

# H.264/AVC In-Loop De-Blocking Filter Targeting a Dynamically Reconfigurable Instruction Cell Based Architecture

Adam Major, Ioannis Nousias, Sami Khawam, Mark Milward, Ying Yi and Tughrul Arslan  
School of Engineering and Electronics  
University of Edinburgh, Edinburgh, Scotland, EH9 3JL  
Email: adam.major@ed.ac.uk

## ABSTRACT

*We present a new De-Blocking Filter module fully optimised for use on a recently introduced dynamically reconfigurable, instruction cell based architecture. The module consists of a novel combination of standard software transforms alongside architecture specific techniques and aims to reduce reconfiguration overheads and increase utilisation of resources. Our proposed filter outperforms the standard FFMpeg based filter code on the target architecture by 4.5 times.*

## 1. INTRODUCTION

H.264/AVC is the most recent video compression standard recommendation from the Joint Video Team (JVT) of ISO MPEG and ITU-T VCEG [1]. The codec extends the feature sets of existing algorithms, adding several key features such as in-loop de-blocking filter, variable block size quarter pixel precision motion prediction and context adaptive binary arithmetic coding (CABAC). Studies have shown that h.264 can provide significant bit rate reductions over previous standards, achieving 50% improvements over MPEG-2 and 30% improvements over H.263+ and MPEG-4 [2][3]. However, the computational complexity of the algorithm is substantial and as such decode and encode times are significantly elevated [4][5][6].

In [7] the authors presented an FFMpeg based baseline profile h.264 decoder implemented on a recently introduced ANSI-C programmable, dynamically reconfigurable architecture [8]. Initial results were promising at three times the speed of ARM9 and a quarter of the speed of Xilinx Virtex II. In order to improve future performance and approach FPGA-like speeds, a complexity analysis was performed to highlight the most time consuming functions which should be targeted for further and more extensive optimisation. Foremost amongst these was the adaptive de-blocking filter which was found to account for around 36% of the total decode time.

The h.264 adaptive de-blocking filter is used to reduce artefacts which occur as a side effect of the block based transform. The strength of the filter is modified based on

the local image characteristics in order to preserve real edges. Hence image sharpness can be maintained whilst still effectively removing artefacts from flat regions where tiling effects are more pronounced [9]. The filter is termed an in-loop filter since motion compensated macro-blocks are reconstructed based on filtered pixel data which improves the quality of the motion compensation. Filtered sequences can achieve bit rate improvements of up to 10% over non-filtered sequences with similar subjective image quality [10].

However, the filtering algorithm is highly complex and has been shown to account for 33% of the total decode time in a study of the baseline profile decoder [10]. This is due to two main factors: the highly adaptive nature of the algorithm and the vast amount of pixel data which must be read from memory. The high density of branches and unpredictability of the filter in particular, present a significant challenge to the target reconfigurable fabric as they lead to low utilisation of resources and increased reconfigurable overhead.

We present a new h.264 in-loop de-blocking filter module fully optimised for use on a dynamically reconfigurable instruction cell based architecture. A novel combination of existing software techniques and architecture specific constructs are employed in order to increase the determinism of the algorithm, and hence both increase resource utilisation and reduce reconfiguration overheads. The throughput of the module is increased further through the deployment of the advanced features of the core such as pipelining, SIMD instructions and packed memory access.

## 2. OVERVIEW OF TARGET ARCHITECTURE

The target architecture consists of a coarse grained reconfigurable fabric of interconnected instruction cells (ICs), with functionality derived from common machine code operations such as ADD, MUL and SHIFT. The array can be dynamically reconfigured to provide highly parallel FPGA-like representations of software. Control is through a series of VLIW instructions which are extracted from a C based high level algorithm by a sophisticated scheduling algorithm [11]. This enables the architecture to take advantage of vastly increased instruction level parallelism compared with existing processors, since it is

not constrained by dependencies between operations, or by small branches. As a result it exhibits a significant performance and flexibility advantage over traditional solutions. For further information refer to [8].

### 3. ADAPTIVE FILTER ALGORITHM

The de-blocking filter is applied to the edges of each of the 4x4 sub-blocks in the current Macro Block (MB) using the order shown in figure 1. First the vertical luma edges are processed followed by the horizontal luma edges. The two chroma channels, Cb and Cr, then follow using the same order. [9]

Depending on the coding type of the 4x4 blocks and their position within the array, a boundary strength  $B_s$  is assigned to each edge. This determines the strength of the filtering to be applied, ranging from  $B_s=4$ , strongest filtering, to  $B_s=0$ , no filtering. The strongest filters  $B_s=4$  and  $B_s=3$  are used only when the 4x4 block is Intra coded and  $B_s=4$  only when the edge is also a MB boundary. For Inter coded blocks,  $B_s=2$  is assigned when a quantised residue exists in the 4x4 block on either side of the boundary and  $B_s=1$  is used when the blocks have different motion compensation reference frames or vectors. Otherwise  $B_s$  is set to 0. This takes account of the fact that motion compensated blocks are based on data which has already been filtered and hence require less filtering.

Figure 2 shows the standard naming convention for pixels across each line in a 4x4 block edge. The filter is turned on or off for each line based the values of  $p_1$ ,  $p_0$ ,  $q_0$ ,  $q_1$  and two thresholds, termed alpha and beta, which have values dependent upon the quantisation parameter (QP) of the current frame. Filtering is only applied if all of the following conditions are true:

$$\begin{aligned} B_s & \neq 0 \\ ABS(p_1 - p_0) & < \beta \\ ABS(p_0 - q_0) & < \alpha \\ ABS(q_0 - q_1) & < \beta \end{aligned} \quad (1)$$

This enables filtering for  $p_0$  and  $q_0$  only. The filter is extended to  $p_1$  and  $q_1$  respectively if the further conditions are also true:

$$\begin{aligned} ABS(p_2 - p_0) & < \beta \\ ABS(q_2 - q_0) & < \beta \end{aligned} \quad (2)$$

For  $B_s=4$ , the filter operation is different. If (1) is true a moderate strength filter is applied to  $p_0$  and  $p_1$  only. If either of (2) are true AND condition (3) is also true, the pixels  $\{p_2, p_1\}$  or  $\{q_1, q_2\}$ , respectively, are also filtered and a stronger filter is applied to  $p_0$  and  $q_0$ .

$$ABS(p_0 - q_0) < ((\alpha \gg 2) + 2) \quad (3)$$

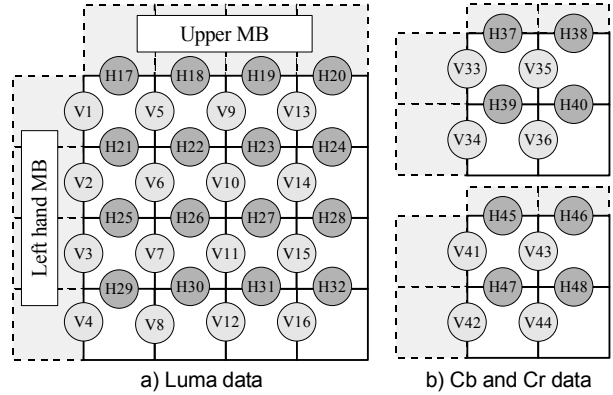


Fig. 1. Processing order of 4x4 sub-block edges.

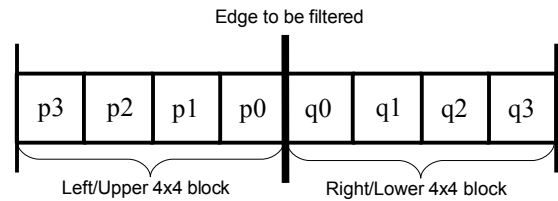


Fig. 2. Boundary Pixel naming convention.

This analysis assesses the likelihood of an edge in the image being natural, or the result of the block based transform. The dependence of the filter thresholds, alpha and beta, on the QP, enables this process to be sensitive to the likely level of artefacts in a given frame. At high QP values the image will be extremely lossy and therefore it is more likely that an edge will be the result of quantisation. However, at low QP levels the reconstruction will be close to the original and artefacts will be less common.

### 4. COMPLEXITY

There are two major contributors to the level of complexity in the h.264 de-blocking filter. The first is the highly adaptive nature of the algorithm, which must modify the strength of the filter based on a number of thresholds, the encoded type of the current macro block, and the underlying pixel values themselves. This results in a very high density of conditional branches right down to the inner loop and hence it becomes very difficult to identify and schedule parallel instructions [9]. The implications this has for the target architecture are significant since it relies heavily on the ability to exploit instruction level parallelism.

The non-deterministic nature of the filter presents a further problem for a reconfigurable fabric. With a deterministic section of code it is possible to preemptively fetch a future array configuration from memory whilst the current configuration is executing. This effectively masks the reconfiguration time. However, when a branch exists, the destination of the program remains unknown until the condition has been evaluated. In this case the fetching of

the next array configuration must wait and therefore, less (or perhaps none) of the reconfiguration time will be hidden. Thus the high density of branches in the filter algorithm results in significant reconfiguration overhead.

The second major bottleneck in the filtering algorithm is the substantial memory bandwidth which is required. Due to the fine grained 4x4 pixel transform, almost every pixel in the current macro block must be accessed to perform the filtering [9]. This is exacerbated by the fact that the filter order specified in the standard makes it difficult to reuse previously loaded pixels and therefore nearly every pixel must be read from memory four times. [12][13]

## 5. PROPOSED FILTER

The aim of the proposed filter module is to minimise reconfiguration overheads and maximise resource utilisation by increasing the determinism of the filter algorithm. The module makes use of a number of different transforms to eliminate branches. Further performance gains are achieved through the deployment of advanced processing modes, such as pipelining and SIMD.

### 5.1. Hardware Multiplexer

Many of the branch operations in the inner loop can be attributed to the calculation of absolute differences and clipping functions. On the target architecture it is more efficient to implement these small branches as hardware multiplexers. Both outcomes are computed along with the condition, and the desired result is directed to the output. Using this technique it is possible to process several conditional statements in parallel and make full use of the available resources. Figure 3 shows how the multiplexer can be used to remove a jump from an example block of code. The conditional operator  $\{x = (\text{true}) ? (a) : (b)\}$  is used to identify the multiplexer operation to the compiler.

It can be seen that with the original code, the operations are split over two steps and hence the core must be reconfigured after the jump. Since the result of the jump will not be known until the end of the step, it is not possible to mask the reconfiguration time. With the tailored code, the array does not need to be reconfigured at all and hence suffers no reconfiguration overhead. Moreover, the ADD and SUB operations now occur in parallel with the conditional logic and therefore the entire function is evaluated in the same time as the first example requires for the jump logic alone. The small amount of energy wasted in discarded results is more than made up for by the substantial increase in performance due to increased parallelism and reduced reconfigurable overheads.

### 5.2. Loop Re-Distribution

Whilst it is possible to remove a great deal of branches using the hardware multiplexer, it is not recommended to

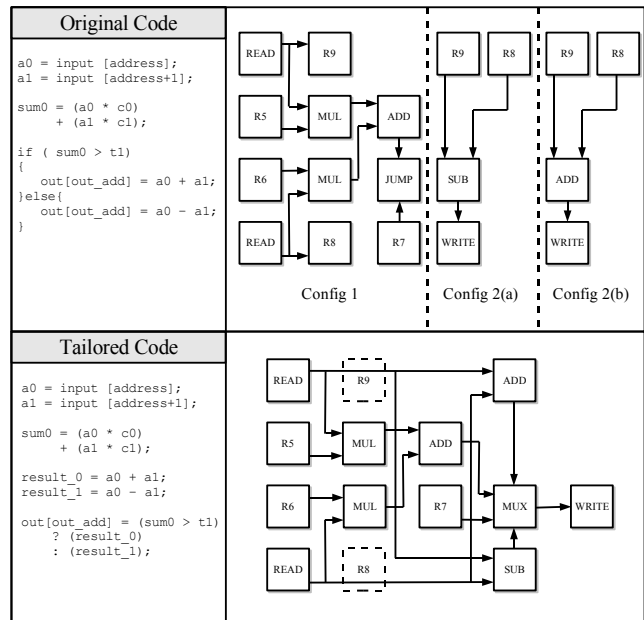


Fig. 3. Example use of multiplexer to remove a jump and reduce reconfiguration overhead.

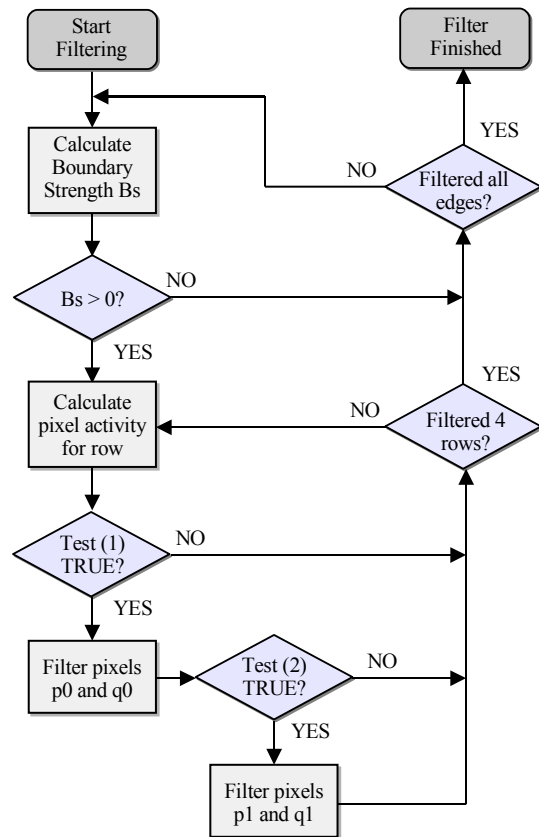


Fig. 4. Standard luma filter order contains many jumps and results in high reconfiguration overheads.

use the technique where a branch is large and heavily unbalanced. The filter boundary strength calculation and pixel level adaptivity are prime examples of this – see figure 4. Depending on the result of each analysis, the filter may be either be on or off; either a lot of work must be done, or none at all. It would be extremely inefficient, and provide negligible speed increases, to always perform the filtering. In these cases, alternative methods must be used to eliminate the jump.

The proposed filter achieves this by re-distributing the filter loop into three distinct phases shown in figure 5. Firstly, the boundary strength calculation is performed for all edges in the macro block. Secondly the conditions in (1), (2) and (3) are calculated for all necessary edges. Finally filtering is applied, as required, to the entire Macro Block.

Each analysis stage stores its results in local registers ready for the next phase to access. In order for the filter to be able to skip lines or edges which do not meet the filter conditions, without having to perform a jump, the results are stored as sets of address vectors. Instead of checking a filter condition flag for each line or edge, each section determines the address of the next compliant line or edge from the stored vector.

This scheme breaks the heavily branched filter algorithm into three deterministic loops containing no branches. Not only does this vastly reduce the reconfiguration overhead since future steps can be pre-fetched, it increases the utilisation of resources allowing more instructions to be evaluated in parallel. Moreover, the two analysis phases can be mapped to a single array configuration each. Thus the array maintains its configuration for the duration of their runtime and the reconfiguration time is therefore zero.

### 5.3. Packed Pixel-Memory Access

Image data in the h.264 decoder is stored in arrays of 8-bit chars and therefore all pixel reads operate on 8-bits at a time. Since the target architecture can retrieve 32-bits of data per memory interface channel, this is a significant waste of bandwidth. In order to improve this, we assign a 32-bit int pointer to the 8-bit char pixel array. This enables the proposed filter module to access four packed pixels in a single 32-bit read operation and hence reduces memory access by a factor of four.

Memory access is further reduced by making use of fast on chip register files. Each 32-bit block is loaded from the main pixel memory only once and is then placed in a register file. Thereafter, all access to the pixel data is made through the register files. Once the filtering is complete the data is paged back to main memory.

### 5.4. SIMD Operations

The target reconfigurable fabric has the ability to perform SIMD instructions with support for up to 4 8-bit data paths per functional unit. Since most of the operations

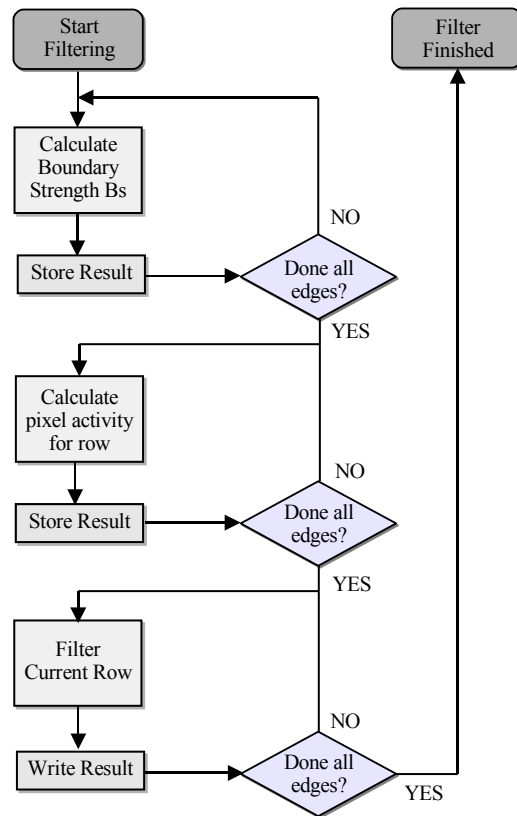


Fig. 5. Proposed three stage filter order increases determinism and reduces reconfiguration overheads.

in the filter are 8- or 16-bit, a great deal of extra bandwidth can be harnessed using SIMD. The analysis logic for example only requires 8-bit precision so it is possible to perform 4 times as many operations with the same resources. The filtering itself requires 16-bit precision so SIMD can provide twice the throughput.

### 5.5. Pipelining

The abundant register resources and FPGA like nature of the target reconfigurable array, provide the ability to implement pipelining within suitable loops. Such loops must consist of a single, self referencing configuration, with no internal branches, and must complete sufficient iterations to counteract the induced pipeline latency.

Pipelining is applied at the machine code level. Register operations are manually inserted at appropriate locations in the target loop in order to reduce the critical path, thus allowing the step to run at a far higher frequency.

Due to the loop transforms performed on the main filter loop there are two loops to which pipelining can be applied. These are the Bs calculation loop and the loop calculating the conditions (1), (2) and (3). We apply a three stage pipeline to both loops.

## 6. RESULTS

The performance of both the standard and optimised filters was profiled in a test decode of a number of 200 frame, baseline profile video sequences. Table 1 details the results for the average core utilisation and configuration load times for each code version. The profiling shows that the transforms and modifications applied to the standard code have increased the average core utilisation by around 3.6 times. Furthermore they have reduced the proportion the execution time spent loading the next array configuration by over 60%.

**Table 1.** Average core utilisation and load times of filter function based array configurations

Code	Average Core Utilisation	Reconfigurable overhead (load time)
Standard	15.00%	48.00%
Optimised	55.00%	17.00%

Executing the standard FFMpeg filter functions, the reconfigurable array achieves an average throughput of 18,063 filtered MBs per second. Our proposed filtering module significantly improves upon this figure and allows the processor to attain an average of 81,405 filtered MBs per second. The increased parallelism and determinism of the new filtering module therefore provide a performance increase of 4.5 times.

## 7. CONCLUSION

We have developed a novel loop filter module specifically tailored for a dynamically reconfigurable, instruction cell based architecture. Several loop transforms have been combined with the fabric's hardware multiplexers to increase the determinism of the algorithm and hence minimise reconfiguration overheads and increase instruction level parallelism. Further performance has been attained through the employment of advanced features of the core, such as SIMD instructions, packed memory access and pipelined loops. The proposed loop filter module achieves a 4.5 times speed increase over the original FFMpeg code. Future work will target the remaining key modules of the h.264 decoder algorithm such as motion compensation and inverse discrete cosine transform.

## 8. REFERENCES

- [1] Joint Video Team of ITU-T and ISO/IEC JTC 1, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. h.264 | ISO/IEC 14496-10 AVC)," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT G050, March 2003.
- [2] Raja, G.; Mirza, M.J.; "Performance comparison of advanced video coding H.264 standard with baseline H.263 and H.263+ standards" Communications and Information Technology, 2004. ISCTIT 2004. IEEE International Symposium on, Volume 2, 26-29 Oct. 2004 Page(s):743 - 746 vol.2
- [3] Wiegand, T.; Sullivan, G.J.; Bjøntegaard, G.; Luthra, A.; "Overview of the H.264/AVC video coding standard", Circuits and Systems for Video Technology, IEEE Transactions on, Volume 13, Issue 7, July 2003 Page(s):560 – 576
- [4] A. Chang, O.C. Au, Y.M. Yeung: "A novel approach to fast multi-block motion estimation for H.264 video coding" Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on, Volume 1, 6-9 July 2003 Page(s):I - 105-8 vol.1
- [5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wedi, "Video coding with H.264/AVC: Tools, Performance, and Complexity", IEEE Circuits and Systems Magazine, First Quarter 2004, pp. 7-28.
- [6] J. Zhang, Y. He, S. Yang, Y. Zhong: "Performance and complexity joint optimization for H.264 video coding", Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on, Volume 2, 25-28 May 2003 Page(s):II-888 - II-891 vol.2
- [7] A. Major, I. Nousias, S. Khawam, M. Milward, Y. Yi and T. Arslan: "H.264 decoder implementation on a dynamically reconfigurable instruction cell based architecture", Proceedings of the 2006 System On Chip Conference, Sept 24th-27th 2006, pp. 49-52.
- [8] Reconfigurable Instruction Cell Array, U.K. Patent Application Number 0508589.9.
- [9] P. List, A. Joch, J. Lainema, G. Bjøntegaard, M. Karczewicz: "Adaptive Deblocking Filter", Circuits and Systems for Video Technology, IEEE Transactions on, Volume. 13, no. 7, July 2003. pp. 614-619
- [10] M. Horowitz, A. Joch, F. Kossentini, and A.Hallapuro, "H.264/AVC baseline profile decoder complexity analysis", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, 715-727, 2003.
- [11] Y. Yi, I. Nousias, M. Milward, S. Khawam, T. Arslan and I. Lindsay: "System-level scheduling on instruction cell based reconfigurable systems", Presented at Design, Automation and Test in Europe Conference 2006 (DATE 2006), 6-10 March, ICM, MESSE Munich, Germany
- [12] Chao-Chung Cheng, Tian-Sheuan Chang, Member, IEEE, and Kun-Bin Lee: "An In-Place Architecture for The Deblocking Filter in H.264/AVC", Circuits and Systems II: Express Briefs, IEEE Transactions on, July 2006 Page(s):530 - 534
- [13] D. Wu, B. Sheng and W. Goo: "An implemented architecture of deblocking filter for H.264/AVC", Image Processing, International Conference on (ICIP 2004), IEEE 2004, pp.665-668