

# Combining the Quantized Color Instruction Set and Loop Unrolling on Portable Video Processing 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

As wireless video products evolve, they demand more sophisticated processing at higher resolutions and frame rates. Computational performance and energy efficiency have become critical design issues. This paper presents the Quantized Color Pack eXtension (QCPX) combined with a loop unrolling (LU) technique to improve execution performance and energy efficiency of color image and video processing applications. QCPX applied to a 32-bit datapath processor supports parallel operations on two packed 16-bit YCbCr (Y: luminance, Cr and Cb: chrominance) color pixels, providing greater subword-level parallelism by increasing the number of smaller color pixels packed into a word. Instruction-level parallelism can be further enhanced through loop unrolling. These techniques provide greater performance and efficiency for multimedia workloads on mobile systems. Experimental results on a set of media benchmark applications indicate that the LU plus QCPX-optimized version achieves a speedup ranging from 3.8 to 7.9 while reducing the energy consumption from 76% to 87% over the baseline version on identically configured, dynamically scheduled ILP superscalar processors. The LU plus QCPX-optimized version also outperforms the LU plus MDMX-like (MIPS's multimedia extension) version.

**Keywords:** media processing, multimedia extension, loop unrolling, subword parallelism, mobile systems

## 1. INTRODUCTION

With the rapid development of handheld devices and wireless communication networks, consumer demand for video-over-wireless capability on mobile computing systems is growing rapidly [1]. However, future multimedia applications including color image and video processing will demand tremendous computational and I/O throughput. Since mobile color image and video products are battery powered, both energy efficiency and high performance are critical design considerations for portable computing systems.

In response to the growing application demand for multimedia processing, manufacturers of general-purpose processors have included multimedia extensions (or subword-parallel instructions) to achieve greater performance on multimedia workloads. Examples of multimedia extensions include Intel's MMX<sup>TM</sup> [2], SSE<sup>TM</sup> and SSE-2 [3], Hewlett Packard's MAX2 for the PA-RISC architecture [4], Sun Microsystems' VIS for SPARC [5], MIPS's MDMX [6], Alpha's MVI [7], and Motorola's ALTIVEC for PowerPC<sup>TM</sup> architecture [8]. These multimedia extensions consist of single instruction and multiple data (SIMD) instructions that operate concurrently on data that are packed in a single register or memory location. However, data representation mismatches between the instruction set extension and the applications result in frequent packing/unpacking and alignment of operand data. Most multimedia extensions store pixel information as a packed 32-bit word composed of an 8-bit red (R), green (G), blue (B), and alpha (A) field (band-interleaved format). While subword parallelism can be exploited on the R, G, and B components, it cannot be exploited on the alpha value [9] unless the data are rearranged into a band-separated format.

Color image and video processing applications are performed on the three-dimensional vectors (e.g., Y, Cb, and Cr) simultaneously rather than performed separately, adding some computational complexity to the algorithms [10-13]. Assuming that the luminance component provides sufficient information for the operations, some researchers take into account only the luminance component in such algorithms to reduce computational complexity, sacrificing accuracy of the algorithms [14-16]. 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 and video applications, we use both luminance and chrominance channels for executing such applications.

Loops typically contain the most critical code segments for color image and video processing programs. Loop unrolling [17, 18] is an optimizing technique that reorganizes and reschedules the loop body. It decreases execution time by reducing loop overhead and exposing ILP for machines with multiple functional units within the loops. However, it also expands the code size by duplicating the loop body. This decreases instruction cache performance, especially for embedded processors that have limited caches.

In this paper, we present the QCPX instruction set architecture (ISA) combined with a loop unrolling technique for portable computing systems to improve the performance and energy efficiency of color multimedia applications. QCPX applied to a 32-bit datapath processor supports parallel operations on two packed, quantized 16-bit YCbCr color pixels, enabling greater concurrency for color image and video applications and solving format conversion problems inherent to RGBA-based extensions. The luminance-chrominance space (e.g., YCbCr) allows coding schemes to exploit the properties of human vision by allocating significantly less bits to the high-frequency chrominance components that are perceptually less significant. This allows the chrominance space to be implemented using shorter fields (e.g., Cb and Cr fields are 4 bits rather than 8) without perceivable distortion of color. Thus, QCPX provides greater subword parallelism by increasing the number of color pixels packed into a word. Loop unrolling applied to QCPX provides increased performance. The resulting increase in the code size is compensated by replacing assembly language fragments of isomorphic statements [19] within a basic block with ones containing QCPX instructions.

The QCPX-optimized benchmark programs are simulated on a modified SimpleScalar toolset [20] to evaluate the performance. The architectural power simulator Wattch [21] is used to determine each benchmark's energy consumption. The benchmark suite includes pre/post-processing algorithms of digital video (vector median filter, scalar median filter, edge detection, and vector quantization) and two computational video kernels, discrete cosine transform (DCT) and motion estimation (ME), within the MPEG standards. Our experiment results on these benchmarks indicate that the LU plus QCPX-optimized version achieves a speedup ranging from 3.8 to 7.9 and reduces corresponding energy consumption from 76% to 87% over the baseline version due to a significant reduction in the number of instructions executed, cache accesses, and branch frequencies. The LU plus QCPX version outperforms the LU plus MDMX-like version (MIPS's Extension for Digital Media [6]) on the same technology and processor configuration.

This paper is organized as follows. In Section 2, an experimental methodology for this study is provided, along with a summary of QCPX plus LU for color multimedia applications. Section 3 presents analysis of the results of the performance and energy efficiency for three different versions of the benchmark programs: (1) baseline ISA, (2) baseline plus MDMX-like ISA, and (3) baseline plus QCPX ISA. Additional simulations and analysis of the results with and without LU are provided. Finally, conclusions are presented in Section 4.

## 2. METHODOLOGY

### 2.1. Multimedia workloads

To cover a full range of color image and video processing workloads, a media benchmark suit was selected and coded. It includes four pre/post-processing applications of video (vector median filter: VMF, scalar median filter: SMF, edge detection: EDGE, and vector quantization: VQ) and computational video kernels (e.g., DCT and ME) within the MPEG standard. These widely used applications and kernels are important components of current and future real-world workloads and are briefly described in Table 1. All benchmarks were run with Quad-CIF resolution (176x144) 3-band (i.e., channel) input images. As mentioned in Section 1, both the luminance and chrominance components are used for executing these algorithms to obtain a high resolution video as well as a higher degree of accuracy in the applications.

Table 1. Summary of the multimedia benchmarks used in this study.

Kernels	Description
ME	Motion estimation routine on a frame of 176x144 3-band pixel image.
DCT	1-D discrete cosine transform of 176x144 3-band pixel image.
<b>Applications</b>	
EDGE	Edge detection using a simple 3x3 convolution mask (Laplacian operator) of 176x144 3-band pixel image.
SMF	Scalar median filter of 176x144 3-band pixel image with a 3x3 filter.
VMF	Vector median filter of 176x144 3-band pixel image with a 3x3 filter.
VQ	Vector quantization for color image and video compression of 176x144 3-band pixel image.

## 2.2. QCPX ISA extension

The QCPX instruction set applied to current microprocessor ISAs has been designed to achieve both efficient color pixel processing and greater subword parallelism. The QCPX functionality differs from other multimedia extensions by including a specific color data representation supported in hardware. Basically, QCPX performs on the number of 16-bit YCbCr color pixels packed into a word of the processor. For example, if the word size of a machine is 32 bits, QCPX supports parallel operations on two packed YCbCr data. In addition, QCPX includes a color-packed accumulator to provide a solution to overflow and other issues caused by the multiply-accumulate and absolute-distance-accumulate operations that are widely used in image and video processing algorithms (e.g., DCT, ME, VMF, VQ, and convolution). Therefore, the use of the QCPX accumulator provides two advantages: (1) provides high precision and (2) reduces the amount of packing/unpacking, truncating multiplication, and saturation overhead instructions inherent in the existing subword-parallel computations. This approach is similar to the MIPS's MDMX extension, except that the QCPX accumulator supports a specific color format (YCbCr) to accelerate color multimedia processing applications. Fig. 1 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.

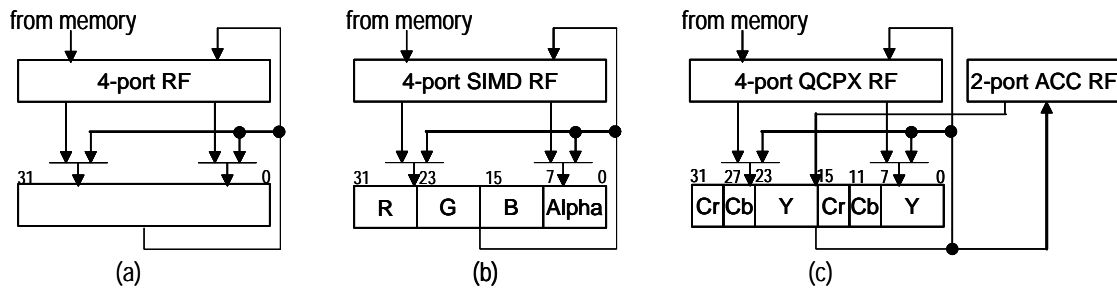


Fig. 1. Type of operations: (a) a baseline 32-bit operation, (b) a 32-bit SIMD operation, and (c) a 32-bit QCPX operation.

A 32-bit QCPX integer function unit performs on two 16-bit color pixels (an 8-bit for Y and 4-bits each for Cr and Cb components) in parallel in a single cycle with a small increase in functional unit complexity. Since the human visual system is less sensitive to color, the chrominance space could be implemented using shorter fields (e.g., Cr and Cb fields are 4 bits rather than 8) while providing satisfactory image quality [22]. Unlike typical multimedia ISA extensions (e.g., Intel's MMX and MIPS's MDMX), QCPX obtains substantial performance and code density improvements through implicit support for color pixel processing rather than depending solely on generic 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.

### 2.2.1 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, and SHIFT\_CRCBY [left|right] instructions, which are the most frequent operations in many color image and video computations. For example, the AVERAGE\_CRCBY instruction adds pairs of subword elements (Y, Cb, and Cr) in the two source registers and then performs a divide by two for each added Y, Cb, and Cr component. Fig. 2-(a) and (b) illustrate the parallel average and broadcast instructions, respectively.

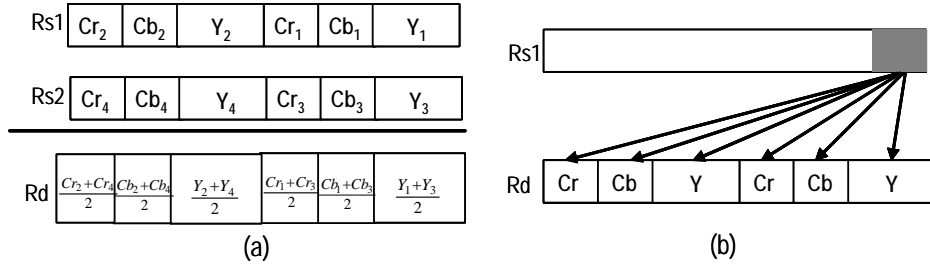


Fig. 2. (a) Parallel average instruction. (b) Broadcast instruction.

### 2.2.2 Parallel compare instructions

Parallel compare instructions compare pairs of subword elements (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 to 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 perform on two YCbCr color pixels concurrently to produce masks that can be used to reduce branches. However, the MIN\_CRCBY instruction compare pairs of two YCbCr values in the two source registers, selecting the smaller of the Y, Cb, and Cr components independently and returning them in the form of two packed YCbCr values. The MAX\_CRCBY instruction is similar, except that it returns the larger values. Fig. 3-(a) and (b) show the MIN\_CRCBY and MAX\_CRCBY instructions, respectively.

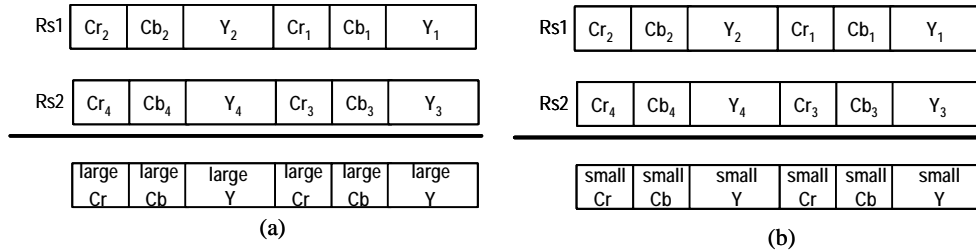


Fig. 3. (a) Parallel maximum instruction. (b) Parallel minimum instruction.

### 2.2.3 Special-purpose instructions

A 128-bit color-packed accumulator is used to support special-purpose QCPX instructions, such as MACC\_CRCBY (multiply-accumulate) and ADACC\_CRCBY (absolute-distance-accumulate). The packed accumulator provides a solution of overflow and other issues caused by packing data as tightly as possible. Fig. 4 shows an absolute-distance-accumulate instruction. 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.

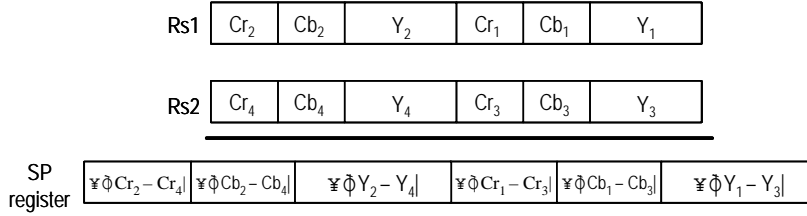


Fig. 4. Absolute-distance-accumulate instruction.

### 2.3. Benefits of QCPX combined with LU

QCPX combined with LU provides higher degrees of parallelism from across loop iterations, within basic blocks, and within the color space representation (e.g., YCbCr). For instance, Fig. 5-(a), (b), and (c) present the inner loop of a block matching algorithm (BMA) of vector quantization (VQ) application, and after loop unrolling, and the loop from the perspective of QCPX-level parallelism, respectively.

```
for (i=0; i<4; i++) {
    diff0 += abs(curr_Y[i] - codebook_Y[i]);
    diff1 += abs(curr_Cb[i] - codebook_Cb[i]);
    diff2 += abs(curr_Cr[i] - codebook_Cr[i]);
}
```

(a)

```
for (i=0; i<4; i += 2) {
```

```
    diff0 += abs(curr_Y[i+0] - codebook_Y[i+0]);
    diff1 += abs(curr_Cb[i+0] - codebook_Cb[i+0]);
    diff2 += abs(curr_Cr[i+0] - codebook_Cr[i+0]);
    diff3 += abs(curr_Y[i+1] - codebook_Y[i+1]);
    diff4 += abs(curr_Cb[i+1] - codebook_Cb[i+1]);
    diff5 += abs(curr_Cr[i+1] - codebook_Cr[i+1]);
}
```

(b)

```
for (i=0; i<4; i += 2) {
```

```
    diff0 += abs(curr_Y[i+0] - codebook_Y[i+0]);
    diff1 += abs(curr_Cb[i+0] - codebook_Cb[i+0]);
    diff2 += abs(curr_Cr[i+0] - codebook_Cr[i+0]);
    diff3 += abs(curr_Y[i+1] - codebook_Y[i+1]);
    diff4 += abs(curr_Cb[i+1] - codebook_Cb[i+1]);
    diff5 += abs(curr_Cr[i+1] - codebook_Cr[i+1]);
}
```

(c)

Fig. 5. (a) Original loop. (b) After loop unrolling. (c) QCPX-level parallelism exposed after loop unrolling.

With QCPX and LU, the loop is unrolled, and multiple operands are packed in each register as shown in dot-line boxes. Assembly language fragments of isomorphic statements grouped together in dash-line boxes are then efficiently replaced with a set of equivalent QCPX instructions. Operands are effectively pre-packed in memory and do not have to be unpacked when processed in registers. Loop unrolling also provides a well-known reduction in overhead instructions for address calculations and loop control. QCPX coupled with LU provides the following benefits:

1. Reduction in branch and address generation overhead.
2. Reduction of memory access instructions for immediate results since immediate results are stored in the special-purpose register rather than in memory.
3. Reduction of the energy consumption significantly due to a large reduction of the executed instructions, cache accesses, and branch frequencies.

### 2.4. Simulation tools

In the experiment, we used the Simplescalar-based toolset, which is an infrastructure for out-of-order superscalar modeling, to simulate a superscalar without and with MDMX-like or QCPX ISA extensions. Its architecture (Potable ISA or PISA) is driven from the MIPS-IV ISA [23]. The QCPX and MDMX-like instructions were synthesized by adding instructions in the assembly files. Since there is no efficient compiler support to automate MDMX-like and QCPX codes, we created the QCPX and MDMX-like versions of the benchmarks by replacing fragments of the baseline

assembly language with ones containing QCPX or MDMX-like instructions. The loops were unrolled, and the loop bodies were replaced by a set of equivalent MDMX-like and QCPX instructions. All the versions of the benchmarks have the same parameters, data set, and calling sequence. Since the target platform is an embedded system, operating system interface code (e.g., file system access) is not included in this study. Additional typical optimizations of embedded compilers were applied to the baseline version to enhance ILP (multiple issue and out-of-order issue) before the QCPX and MDMX-like versions were created. The simulations are executed on the same configuration parameters of a processor shown in Table 2. In addition, the architectural power simulator Watch [21] was used to determine each benchmark’s energy consumption.

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>

### 3. ANALYSIS OF RESULTS

In the experiment, the three different versions of the benchmark programs: (1) the baseline 32-bit ISA without subword parallelism, (2) the baseline plus 32-bit MDMX-like ISA [6], and (3) the baseline plus 32-bit QCPX ISA were executed on a modified version of the SimpleScalar simulator. In addition, additional simulations were performed with and without loop unrolling on these versions by replicating the body  $n$  times, where  $n$  is the unrolled factor. Their profile statistics obtained from the SimpleScalar-based simulator were then applied to the Watch power simulator to evaluate 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 study comparison. Table 3 summarizes each benchmark version’s performance and energy consumption, where EDGE, SMF, and VMF use the unrolled factor of 3; others use the LU factor of 4.

#### 3.1. Impact of QCPX plus LU on the performance

Fig. 6 illustrates the execution performance (speedup in executed cycles) of the baseline, MDMX-like, and QCPX versions with and without LU over the baseline version without LU. The use of QCPX enables a significant reduction in the executed instruction counts for all the benchmarks because of processing six color components in parallel. Loop unrolling eliminates a significant number of branches and reduces the total number of issued instructions by eliminating unnecessary increments and decrements between adjacent blocks. More interestingly, loop unrolling applied to the QCPX-optimized programs is more effective in the execution performance than applying to the baseline programs. LU

applied to QCPX-optimized programs achieves an additional speedup ranging from 1.66 to 3.5, while LU applied to baseline programs achieves only a speedup ranging from 1.01 to 1.19. This is because LU plus QCPX allows for a better schedule by increasing the basic block size, reducing the significant number of loop overhead, address calculation, and computational instructions. The use of QCPX plus LU provides a speedup ranging from 3.79 (SMF) to 7.89 (DCT) over the baseline version while achieving a speedup ranging from 1.83 (ME) to about 2 (VQ and DCT) over the MDMX-like plus LU version.

Table 3. A comparison of the performance and energy consumption of the baseline, MDMX-like, and QCPX versions with and without loop unrolling.

Benchmark	Version	# of executed instructions	# of cycles	Total energy (nJ)
EDGE	Baseline	29,039,536	15,731,814	230,165,711
	Baseline plus LU	27,563,352	15,057,240	218,310,734
	MDMX-like	11,372,209	4,892,165	78,403,986
	MDMX-like plus LU	9,121,384	3,945,080	64,221,254
	QCPX	5,852,647	2,567,534	41,295,438
	QCPX plus LU	4,640,208	2,020,285	32,952,536
SMF	Baseline	74,510,711	33,135,080	567,084,467
	Baseline plus LU	69,914,491	31,412,993	528,909,860
	MDMX-like	47,950,961	21,530,736	350,011,040
	MDMX-like plus LU	37,704,879	17,455,709	269,307,634
	QCPX	23,978,441	10,872,693	175,030,764
	QCPX plus LU	18,862,657	8,751,759	134,615,193
VMF	Baseline	215,405,293	116,221,809	1,708,569,488
	Baseline plus LU	177,436,952	97,716,725	1,396,750,308
	MDMX-like	105,317,925	46,186,466	827,254,335
	MDMX-like plus LU	65,176,721	30,863,577	479,462,808
	QCPX	53,011,538	23,153,363	415,612,906
	QCPX plus LU	32,589,280	15,450,924	239,549,479
VQ	Baseline	1,051,030,648	501,593,681	7,718,016,820
	Baseline plus LU	997,078,671	482,108,915	7,388,880,446
	MDMX-like	345,102,076	145,615,328	2,351,238,493
	MDMX-like plus LU	288,866,915	127,823,906	2,018,050,734
	QCPX	172,555,372	72,829,736	1,175,596,197
	QCPX plus LU	144,437,795	63,934,058	1,008,991,313
DCT	Baseline	186,179,946	96,321,954	1,470,629,816
	Baseline plus LU	180,452,232	96,019,246	1,425,505,023
	MDMX-like	64,054,777	27,667,930	457,776,363
	MDMX-like plus LU	53,308,888	24,402,420	386,171,381
	QCPX	32,056,351	13,828,616	229,066,099
	QCPX plus LU	26,708,776	12,207,175	193,488,830
ME	Baseline	4,337,061,259	2,271,018,469	33,886,561,164
	Baseline plus LU	4,279,491,481	2,226,385,183	33,471,753,198
	MDMX-like	1,320,933,978	596,491,083	9,477,466,173
	MDMX-like plus LU	1,167,242,684	546,234,831	8,745,288,663
	QCPX	719,872,805	326,316,514	5,237,187,152
	QCPX plus LU	636,960,402	298,010,011	4,770,238,873

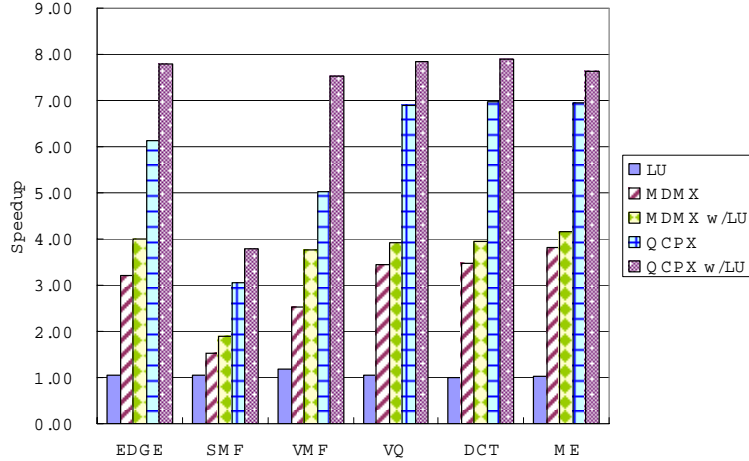


Fig. 6. Speedups over the baseline version.

### 3.1.1 Edge detection

Edge detection (EDGE) is an effective technique in image and video analysis, such as segmentation, registration, and identification of objects in a scene [24]. It is an image-to-image operation where each output pixel is determined by multiplying the input pixel and its surrounding neighbor pixels with a filter operator (a mask) in the window and summing the products.

We implemented an edge detection application by taking into account local changes in both the luminance and chrominance components. The most time critical operation in this application is the multiply accumulation between a color pixel and a coefficient operator (Laplacian operator [12] in this paper) in the 3x3 window. With the BCAST\_CRCBY instruction, each of coefficient values saved in memory is distributed to the Y, Cb, and Cr positions of the register. The MACC\_CRCBY instruction then multiplies each color pixels in a source register with a coefficient value independently while accumulating its calculation results in a packed accumulator register. This process continues until the ZACC instruction is called. As a result, a large amount of the issued instructions committed and memory accesses for intermediate results (since immediate results are stored in the accumulator register rather than in memory) is decreased. In addition, loop unrolling (with the unrolling factor of 3) reduces loop overhead and address calculation instructions. The EDGE application using QCPX plus LU achieves performance improvement of 679% than the baseline version without LU and 95% than the MDMX-like plus LU version.

### 3.1.2 Vector median filter

Vector median filters (VMFs) are widely used in color image and video because of their ability to reduce impulse noise while preserving edge capabilities [13]. However, high computation makes them impractical for real-time systems. In this application, we implemented the VMF algorithm by using QCPX plus LU to improve the execution performance. The construction of the VMF implementation is based on the YCbCr color space rather than the RGB color space, and also its construction is based on the norm (L1) to order vectors according to their relative magnitude differences in the window.

Given  $N$  sample vectors  $\{\mathbf{C}_1, \mathbf{C}_2 \dots \mathbf{C}_N\}$ , each containing a Y, Cb, and Cr channel, the output of the 1-norm (Manhattan distance) operator is defined as follows

$$\left\{ \begin{array}{l} \mathbf{C}_{VM1} \in \{\mathbf{C}_1, \mathbf{C}_2 \dots \mathbf{C}_N\} \\ \sum_{i=1}^N \|\mathbf{C}_{VM1} - \mathbf{C}_i\|_1 \leq \sum_{i=1}^N \|\mathbf{C}_j - \mathbf{C}_i\|_1 \quad j = 1 \text{ to } N \end{array} \right. \quad (1)$$

where  $N$  is the number of pixels in the window.

The most time critical operation is the absolute-distance accumulation, which computes the absolute magnitude difference between pixels in the inner loop and accumulates it. With the ADACC\_CRCBY instruction, a VMF operation can be performed on the two windows of pixels in parallel. ADACC\_CRCBY compares  $N$  registers, each containing two packed YCbCr values, to one another in the window (3x3 in this paper) while summing their absolute magnitude differences in parallel. Furthermore, the Y, Cb, and Cr values of each pixel are processed simultaneously. This results in a significant reduction of logic overhead and memory access instructions. Loop unrolling reduces additional branch and address calculation instructions. The LU plus QCPX-optimized version achieves an overall speedup of 7.52 over the baseline version and 1.99 over the MDMX-like plus LU version.

### 3.1.3 Scalar median filter

Like VMF, scalar median filter (SMF) is also an image restoration technique that eliminates impulse noise spikes from an image by taking the median pixel value for the output. A drawback of this application is that it separately replaces the corrupted color values (e.g. Y, Cb, Cr) with corresponding neighborhood color values, producing uncorrelated pixels [25].

For this implementation, a 3x3 window is used. The key computational complexity of SMF is to sort pixels in the window. The MAX\_CRCBY and MIN\_CRCBY instructions compare pairs of sub-elements in the two source registers, outputting the minimum or maximum values of the corresponding elements. Furthermore, two windows of pixels are processed in parallel since two pixels are loaded into the register concurrently for further processing. Also, loop unrolling extends the size of loop bodies by replicating the bodies 3 times, allowing for a better schedule. As a result, significant branch instructions as well as arithmetic and logical instructions are reduced. The LU plus QCPX-optimized version achieves a speedup of 3.79 and 1.99 versus the baseline version and the MDMX-like version, respectively.

### 3.1.4 Vector quantization

Vector quantization (VQ) is an attractive technique for image and video compression [26]. A vector quantization is defined as a mapping of  $k$ -dimensional vectors in the vector space  $\mathbf{R}^k$  into a finite set of vectors  $\mathbf{Y} = \{y_i, i=1, \dots, N\}$ , 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 256-word codebook is used to achieve a 0.5 bit per pixel encoding of 24-bit color images, using 4x4 ( $k=16$ ) vectors. The ADACC\_CRCBY instruction efficiently replaces the most time critical operations of the distortion calculation between a 4x4 input block and a local codeword in the block-matching algorithm (BMA). In addition, loop unrolling helps to reduce loop overhead of BMA by replicating the bodies 4 times in this implementation. The VQ application using QCPX plus LU shows an overall speedup of 7.85 over the baseline version and about 2 over the MDMX-like plus LU version due to a significant reduction of ALU and branch instructions.

### 3.1.5 Discrete cosine transformation (DCT)

DCT is one of the most computationally intensive kernels in image and video compression such as JPEG, H.263, and MPEG-1/2 baseline encoding [27]. We simulated an 8x8 1-dimensional DCT.

For this implementation, cosine terms were used as coefficient and preloaded into the system; a fixed-point arithmetic was used to perform the calculation; and a 4:4:4 (=Y:Cb:Cr) video format was used. In DCT, multiplications of an 8x8 DCT coefficient matrix with an 8x8 input data matrix are the most time consuming process. Similar to EDGE application's process, the BCAST\_CRCBY instruction operates to distribute each of coefficient values saved in memory into Cr, Cb, and Y positions of the register. The MACC\_CRCBY instruction then multiplies and accumulates each sum of the Y, Cb, and Cr components individually in a packed accumulator register. The use of QCPX with LU (the LU factor of 4) provides a speedup of 7.89 versus the baseline version and 2 versus the MDMX-like with LU version.

### 3.1.6 Motion estimation (ME)

Motion estimation (ME) has become a critical component to video compression algorithms. In BMA, compression is achieved by subdividing the current frame into small reference blocks and then finding the best match for each block among the available blocks in the previous frame. In this implementation, the macroblock, a 16 x 16 region, was used as the basic block, and a search range of -15 to +16 pixels was used in accordance with the set of parameters typically adopted by ITU-T H.26x and ISO MPEG standards [27]. Since our objective is to achieve accurate motion estimates, both the luminance and chrominance components are used for executing the ME kernel rather than using luminance only.

Like VQ, the most time critical operation is the absolute-distance accumulation that computes the sum of the absolute magnitude differences between a reference block of pixels and all the candidate blocks of pixels of the search area. The ADACC\_CRCBY instruction compares two adjacent 16x16 reference blocks in the current frame to candidate blocks in the previous frame, using the sum of absolute magnitude differences between pixels as a comparator. Again, loop unrolling allows for a better schedule by increasing the basic block size, reducing total instructions committed. The ME kernel using QCPX plus LU shows an overall speedup of 7.62 over the baseline version and 1.83 over the MDMX-like plus LU version.

### 3.2. Impact of QCPX plus LU on the energy consumption

Shorter execution times result in lower energy consumption when execution platforms employ identical clock rates, implementation technology, and processor parameters [28]. QCPX plus LU significantly reduces the energy consumption on color image and video processing algorithms. Experimental results using the Watch-based power simulator on a set of the selected applications indicate that the QCPX plus LU version reduces the energy consumption from 75% (SMF) to 86% (DCT) over the baseline version and 45% (ME) to 50% (DCT and VQ) over the MDMX-like plus LU version due to a large reduction of the total instructions executed and cache accesses. Fig. 7 illustrates relative total energy consumptions of the QCPX and MDMX-like versions with and without LU versus the baseline version. As expected, DCT using QCPX plus LU shows the highest reduction rate in the energy consumption over the baseline version because of the largest reduction rate in the total executed instructions and cache accesses. Since ME has the longest execution cycle counts, it consumes the greatest energy of the algorithms.

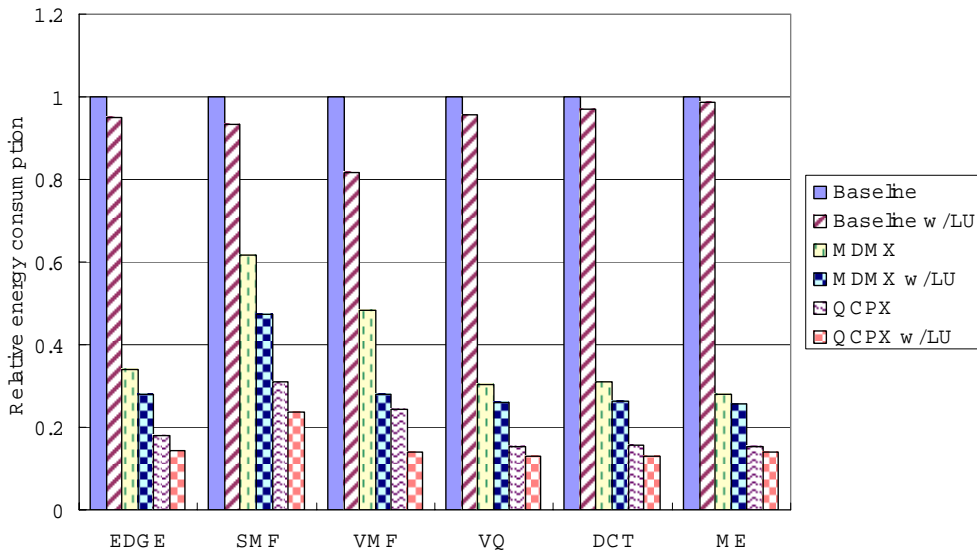


Fig. 7. Relative total energy consumptions over the baseline version.

Since the ALUs, branches, and cache accesses are reduced by using QCPX and LU, less energy is spent on speculative execution and caches accesses. Fig. 8 shows relative energy consumptions of the QCPX and MDMX-like versions with and without LU in each functional unit over the baseline version. As we expected, loop unrolling reduces the number of

branch instructions significantly for all applications, resulting in reduction of the power dissipation of the branch prediction hardware and the energy consumption as shown in Fig. 8. Removing branches by using loop unrolling also help to decrease the average fetch unit power dissipation. The fetch unit can fetch large basic blocks without being interrupted by taken branches, providing more work for the renaming unit and filling up the RUU faster. Therefore, when the instruction queue and RUU are full, the fetch unit is stalled during the cycles, resulting in reducing its power dissipation. However, loop unrolling provides less effectiveness for reducing the energy consumption in ALUs and caches. Our MDMX-like instructions provides a significant reduction of the energy consumption in ALUs and caches for all applications, while the energy reduction of the branch unit is varied depending on the programs. DCT using the MDMX-like instructions did not reduce the branch instructions at all; others reduced the branch instructions from 15% (EDGE) to 57% (ME). Since QCPX operates on two packed YCbCr color pixels, it reduces a huge number of loop overhead as well as computational instructions. This results in a large reduction of the energy consumption for each functional unit. Greater performance and energy efficiency were achieved by using QCPX combined with loop unrolling as shown in Fig. 6 and Fig. 7, respectively.

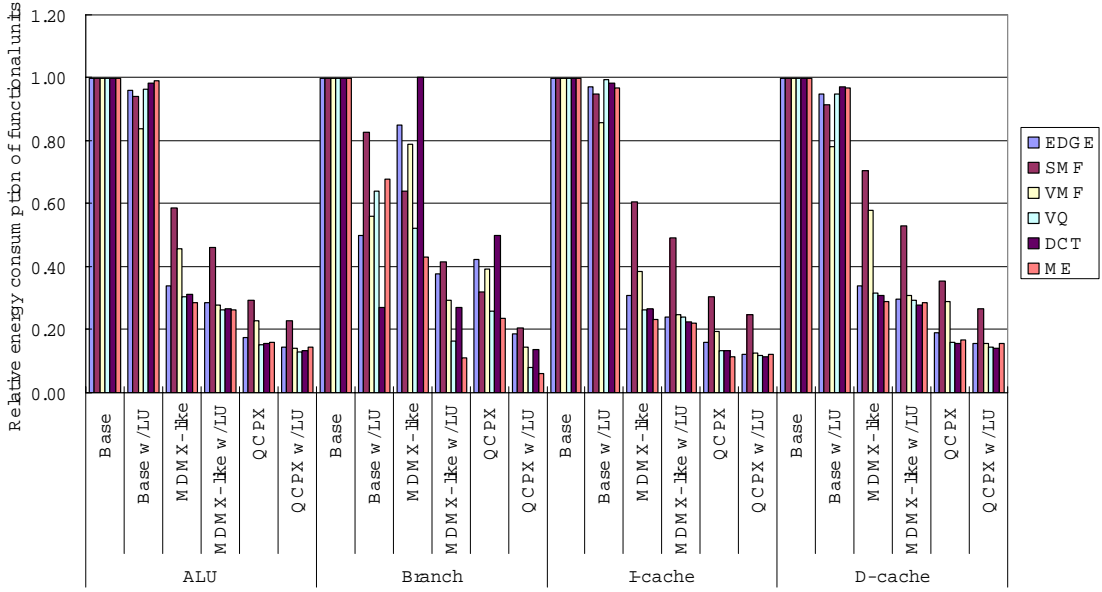


Fig. 8. Relative energy consumptions of functional units over the baseline version.

#### 4. CONCLUSIONS

In this paper, we have examined the impact of the QCPX instruction set and loop unrolling for color image and video processing applications on dynamically scheduled ILP superscalar processors. QCPX solves the problems inherent in the existing subword-parallel ISA extensions (storage data types and operands that are not in a format amenable to SIMD processing) through implicit support for color pixel processing in addition to the color packed accumulator, providing greater performance and code density improvement. With the addition of loop unrolling, QCPX exploits a higher degree of parallelism across loop iterations, within basic blocks, and within the color space representation (e.g., YCbCr). The combined results also provide a significant reduction in the energy consumption. Our results have shown that the LU plus QCPX-optimized version achieves a speedup ranging from 3.8 to 7.9 and reduces the energy consumption from 76% to 87% over the baseline version for the color image and video benchmarks. The LU plus QCPX version also outperforms the MDMX-like version in a same machine platform. These results demonstrate that QCPX combined with LU is an effective approach to supporting color video workloads on portable systems. In the future, we will evaluate the QCPX effectiveness for the completed video applications such as H.263 and MPEG on both VLIW and superscalar processors. Also, detailed power and energy analysis of QCPX while varying the LU factors will be conducted.

## REFERENCES

- [1] K. Diefendorff and R. Dubey, "How multimedia workloads will change processor design," *IEEE Computer*, vol. **30**, no. 9, pp. 43-45, 1997.
- [2] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol. **16**, no. 4, pp. 42-50, 1996.
- [3] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol. **20**, no. 4, pp. 28-39, 2000.
- [4] R. B. Lee, "Subword parallelism with MAX-2," *IEEE Micro*, vol. **16**, no. 4, pp. 51-59, 1996.
- [5] 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, 1996.
- [6] *MIPS extension for digital media with 3D*, Technical Report <http://www.mips.com>, MIPS technologies, Inc., 1997.
- [7] R. Sites, Ed., *Alpha Reference Manual*, Burlington, MA: Digital, 1992.
- [8] H. Nguyen and L. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology," *ICS'99*, pp. 11-20, 1999.
- [9] A. Peleg, S. Wilkie, and U. Weiser, "Intel MMX for multimedia PCs," *Communications of the ACM*, vol. **40**, no. 1, pp. 25-38, 1997.
- [10] 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, 1999.
- [11] T. Viero, K. Oistamo, and Y. Neuvo, "Three-dimensional median-related filters for color image sequence filtering," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. **4**, no. 2, pp. 129-142, 1994.
- [12] M. A. Ruzon and C. Tomasi, "Color edge detection with the compass operator," *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 160-166, 1999.
- [13] J. Astola, P. Haavisto, and Y. Neuvo, "Vector median filters," *Proc. of the IEEE*, vol. **78**, no. 4, pp. 678-689, 1990.
- [14] K. A. Prabhu and A. N. Netravali, "Motion compensated component color coding," *IEEE Trans. Commun.*, vol. **COM-30**, pp. 2519-2527, 1982.
- [15] M. Barni, F. Bartolini, and A. Piva, "Multichannel watermarking of color images," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. **12**, no. 3, pp. 142-156, 2002.
- [16] H. Kimata, Y. Yashima, and N. Kobayashi, "Edge preserving pre-post filtering for low bitrate video coding," *Proc. IEEE Intl. Conf. on Image Processing*, pp. 554-557, 2001.
- [17] J. J. Dongarra and A. R. Hinds, "Unrolling loops in Fortran," *Software-Practice and Experience*, vol. **9**, no. 3, pp. 219-226, 1979.
- [18] S. Weiss and J. E. Smith, "A study of scalar compilation techniques for pipelined supercomputers," *Proc. of Second Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 105-109, 1987.
- [19] S. Larsen and S. Amarasinghe, "Exploiting superword level parallelism with multimedia instruction sets," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 145-156, 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," *Intl. Symposium on Computer Architecture*, pp. 83-94, 2000.
- [22] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, 1993.
- [23] C. Price, *MIPS IV Instruction Set, revision 3.1*, MIPS Technologies, Inc., Mountain View, CA, 1995.
- [24] G. A. Baxes, *Digital image processing*, John Wiley & Sons Inc., 1994.
- [25] 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, 1994.
- [26] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Press, 1992.
- [27] *Coding of Moving Pictures and Audio*, ISO/IEC JTC1/SC29/WG11 N3312, 2000.
- [28] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," *Proc. Symp. Low Power Electron.*, pp. 38-39, 1994.