explore Bristol Research

PDF-document

---

**University of Bristol - Explore Bristol Research**

**General rights**

This document is made available in accordance with publisher policies. Please cite only the published version using the reference above. Full terms of use are available: http://www.bristol.ac.uk/pure/user-guides/explore-bristol-research/ebr-terms/
A TOOLSET FOR THE ANALYSIS AND OPTIMIZATION OF MOTION ESTIMATION ALGORITHMS AND PROCESSORS

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 processor designed to execute user-defined block-matching motion estimation algorithms, and a toolset for the design of such algorithms and for the configuration of the processor. The toolset enables the exploration of the processor’s design space in order to find an optimal configuration depending on the target application. The use of the toolset to test different configurations for different kinds of video sequences is illustrated. Experimental results show the benefits and cost of certain optimizations in the motion estimation process, and that fast block-matching search algorithms can outperform full search algorithms commonly used in hardware implementations. The usefulness of the toolset in exploring the configuration space is also shown.

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 XPRESS 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 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 provides experimental results. 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 $112 \times 128$ 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
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 x 20 pixel area that contains the 16 x 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 16-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].

<table>
<thead>
<tr>
<th>Processor impl.</th>
<th>Cycles per MB</th>
<th>FPGA slices</th>
<th>Virtex-II clock</th>
<th>Memory (BRAMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel P4 assembly</td>
<td>~ 3000</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Babionitakis et al. [7]</td>
<td>4532</td>
<td>2052</td>
<td>47 MHz</td>
<td>4 (external reference area)</td>
</tr>
<tr>
<td>Proposed, 1 IPEU</td>
<td>660</td>
<td>2127</td>
<td>50 MHz</td>
<td>11 (1 ref. area, 48 x 48 pixels)</td>
</tr>
<tr>
<td>Proposed, 2 IPEUs</td>
<td>510</td>
<td>1231</td>
<td>125 MHz</td>
<td>21 (2 ref. areas, 112 x 128 pixels)</td>
</tr>
<tr>
<td>Proposed, 3 IPEUs</td>
<td>287</td>
<td>2061</td>
<td>125 MHz</td>
<td>38 (2 ref. areas, 112 x 128 pixels)</td>
</tr>
</tbody>
</table>

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 SharpEye 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.

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 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 for the analysis of 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 ratio (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.

5. ANALYSIS OF MOTION ESTIMATION ALGORITHMS

For analysis, a number of test video sequences from [14] were used. Each sequence has a frame rate of 25 fps. The
The processor supports operating the IPEUs and FPEUs in parallel. In case (a), since no sub-pel estimation was used, this does not affect the results. In case (b), the fps plotted is for the case of using this parallelization. The advantage of parallel operation is that for the same fps rate, less logic cells are required than in the case of sequential operation. For example, if the integer-pel and sub-pel execution units operate sequentially instead of in parallel, we will have to use 3 IPEUs and 2 FPEUs for similar frame rates, further raising the number of required logic cells to 14,600.

Another experiment explored the effect of using more than 1 reference frame and different partition sizes. The experiment was performed on the three 720 × 576 sequences park run, shields and stockholm using 1 IPEU and 1 FPEU. Using 2 reference frames reduced the bit rate by 2.9% for park run, 5.0% for shields, and 3.9% for stockholm, while using 3 reference frames reduced the bit rate by 4.2%, 6.4% and 5.4% respectively. The bit rate is reduced at the cost of a reduction in processing speed. However, since the frame rate is still high enough for real-time processing, no extra execution units are required. Using a minimum partition size of 8 × 8 reduced the bit rate by 1.8% for park run, 4.2% for shields, and 2.1% for stockhom, while a minimum partition size of 4 × 4 reduced the bit rate by 1.9%, 4.4%, and 2.1% respectively. As can be seen, using a minimum partition size of 4 × 4 instead of 8 × 8 has virtually no effect on the bit rate, which means that nothing is gained for the extra complexity.

Fig. 5 shows the effect of different configurations when processing the 1920 × 1080 sequence pedestrian area. The area of the points in the figure is proportional to the number of logic cells required. Some configurations had an fps smaller than 25, which is the minimum for real-time processing, so the number of execution units was increased. This can be seen by their larger area requirements. The point labelled none supports none of the optimizations.

When the multiple motion vector candidate optimization is used (points having c in the label), the bit rate is reduced, and the processing speed changes. When 8 × 8 partition sizes are used (8) with no Lagrangian optimization (points having no l in the label), the bit rate is actually worse than when no sub-block partitions are used, indicating that Lagrangian optimization is essential when using sub-block partitions. Lagrangian optimization has the advantage of both reducing the bit rate and increasing the processing speed in all the cases. The processing speed is increased because the number of search points is reduced owing to faster convergence.

The hexagonal-search algorithm was compared to the full-search algorithm in another experiment. The hexagonal search used consists of up to 8 iterations, with a hexagon radius of 2 pixels; it can select points up to 16 pixels in each direction from the initial point. The full search used can span the same range, 16 pixels in each direction from the
Table 2. Bit rate obtained by using hexagonal and full searches, with and without sub-pel motion estimation.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>No sub-pel</th>
<th>Sub-pel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hex (kbit/s)</td>
<td>Full (kbit/s)</td>
</tr>
<tr>
<td>pedestrian station</td>
<td>4064</td>
<td>4047</td>
</tr>
<tr>
<td>tractor</td>
<td>10187</td>
<td>10140</td>
</tr>
<tr>
<td>park run</td>
<td>10965</td>
<td>10947</td>
</tr>
<tr>
<td>shields</td>
<td>5854</td>
<td>5812</td>
</tr>
<tr>
<td>stockholm</td>
<td>4440</td>
<td>4422</td>
</tr>
</tbody>
</table>

With no sub-pel refinement, the full search produces a marginally better bit rate. With sub-pel refinement, the bit rates are very similar for the 1920 × 1080 sequences, but the hexagonal search performs better in the 720 × 576 sequences by 8% for park run, 18% for shields, and 33% for stockholm. This can be because the hexagonal search is a more logical search which has a higher chance of corresponding to the real object motion in the video sequence, while the full search is much more likely to select a point which does not correspond to the real object motion; the motion vectors given by the full search are more susceptible to noise. This result confirms that a well-designed fast block matching algorithm can provide better rate-distortion performance than the full search algorithm as shown by [6].

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 experiments conducted with the toolset have shown the benefits of the Lagrangian optimization which improves both the obtainable bit rate and the processing speed, as well as the benefits of having multiple motion vector candidates. On the other hand, the use of a 4 × 4 minimum partition size offers virtually no improvement over an 8 × 8 minimum partition size. Experimental data also shows that even simple block-matching search algorithms, such as hexagonal search, can match or improve on the bit rate obtained by full search. The paper demonstrates how the toolset is useful in producing a processor configuration and program efficiently without needing knowledge of the underlying microarchitecture or of the instruction set of the resulting processor. The presented 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