

# Efficient Processing of Color Image Sequences Using a Color-Aware Instruction Set on Mobile Systems

Jongmyon Kim and D. Scott Wills

*Microelectronics Research Center  
School of Electrical and Computer Engineering  
Georgia Institute of Technology, Atlanta, Georgia 30332-0250  
{jmkim and scott.wills}@ece.gatech.edu*

## **Abstract**

*This paper investigates the use of both the luminance and chrominance components in color image and video processing algorithms and proposes a color-aware instruction set that improves the performance of such multimedia applications. The proposed color instruction set, quantized color pack extension (QCPX), which applied to a 32-bit datapath processor, supports parallel operations on two-packed 16-bit YCbCr (6:5:5) color pixels, resulting in more efficient processing of color pixel data as well as greater concurrency for color multimedia workloads on mobile systems. Experimental results on a set of media benchmark applications indicate that the QCPX-optimized version achieves a speedup ranging from about three to seven times while reducing energy consumption from 69% to 85% over the baseline version on identically-configured, dynamically-scheduled ILP superscalar processors. The QCPX-optimized version also outperforms the MDMX-like (MIPS's multimedia extension) version.*

## **1. Introduction**

As multimedia is rapidly revolutionizing our society, its applications, including color image and video processing, are becoming one of the dominant computing workloads [1]. However, their sophisticated encoding and decoding cost at higher resolutions and frame rates prevents their use in most real-time applications. In response to the growing application demands for such multimedia processing, manufacturers of general-purpose processors have included multimedia extensions (e.g., Intel's MMX<sup>TM</sup> [2], Sun's VIS [3], MIPS's MDMX [4]) that solve the mismatch between wide data paths and the relatively short data types (e.g., eight-bit pixels) found in multimedia applications and execute several parallel operations on these sub-elements, resulting in the performance increase of multimedia workloads. However, data representation mismatches between the instruction set extension and the applications result in frequent packing/unpacking and alignment of operand data. For example, most multimedia extensions store pixel information as a packed 32-bit word composed of an eight-bit red (R), green (G), blue (B), and alpha (A) field (a band-interleaved format). While subword parallelism can be exploited on the R, G, and B components, it cannot be exploited on the alpha value [5] unless the data are rearranged into a band-separated format.

In addition to the huge increase in performance through SIMD multimedia extensions, many researchers have developed fast video coding algorithms [6]-[11] that run in real-time on embedded systems. Some of these coding algorithms, which discard the chrominance (color) components, take into account only the luminance pixels to reduce computational complexity while sacrificing accuracy in terms of video quality [8]-[11]. In general, the use of only the luminance component in color image and video processing algorithms (e.g., motion estimation and edge detection) provides sufficient information for the operations [12]. However, color image and video applications are performed on the three-

dimensional vectors (e.g., YCbCr) simultaneously or separately. Incorporation of chrominance channels must improve the accuracy of video processing algorithms because of the additional information that each channel provides. In [12], [13], it was found that performing motion estimation using both the luminance and chrominance components yielded savings in bit rate as well as a high degree of accuracy in motion estimates over the luminance-only motion estimation while adding some computational complexity to the algorithm. Thus, the trade-off between improvements in compression and computational complexity has been case dependent. Since we desire a high degree of accuracy in color image sequences, we use both the luminance and chrominance channels for executing such applications.

In this paper, we investigate the use of the color components in video coding algorithms and propose a color-aware instruction set for ILP processors, quantized color-pack extension (QCPX) that improves the performance of color multimedia applications through an efficient color space representation in the YCbCr color space in addition to subword parallelism. Since the YCbCr color space allows coding schemes to exploit the properties of human vision by truncating some of least significant bits for every color pixel as well as by allocating fewer bits to the high-frequency chrominance components that are perceptually less significant, a 16-bit YCbCr (6:5:5) format (a six-bit luminance and two five-bit chrominance components) provides satisfactory image quality, as shown in Figure 1. Note that the color version of this paper can be obtained from [14]. Thus, QCPX provides not only greater subword parallelism by packing more of these smaller color pixel values into a word while processing these pixel values in parallel but also a solution of format conversion problems inherent to RGBA-based extensions.

This paper is organized as follows. Section 2 discusses advantages of using the chrominance components in video coding algorithms. Section 3 presents a summary of QCPX along with pictorial examples. Section 4 describes our workloads, the architecture modeled, and the simulation methodology. Section 5 presents experimental results and analyzes the results, and Section 6 concludes the paper.



**Figure 1. Original images with converted output images [14] using a 16-bit YCbCr (6:5:5) format.**

## 2. Advantages of using color information in video coding algorithms

### 2.1. Simultaneous motion estimation of all color components

The most important step in estimating the quality of color video frames is that of motion estimation [15], one of the most computational intensive tasks by today's compression standards. To reduce the computational complexity in conventional motion estimation, the standard block matching algorithm (BMA) for motion estimation is used only on the luminance or intensity component of the video signal. In general, the use of only the luminance component in estimating the motion field of a color sequence provides sufficient information for the operations [12]. However, for color video frames that have low luminance or detailed color information, the chrominance components may be required for accurate motion estimation. The use of the chrominance components is investigated in this paper. In particular, our full-search BMA (FSBMA) uses both luminance and chrominance components, finding one motion vector for all components. The matching criterion of the proposed BMA for motion estimation is defined as

$$\begin{aligned}
 MAD(m, n) = & \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |y(i+m, j+n) - x(i, j)|_Y \\
 & + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |y(i+m, j+n) - x(i, j)|_{Cb} \\
 & + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |y(i+m, j+n) - x(i, j)|_{Cr}
 \end{aligned} \tag{1}$$

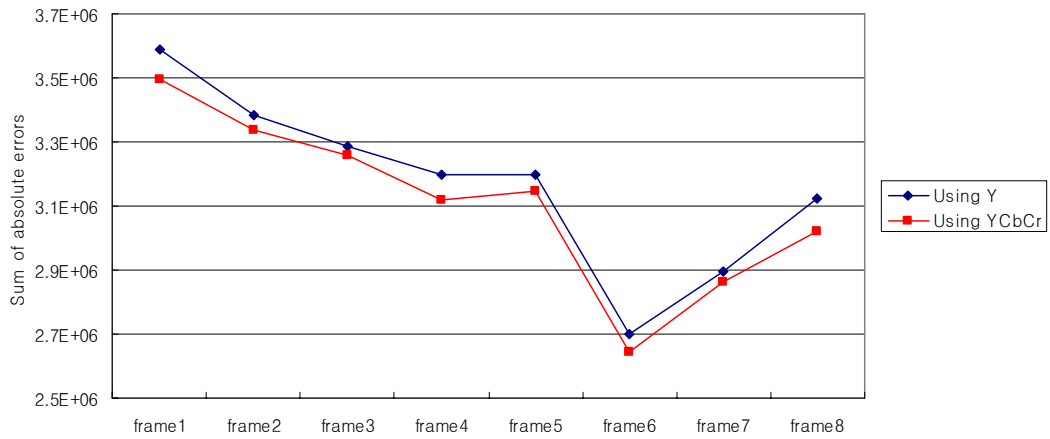
$$-p \leq m, n \leq p \tag{2}$$

$$\mathbf{v} = \arg \min_{-p \leq m, n \leq p} MAD(m, n), \tag{3}$$

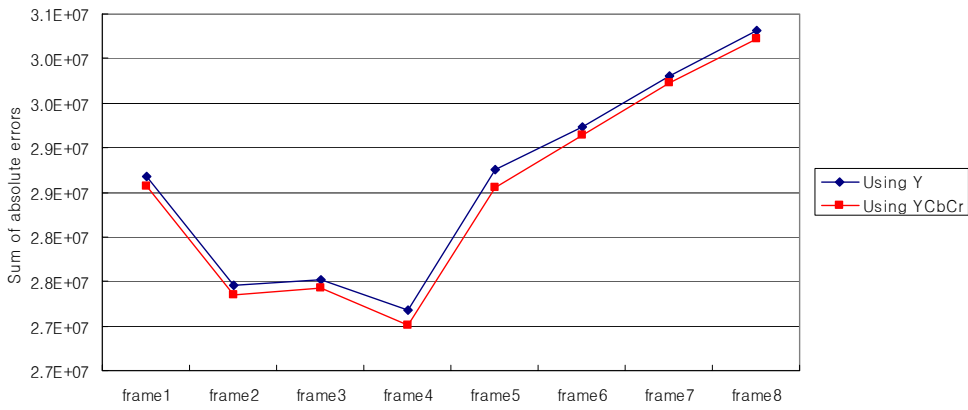
where  $x(i, j)$  is the reference block of size  $N \times N$  pixels at coordinates  $(i, j)$ ,  $y(i+m, j+n)$  is the candidate block within a search area in the previous frame,  $(m, n)$  represents the candidate displacement vector, and  $\mathbf{v}$  is the motion vector.

Two different versions of motion estimation, one using only the luminance (Y) component and the other using both the luminance (Y) and chrominance (Cb and Cr) components, were executed in MATLAB [16] for three different color videos, each of which contains nine frames. Each frame consisted of three-band Quad-CIF resolution (176×144) pixels. In this experiment, the 16×16 pixels for the macroblock size and -/+16 for the search range were used.

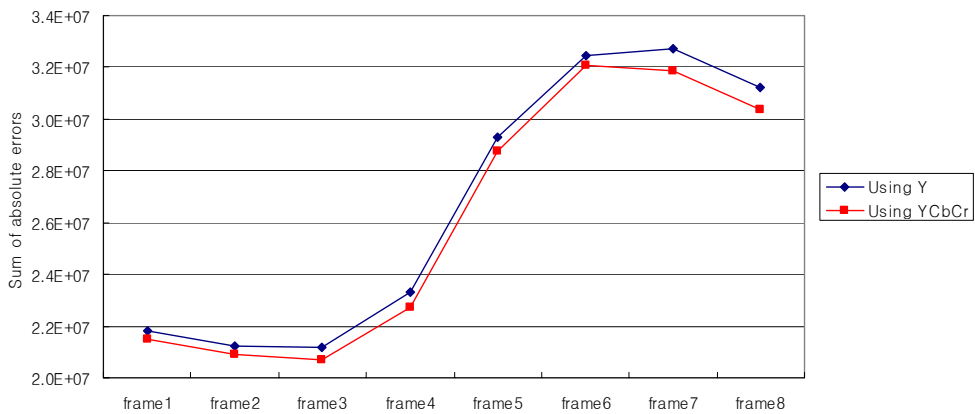
Figures 2, 3, and 4 show a comparison of the sum of absolute errors between using only the luminance component and both the luminance and chrominance components in motion estimation. We observe that the FSBMA using both the luminance and chrominance components performs better than luminance-only FSBMA in the sum of absolute errors for all the simulations in terms of the accuracy of motion estimation. Note that frame 1 estimated from frame 0, frame 2 estimated from frame 1, and so on. However, this comes with approximately 41% more computations.



**Figure 2. Sum of absolute errors for the foreman video sequence.**



**Figure 3. Sum of absolute errors for the mobile calendar video sequence.**



**Figure 4. Sum of absolute errors for the football video sequence.**

## 2.2. Vector median filter in the YCbCr color space

Vector median filters (VMFs) are widely used in color image and video because of their ability to reduce impulse noise while preserving edge capabilities [17]. Unlike typical VMFs based on the RGB color space, we implemented a VMF using all color components in the YCbCr color space and then compared with a VMF using only the luminance component. The VMF orders vectors according to their relative magnitude differences in the window, and the vector median of given  $N$  sample vectors  $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_N\}$ , is  $\mathbf{p}_{vm}$  such that

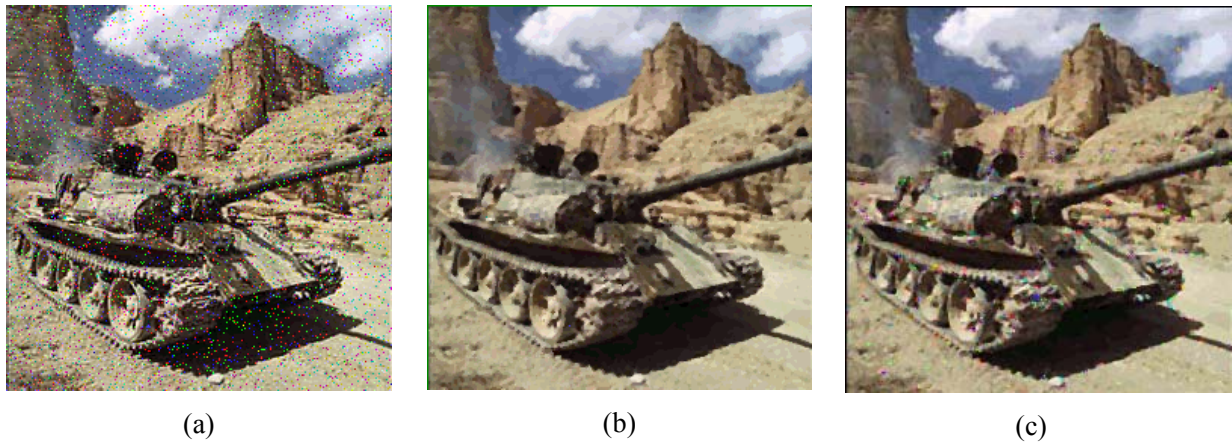
$$\mathbf{p}_{vm} \in \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_N\} \quad (4)$$

and for all  $j = 1, \dots, N$ ,

$$\sum_{i=1}^N \|\mathbf{p}_{vm} - \mathbf{p}_i\|_1 \leq \sum_{i=1}^N \|\mathbf{p}_j - \mathbf{p}_i\|_1, \quad (5)$$

where  $N$  is the number of pixels in the window and  $\|\cdot\|_1$  denotes the L-1 norm.

In this paper,  $N$  is equal to nine ( $3 \times 3$ ), each containing a Y, Cb, and Cr component in a color image. Figure 5 shows a corrupted tank image by 5% (of each R, G, and B component) ‘salt and pepper’ noise with recovered output images using a VMF of both the luminance and chrominance components as well as only the luminance component. The experimental results indicate that the VMF using both the luminance and chrominance components efficiently reduces impulse noise, while luminance-only VMF could not reduce impulse noise accurately because the chrominance channels also contain noise factors. However, its complicated coding algorithm in which median filtering is simultaneously performed on the three-dimensional vectors (e.g., YCbCr) contributes to the computational burden.

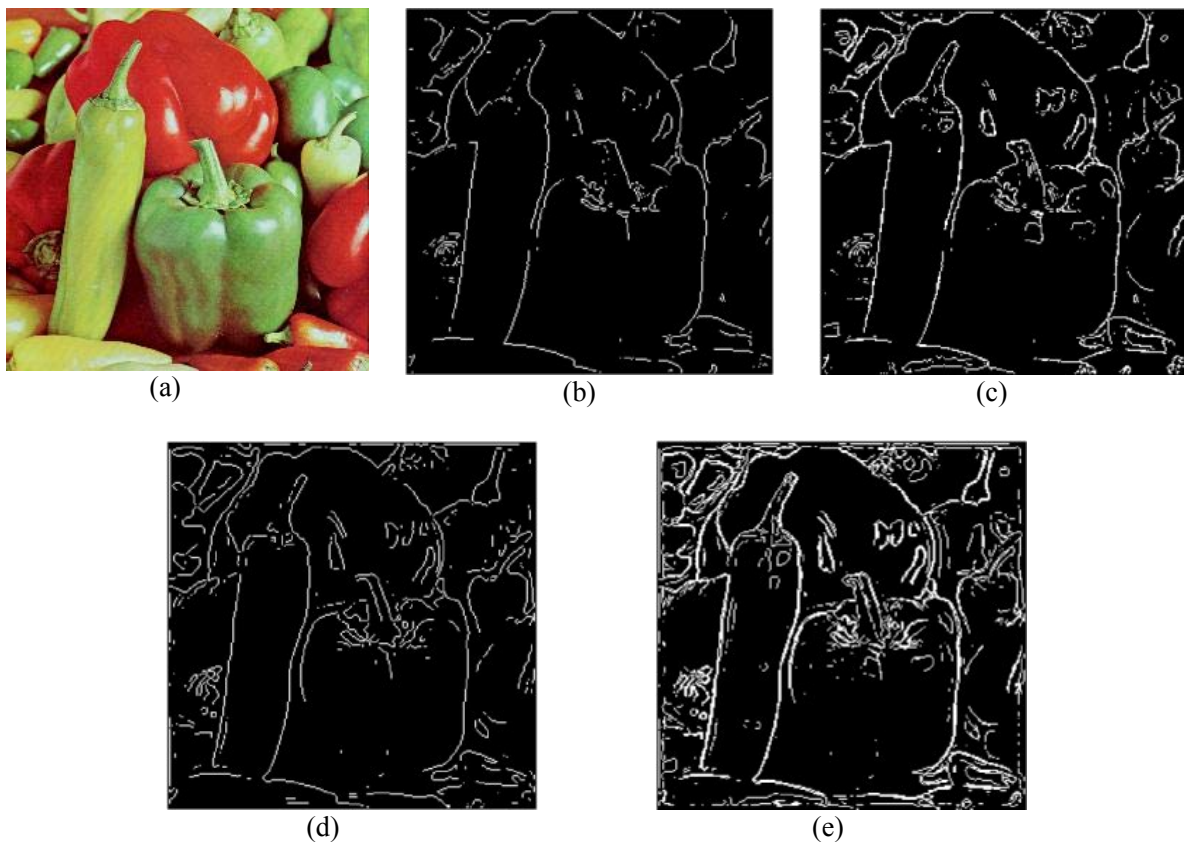


**Figure 5. A corrupted image with recovered output images using a VMF [14]: (a) corrupted by 5%, (b) using both luminance and chrominance components, and (c) using only the luminance component.**

## 2.3. Color edge detection of both luminance and chrominance components

Edge detection, an effective technique in image and video analysis, such as segmentation, registration, and identification of objects in a scene, is an image-to-image operation in which each output pixel is determined by multiplying the input pixel and its surrounding neighbor pixels with a filter operator (a mask) in a  $3 \times 3$  convolution window and summing the products. In a monochrome image, an edge is defined as an intensity discontinuity. However, monochrome edge detection may not correspond to the set of edges existing in a color image that has low luminance or detailed color information.

In [18], [19], several different approaches have been tested to use color components for edge detection purposes. The most straightforward approach is to apply monochrome edge detection to three color channels independently. The edge results of the three channels are then combined by using a certain logical operator (fused by means of an *or* logical operator in this paper) in order to obtain a more complete edge information. Figure 6 shows a comparison for edge detection of using both the luminance and chrominance components as well as using only the chrominance component. The results indicate that the use of both the luminance and chrominance components in the edge detection extracts better edge information than luminance-only edge detection while adding some computational complexity to the algorithm. To accelerate such computational color coding algorithms, we propose a new color-aware instruction set, QCPX, for mobile systems. QCPX, along with pictorial examples, is presented in the following section.

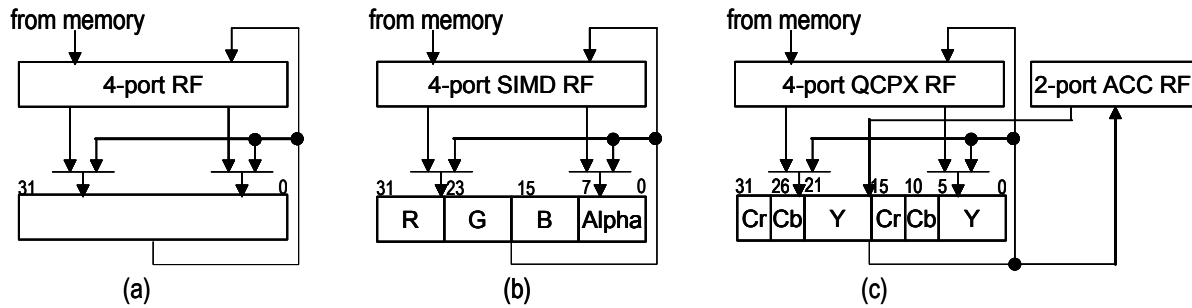


**Figure 6. An original image with recovered output images using edge detection techniques [14]: (a) an original peppers image, (b) using a Sobel operator for only the luminance component, (c) using a Sobel operator for both luminance and chrominance components, (d) using a Laplace operator for only the luminance component, and (e) using a Laplace operator for both luminance and chrominance components. .**

### 3. A color-aware instruction set

The proposed color-aware instruction set, QCPX, is targeted at accelerating color image and video processing applications. The instructions, applied to a 32-bit datapath processor, operate on two-packed, quantized 16-bit YCbCr data in parallel. A 16-bit color pixel representation includes a six-bit luminance (Y) and two five-bit chrominance (Cr and Cb) components while providing satisfactory image quality.

Note that this pixel-truncation technique differs from similar techniques used in image and video compression applications (e.g., JPEG and MPEG) in that it reduces spatial redundancy in individual pixel word sizes rather than in each dimension through subsampling. In addition, QCPX includes a 128-bit color-packed accumulator that provide a solution to overflow and other issues caused by packing data as tightly as possible by implicit width promotion and enough extra space. Figure 7 illustrates a normal 32-bit operation, a  $4 \times 8$ -bit SIMD operation used in many general-purpose processors, and a  $2 \times 16$ -bit QCPX operation employing heterogeneous (non-uniform) subword parallelism.



**Figure 7. Type of operations: (a) a baseline 32-bit operation, (b) a 32-bit SIMD operation, and (c) a 32-bit QCPX operation.**

Typical subword-parallel ISA extensions store a true color pixel as a packed 32-bit word containing an eight-bit red, green, blue, and alpha field (an interleave format). While subword parallelism can be exploited on the RGB components, it cannot be exploited on the alpha value unless the data are rearranged into a band-separated format. However, QCPX could eliminate the RGB limitations by storing two-packed 16-bit YCbCr color pixels in a 32-bit wide datapath of the processor while performing these data operations in parallel in a single cycle with a small increase in functional unit complexity. Thus, QCPX exploits parallelism within the color space representation in addition to subword parallelism. The QCPX instruction set is classified in three different groups: (1) parallel ALU (Arithmetic Logical Unit) instructions, (2) parallel compare instructions, and (3) special-purpose instructions.

### 3.1. Parallel arithmetic and logical instructions

Parallel arithmetic and logical instructions include ADD\_CRCBY (signed, unsigned saturation, and modulo), SUBTRACT\_CRCBY (signed, unsigned saturation, and modulo), MULTIPLY\_CRCBY, BCAST\_CRCBY (broadcast), AVERAGE\_CRCBY, DIVIDE\_CRCBY, SHIFT\_CRCBY [left|right], ROTATE\_CRCBY, and MIX\_CRCBY [left|right] instructions, the most frequent operations in many color image and video computations. For example, the ROTATE\_CRCBY instruction rotates the source YCbCr quantities in the packed data-type destination, while the MIX\_CRCBY instruction mixes every other YCbCr quantity of a packed data-type source register with the corresponding YCbCr quantity from the second source register. Figure 8-(a) and (b) show a rotate instruction and a mix instruction, respectively.

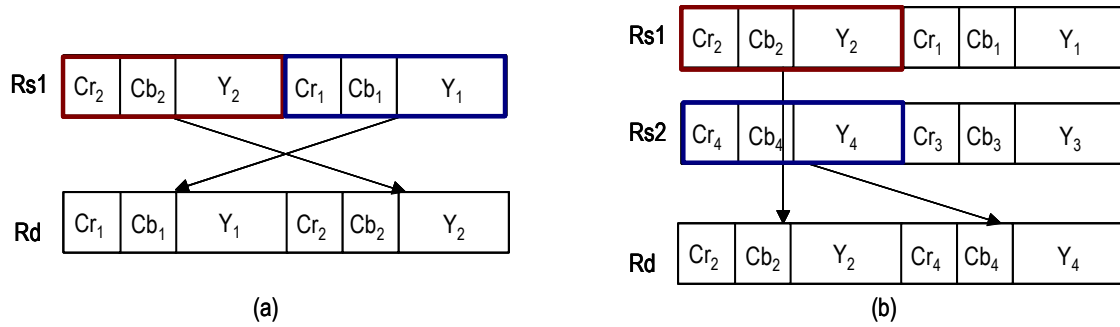


Figure 8. (a) A rotate instruction. (b) A mix instruction.

### 3.2. Parallel compare instructions

Parallel compare instructions compare pairs of subword elements (e.g., Y, Cb, and Cr) in the two source registers. Depending on the instructions, the results are varied for each subword comparison. For example, a CMPEQ\_CRCBY instruction compares pairs of two-packed YCbCr color pixels in the two 32-bit source registers, generating all 1's or 0's of each Y, Cb, and Cr sub-element for a 32-bit wide element mask. Parallel compare instructions are useful for image enhancement (e.g., scalar median filter), edge detection, and 3-D rendering algorithms that determine which objects are in front of others. These include CMPEQ\_CRCBY, CMPGT\_CRCBY, CMPLT\_CRCBY, MIN\_CRCBY, and MAX\_CRCBY. The first three instructions are performed on two-packed YCbCr color pixels concurrently to produce masks that can be used to reduce branches. However, MIN\_CRCBY and MAX\_CRCBY instructions output the minimum or maximum values of the corresponding elements in the two source registers, respectively. Figure 9-(a) and (b) show a MIN\_CRCBY instruction and a MAX\_CRCBY instruction, respectively.

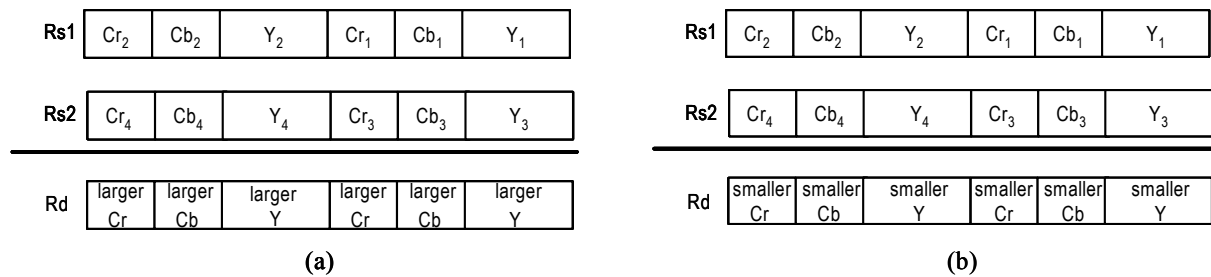
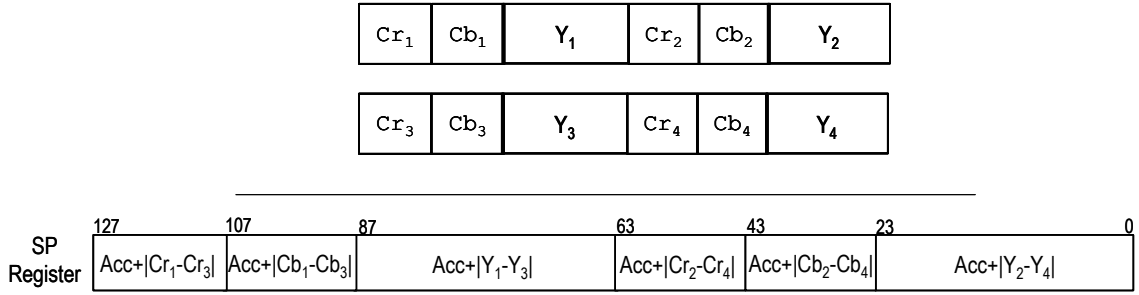


Figure 9. (a) A maximum instruction. (b) A minimum instruction.

### 3.3. Special-purpose instructions

Special-purpose QCPX instructions include ZACC (zero-accumulate), MACC\_CRCBY (multiply-accumulation), and ADACC\_CRCBY (absolute-distance-accumulation), which provide the most computational benefits of all the QCPX instructions. Moreover, these special-purpose instructions use a 128-bit color-packed accumulator that provides a solution of overflow and other issues caused by packing data as tightly as possible. For example, an absolute-distance-accumulate instruction is primarily targeted at accelerating motion estimation in which the ADACC\_CRCBY instruction sums the absolute differences between the respectively packed YCbCr elements while accumulating its calculation results in a packed accumulator register, as shown in Figure 10. Each of the ADACC\_CRCBY instruction calls saves its sum in a partitioned 128-bit special-purpose register until the ZACC (zero-accumulate) instruction is called. These special-purpose instructions are commonly used in color image and video compression algorithms such as vector quantization, JPEG, H.26x, and MPEG-1/2.



**Figure 10. An absolute-distance-accumulation instruction.**

## 4. Methodology

### 4.1. Color Image and Video Workloads

To cover a full range of color image and video processing workloads, a media benchmark suit was selected and coded. It includes four applications for color sequences: edge detection (EDGE), scalar median filter (SMF), vector median filter (VMF), and vector quantization (VQ), and two important video kernels, the discrete cosine transform (DCT) and motion estimation (ME) within the MPEG standard. These widely used applications and kernels, briefly described in Table 1, are important components of current and future real-world workloads. All benchmarks were run with Quad-CIF resolution (176x144) 3-band input images.

**Table 1. Summary of benchmark kernels and applications.**

Kernels	Description
ME	Removes temporal redundancies between video frames in MPEG/H.26L video algorithms.
DCT	Converts a signal into elementary frequency components to exploit spatial redundancy inherent in image and video data.
<b>Applications</b>	
EDGE	Performs feature extraction and segmentation of binary color sequences by using a simple 3x3 convolution mask (Laplacian operator).
SMF	Removes binary noise by using a 3x3 filter performing separately on the three vectors for color video.
VMF	Removes or suppresses noise by using a 3x3 filter performing simultaneously on the three vectors for color video
VQ	Compresses and quantizes collections of input data by mapping k-dimensional vectors on vector space $R^k$ into a finite set of vectors.

### 4.1. Processor configuration and tools

A modified version of the SimpleScalar toolset [20], an infrastructure for out-of-order superscalar modeling, was used to profile instruction execution for the baseline, MDMX-like and QCPX-optimized versions. In addition, an architectural-level power simulator, Watch [21], was used to estimate the energy consumption of each benchmark program. Figure 11 shows a methodology framework that evaluates the

performance and energy consumption of three different versions of the programs on a dynamically-scheduled superscalar ILP processor architecture.

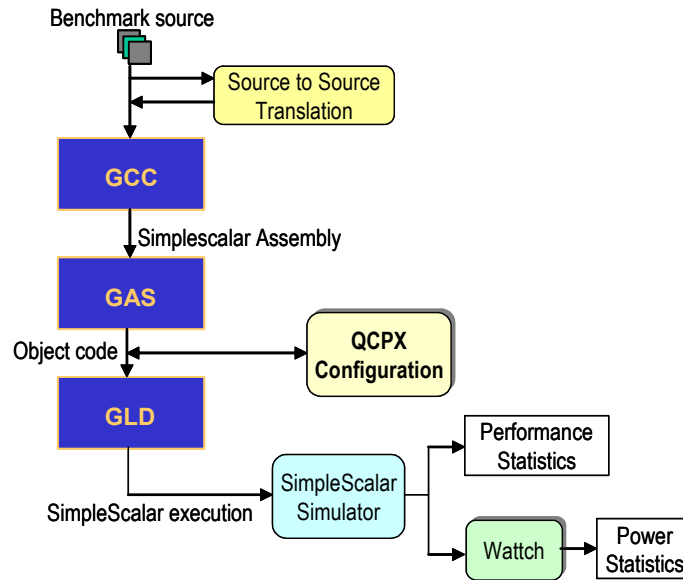


Figure 11. A methodology framework for dynamically-scheduled simulation.

Table 2. Baseline configuration of simulated processor.

Parameter	Value
Fetch queue size, Decode width, Issue width, and Commit width	4 instructions/cycle
RUU (window) size	16 instructions
LSQ (Load Store Queue)	8 instructions
Fus	ALU:4, MULT/DIV:1, FP-ALU:2, FP-MULT:1
Branch Predictor	Combined predictor (1K entries) of bimodal predictor (4K entries) table and 2-level predictor (2-bit counters and 10-bit global history)
L1 D-cache	128-set, 4-way, 32-byte line, LRU, 1-cycle hit, total of 16KB
L1 I-cache	512-set, direct-mapped 32-byte line, LRU, 1-cycle hit, total of 16KB
L2 unified cache	1024-set, 4-way, 64-byte line, LRU, 6-cycle hit, total of 256KB
Memory latency (memory width)	50 cycles for first chunk, 2 thereafter (64 bits)
Instruction TLB	16-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty
Data TLB	32-way, 4096 byte page, 4-way, LRU, 30 cycle miss penalty
<b>QCPX FUs</b>	<b>ALU: 4, MULT: 1</b>

The MDMX-like and QCPX-optimized versions of the benchmark applications were created by replacing fragments of the baseline assembly language with ones containing QCPX or MDMX-like instructions. The baseline, MDMX-like, and QCPX-optimized versions of the benchmarks have the same parameters, data sets, and calling sequences. Since the target platform is an embedded system, operating system interface code (e.g., file system access) is not included in this study. Additional optimizations

typical of embedded compilers were applied to the baseline version before the MDMX-like and QCPX-optimized versions were created. The simulations were executed on the same technology and processor configuration, as summarized in Table 2.

## 5. Experimental Results

In the experiment, three different versions of the benchmark programs: (1) baseline ISA without subword parallelism, (2) baseline plus MDMX-like ISA, and (3) baseline plus QCPX ISA, were executed on a modified version of the SimpleScalar simulator. Their profile statistics obtained from the SimpleScalar simulator were then applied to the Wattch power simulator which evaluates each benchmark's energy consumption. In this study, we assume a 0.18 micron process technology at 600MHz, and aggressive, non-ideal conditional clocking. (Power is scaled linearly with port or unit usage, and unused units still dissipate 10% of the maximum power rather than drawing zero power.) The execution cycle count and corresponding energy consumption of each case form the basis of the study comparison. Table 3 summarizes the performance and energy consumption of each benchmark version.

**Table 3. A comparison of the performance and energy consumption of the baseline, MDMX-like, and QCPX-optimized versions.**

Benchmark	Version	# of executed instructions	# of cycles	Total energy (mJ)
EDGE	Baseline	29,039,536	15,731,814	168
	MDMX-like	11,372,209	4,892,165	57
	QCPX	5,852,647	2,567,534	30
SMF	Baseline	74,510,711	33,135,080	415
	MDMX-like	47,950,961	21,530,736	256
	QCPX	23,978,441	10,872,693	128
VMF	Baseline	215,405,293	116,221,809	1,250
	MDMX-like	105,317,925	46,186,466	605
	QCPX	53,011,538	23,153,363	304
VQ	Baseline	1,051,030,648	501,593,681	5,648
	MDMX-like	345,102,076	145,615,328	1,721
	QCPX	172,555,372	72,829,736	860
DCT	Baseline	186,179,946	96,321,954	1,266
	MDMX-like	64,054,777	27,667,930	404
	QCPX	32,056,351	13,828,616	203
ME	Baseline	4,337,061,667	2,271,018,469	29,137
	MDMX-like	1,318,302,336	623,593,148	8,260
	QCPX	719,067,804	340,170,262	4,504

### 5.1. Performance evaluation results

Figure 12 illustrates the execution performance (speedup in executed cycles) of the baseline, MDMX-like, and QCPX-optimized versions over the baseline version without subword parallelism. The use of QCPX enables a significant reduction in the executed instruction counts (i.e., ALUs, branches, and cache accesses) for all the benchmarks because of six-way subword parallelism in a 32-bit datapath width. The results indicate that the QCPX-optimized version provides a speedup ranging from 3x (SMF) to 6.9x (VQ) over the baseline version while achieving a speedup ranging from about 1.83x (ME) to 2x (VQ and DCT) over the MDMX-like version.

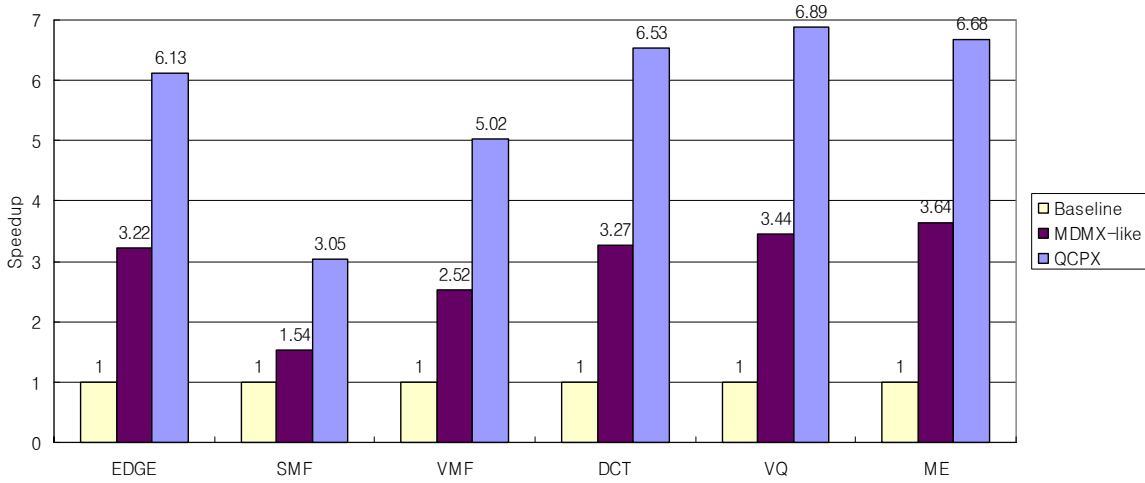


Figure 12. Speedups over the baseline version.

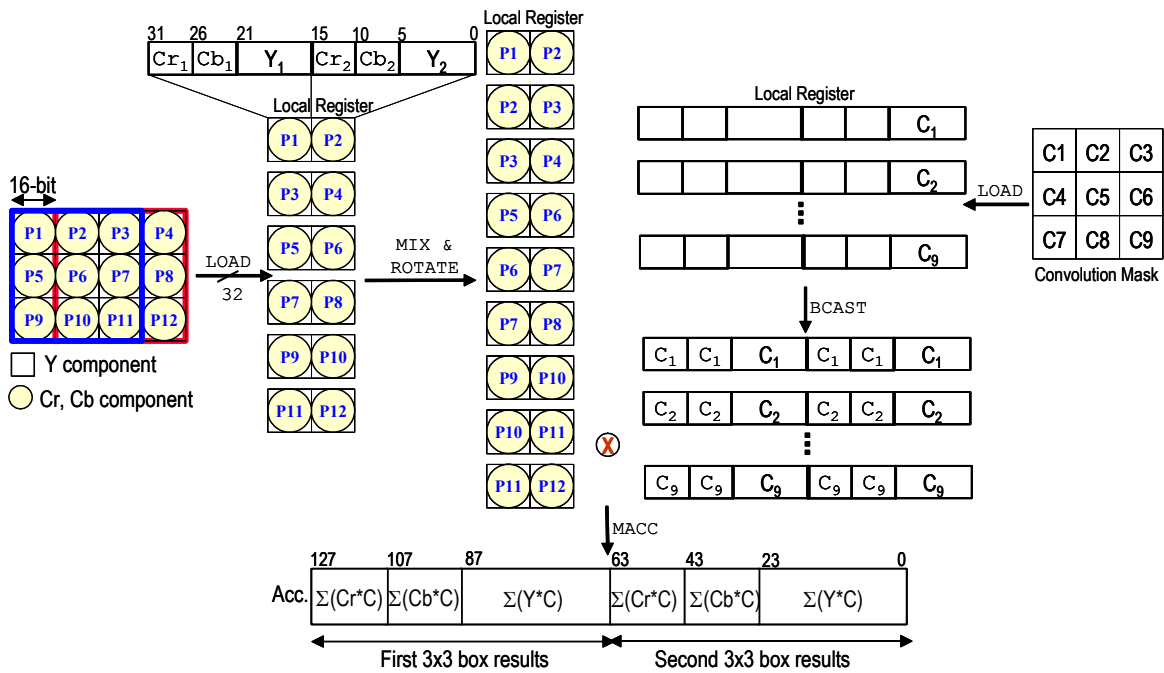


Figure 13. The procedure for edge detection implementation with QCPX instructions.

**5.1.1. Edge detection:** We implemented an edge detection application based on a Laplacian operator [22], taking into account local changes in both the luminance and chrominance components. The most time critical operation in this algorithm is the multiply-accumulation between pixels and corresponding coefficient values in the  $3 \times 3$  window. Figure 13 illustrates the procedure for EDGE implementation with the QCPX instructions. The BCAST\_CRCBY instruction broadcasts each of coefficient values saved in memory to the Y, Cb, and Cr positions of the registers; the ROTATE\_CRCBY and MIX\_CRCBY instructions rearrange the YCbCr data of the registers as an efficient format for calculation. The MACC\_CRCBY instruction then multiplies pairs of elements in the two source registers (one for pixels and

the other for coefficient values) while accumulating its calculation results in a packed accumulator register. Furthermore, two window boxes are efficiently processed in parallel, resulting in a large reduction in executed instructions and memory accesses for intermediate results (since immediate results are stored in the accumulator register rather than in memory). The QCPX-optimized version of EDGE achieves an overall speedup of 6.13x over the baseline version and 1.91x over the MDMX-like version.

**5.1.2. Vector median filter:** A VMF implementation in this paper employs the YCbCr color space rather than the RGB. Its construction was based on a L1 norm to order vectors according to their relative magnitude difference in a window.

The most time critical operation for this implementation is the pixel-distance computation, which computes the absolute magnitude difference between pixels in a  $3 \times 3$  window and accumulates it. With the `ADACC_CRCBY` instruction, a VMF operation can be performed on the two windows of pixels in parallel. In particular, `ADACC_CRCBY` compares six pairs of registers for equality while summing their absolute magnitude differences in a packed accumulator register, resulting in a significant reduction of logic overhead and memory access instructions. The QCPX-optimized version achieves an overall speedup of 5x over the baseline version and 2x over the MDMX-like version.

**5.1.3. Scalar Median Filter:** Like the VMF, scalar median filter (SMF) is a noise reduction technique that eliminates impulse noise spikes from an image by taking the median pixel value in a  $3 \times 3$  window that is stepped across the whole image. A drawback of this application is that it separately replaces the corrupted color values (e.g. Y, Cb, and Cr) with corresponding neighborhood color values, producing uncorrelated pixels [23].

Its most compute-intensive step is to find the median pixel value from nine pixels enclosed within the window. This application is implemented efficiently by using the `MAX_CRCBY` and `MIN_CRCBY` instructions, which compare pairs of sub-elements in the two source registers, outputting the minimum or maximum values of the corresponding elements. The results indicate that the QCPX-optimized version achieves a speedup of 3x and 1.98x versus the baseline version and the MDMX-like version, respectively.

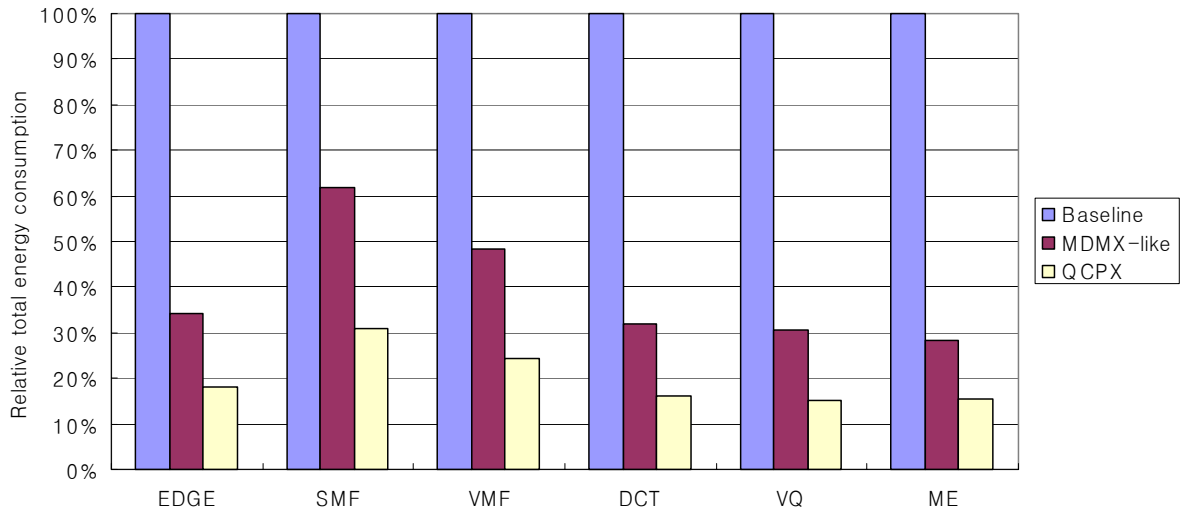
**5.1.4. Vector quantization:** Vector quantization (VQ) [24], an attractive technique for image and video compression, is defined as a mapping of k-dimensional vectors on vector space  $R^k$  into a finite set of vectors  $Y = \{y_i, i=1, \dots, N\}$ , where N is size of the codebook. Each vector  $y_i=(y_0, \dots, y_{k-1})$  is called as a code vector or codeword. Only index i of the resulting code vector is sent to the decoder. At the decoder, identical copy of the codebook is retrieved as the encoder by a simple table-lookup operation. The compression ratio depends on the cardinality of the codebook, usually much smaller than that of the input domain.

In this implementation, a codebook of 256  $4 \times 4$  code vectors was used for achieving a 0.5 bit per pixel encoding of 16-bit color images. The `ADACC_CRCBY` instruction efficiently replaces the most time critical operation of the distortion calculation between a  $4 \times 4$  input block and a local codeword in the block matching algorithm (BMA). The QCPX-optimized version shows an overall speedup of 6.9x over the baseline version and about 2x over the MDMX-like version.

**5.1.5. The discrete cosine transformation:** The discrete cosine transformation (DCT) is one of the most computationally intensive kernels in image and video compression such as JPEG, H.263, and MPEG-1/2 encoding. A 2D-DCT has been implemented on the SimpleScalar simulator using the row-column method in which a one-dimensional DCT is applied to the rows and then columns.

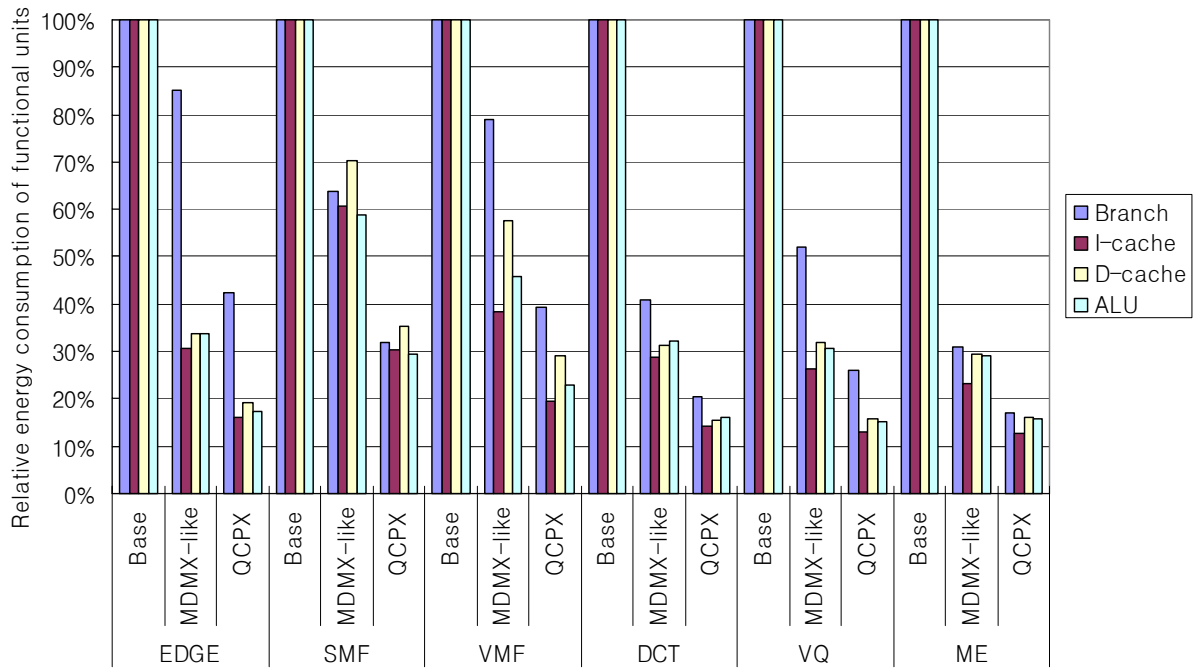
For this implementation, cosine terms were used as coefficient and preloaded into the system, a fixed-point arithmetic was used to perform the integer calculation, and the 16-bit YCbCr (6:5:5) format without down-sampling (e.g., 4:2:2 or 4:2:0) was used. In the DCT, matrix multiplication is the most time consuming process. Using the `MACC.C16_CRCBY` instruction, matrix multiplication is achieved very efficiently. `MACC.C16_CRCBY` allows multiplications between the upper 16-bit YCbCr components and the upper 16-bit coefficient as well as between the lower 16-bit YCbCr components and the lower 16-bit





**Figure 15. Relative total energy consumption over the baseline version.**

Since the ALUs, branches, and cache accesses are reduced significantly through QCPX, less energy is spent on speculative execution and caches accesses. Figure 16 shows a relative energy consumption of the MDMX-like and QCPX-optimized versions in each functional unit over the baseline version.



**Figure 16. Relative energy consumption of functional units over the baseline version.**

## 6. Conclusions

In this paper, we have investigated the use of color information in video coding algorithms and have examined the impact of the QCPX instruction set for color image and video processing applications on dynamically-scheduled ILP superscalar processors. The use of all color components in such algorithms increased the accuracy of the results while adding some computational complexity. However, QCPX have provided greater performance and parallelism through implicit support for color pixel processing in addition to subword parallelism while solving the problems inherent in the RGBA-based extensions. The experimental results have shown that the QCPX-optimized version achieves a speedup ranging from three to seven times and reduces energy consumption from 69% to 85% over the baseline version across a set of color image and video benchmarks. The QCPX-optimized version also outperformed the MDMX-like version in a same machine platform. These results demonstrate that QCPX is an effective approach to support color image and video workloads on mobile systems.

## 7. References

- [1] K. Diefendorff and R. Dubey, "How multimedia workloads will change processor design," *IEEE Computer*, vol. 30, no. 9, pp. 43-45, Sept. 1997.
- [2] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol.16, no. 4, pp. 42-50, Aug. 1996.
- [3] M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing," *IEEE Micro*, vol. 16, no. 4, pp. 10-20, Aug. 1996.
- [4] *MIPS extension for digital media with 3D*, Technical Report <http://www.mips.com>, MIPS technologies, Inc., 1997.
- [5] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PCs," in *Proc. Communications of the ACM*, vol. 40, no. 1, pp. 25-38, Jan. 1997.
- [6] C. Cheung and L. Po, "A novel small-cross-diamond search algorithm for fast video coding and videoconferencing applications," *Proc. IEEE Intl. Conf. on Image Processing*, vol. 1, pp. 681-684, Sept. 2002.
- [7] W. Pan, "A fast 2-D DCT algorithm via distributed arithmetic optimization," in *Proc. IEEE Intl. Conf. on Image Processing*, pp. 114-117, Sept. 2000.
- [8] V. G. Moshnyaga, "A new computationally adaptive formulation of block-matching motion estimation," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 11, no. 1, pp. 118-124, Jan. 2001.
- [9] F. Moschetti, M. Kunt, and E. Debes, "A statistical adaptive block-matching motion estimation," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 13, no. 4, pp. 417-431, May 2003.
- [10] K. M. Nam, J. S. Kim, R. H. Park, and Y. S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 5, no. 4, pp. 344-351, Aug. 1995.
- [11] S. C. Cheng and H. M. Hang, "A comparison of block matching algorithms mapped to systolic-array implementation," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 7, no. 5, pp. 741-757, Oct. 1997.
- [12] S. C. Kwatra, C. M. Lin, and W. A. Whyte, "An adaptive algorithm for motion compensated color image coding," *IEEE Trans. Commun.*, vol. COM-35, pp. 747-754, July 1987.
- [13] N. R. Shah and A. Zakhor, "Resolution enhancement of color video sequences," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 8, no. 6, pp. 879-885, June 1999.
- [14] The related materials of this paper: <http://www.ece.gatech.edu/jmkim/ASAP04/>
- [15] B. C. Tom and A. K. Katsaggelos, "Resolution enhancement of monochrome and color video using motion compensation," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 10, no. 2, pp. 278-287, Feb. 2001.
- [16] MATLAB Homepage: <http://www.mathworks.com/>
- [17] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filters," *Proc. of the IEEE*, vol. 78, no. 4, pp. 678-689, April 1990.

- [18] A. Koschan, "A comparative study on color edge detection," *Proc. of 2<sup>nd</sup> Asian Conference on Computer Vision*, vol. III, pp. 574-578, Dec. 1995.
- [19] K. N. Plataniotis and A. N. Venetsanopoulos, *Color Image Processing and Applications*, Springer, 2000.
- [20] T. Austin and D. Burger, "The SimpleScalar Tool Set, Version 2.0," TR-1342, Computer Sciences department, University of Wisconsin, Madison.
- [21] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. Intl. Symposium on Computer Architecture*, pp. 83-94, June 2000.
- [22] B. Kamgar-Parsi and A. Rosenfeld, "Optimally isotropic Laplacian operator," *IEEE Trans. on Image Processing*, vol. 8, no. 10, pp. 1467-1472, Oct. 1999.
- [23] M. Barni, V. Cappellini, and A. Mecocci "Fast vector median filter based on Euclidean norm approximation," *IEEE Signal Processing Letter*, vol. 1, no. 6, pp. 92-94, June 1994.
- [24] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Press, 1992.
- [25] *Coding of Moving Pictures and Audio*, ISO/IEC JTC1/SC29/WG11 N3312, 2000.
- [26] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," in *Proc. Symp. Low Power Electron.*, pp. 38-39, Oct. 1994.