Optimization of Content-Based Image Retrieval Functions

Asadollah Shahbahrami[1, 2]
[1]Computer Engineering Laboratory
Delft University of Technology
2628 CD Delft, The Netherlands
A.Shahbahrami,B.H.H.Juurlink@TUDelft.nl

Ben Juurlink[1]
[2]Department of Computer Engineering
Faculty of Engineering
University of Guilan
Rasht, Iran

Abstract

Feature extraction and similarity measurement are two important operations in content-based image retrieval systems. We optimize and vectorize typical feature extraction algorithms, mean and standard deviation, and some similarity measurement functions such as the Sum-of-Squared-Differences (SSD), the Sum-of-Absolute Differences (SAD), and histogram intersection on a general-purpose processor enhanced with SIMD extensions. In the straightforward implementation of the mean and standard deviation, there are two passes, one to compute the mean and one to compute the standard deviation. We use a single-loop approach that computes both the mean and the standard deviation in a single pass. This technique yields a speedup of up to 1.85 over the double-loop implementation. We vectorize the single-loop implementation using the MMX and SSE2 extensions. The vectorized versions improve performance by a factor of up to 14.49. In addition, we vectorize the SSD, SAD, and histogram intersection similarity measurements using SSE. The vectorized versions provide a maximum speedup of 1.45, 2.33, and 5.24 for the SSD, the SAD, and histogram intersection, respectively, over the optimized scalar implementations.

Keywords: Feature Extraction, Similarity Measurements, Vectorization, SIMD.

1 Introduction

In a Content-Based Image Retrieval (CBIR) system, an image is represented as a vector of features and the similarity between images is determined by measuring the similarity between feature vectors. In other words, feature extraction and similarity measurement are two important stages in CBIR systems [5]. The mean and standard deviation features are the most common features in CBIR systems. For example, in [14], it has been shown that the mean and standard deviation are efficient and effective features in representing images color distribution.

Among the different similarity measurements, the Euclidean distance or Sum-of-Squared-Differences (SSD) and the Sum-of-Absolute Differences (SAD) have been found to be the most useful. For instance, in [16] eight similarity measurements for image retrieval have been evaluated. In terms of retrieval effectiveness and retrieval efficiency, the SSD and SAD functions are better than other functions. Histogram intersection is another similarity measurement commonly used in CBIR systems [16, 4].

Our objective in this paper is to increase the operating speed of CBIR systems by optimizing and vectorizing a set of typical feature extraction algorithms, mean and standard deviation, and some similarity measurement functions such as the SSD, the SAD, and histogram intersection. The main reason for this is that time is critical in on-line CBIR systems as the response time requires to be low for good interactivity [3]. Since both feature extraction and similarity measurement algorithms exhibit significant amounts of data-level parallelism [11, 13], they could be implemented using the Single-Instruction Multiple-Data (SIMD) instructions supported by most General-Purpose Processors (GPPs). In this paper we make the following contributions compared to other works.

- We use a single-loop approach to compute the mean and standard deviation features. In the straightforward implementation, there are two passes, one to compute the mean and the other to compute the standard deviation, while in the single-loop approach, there is a single pass to compute both features. This technique yields a speedup of up to 1.85 over the straightforward implementation.
- In order to increase the performance of the single-loop technique, we vectorize it using the MMX and SSE2 extensions. The vectorized implementations yield a speedup of up to 14.49.

This research was supported in part by the Netherlands Organization for Scientific Research (NWO).

- We vectorize the SSD, the SAD, and histogram intersection similarity measurements using SSE. The SIMD implementations improve performance by up to 1.45, 2.33, and 5.24 for the SSD, the SAD, and histogram intersection, respectively, over the optimized scalar implementations.

This paper is organized as follows. Related work is discussed in Section 2. Section 3 describes the background information about feature extraction, similarity measurements, and some SIMD extensions such as MMX [9], SSE, and SSE2 [10]. Section 4 discusses the single-loop algorithm and its vectorization and Section 5 describes the vectorization of similarity measurement functions. Section 6 presents the performance results and, finally, conclusions are drawn in Section 7.

2 Related Work

In this section, we discuss related work. We first discuss some work related to accelerating feature extraction algorithms. Then we discuss related acceleration techniques for similarity measurements.

Histogram features are usually used in CBIR systems [4]. The vectorization of the histogram computation is a challenging problem due to memory collisions [1]. Kotoulas et al. [7] have proposed a novel parallel hardware architecture for local histogram comparison. An image is divided into n subimages and a histogram is computed for each subimage. A mean value is computed in order to determine the similarity between subimages. In addition, a parallel architecture has been proposed to calculate the features of subimages independently. Shahbahrami et al. [12] have proposed two techniques, hierarchical structure and parallel comparators, to vectorize the histogram calculation using SIMD instructions. In the hierarchical structure, they either loaded or stored data from n locations at the same time using indexed load/store instructions. The parallel comparators have been used to count the number of subwords that are the same. After counting the number of subwords that are the same, the results are added to the values in the histogram array simultaneously. Chung et al. [2] have focused on multiple instruction multiple data machines to parallelize the feature extraction algorithms, while we use SIMD architectures.

Many processor vendors have designed special-purpose SIMD instructions to accelerate different similarity measurements. Example is the SSE instruction `psadbw` [10] which accelerates motion estimation based on the SAD function. This instruction has limited usefulness except for the motion estimation kernel [11]. Shahbahrami et al. [11] have discussed the limitations of special-purpose instructions for similarity measurements supported by the SIMD extensions. They have designed and implemented a few new SIMD instructions using wider registers to implement different similarity measurement functions. Recently, the SSE4 extension supports special-purpose `pabsb`, `pabsw`, and `pabsd` instructions for 8-, 16-, and 32-bit integers, respectively, but there are no such instructions for single-precision floating-point numbers.

3 Background

In this section, we briefly describe the straightforward implementation of the mean and standard deviation, some similarity measurement algorithms, and SIMD extensions.

3.1 Straightforward Implementation

The mean and standard deviation features of an $N \times M$ image are computed using the following equations.

$$\bar{x} = \frac{\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_{ij}}{NM}. \tag{1}$$

$$\delta = \sqrt{\frac{1}{NM}\sum_{i=0}^{N-1}\sum_{j=0}^{M-1}(x_{ij}-\bar{x})^2}. \tag{2}$$

In these equations, x_{ij} represents the pixel value in row i and column j, $\bar{x}$ denotes the mean, and δ denotes the standard deviation. As these equations show, there are two passes to compute the mean and the standard deviation; one to compute the mean and the other to compute the standard deviation.

In a straightforward implementation, it is necessary that whole images are stored in memory. This is because this implementation requires two passes through the image pixels. It costs memory, bandwidth, and performance. For example, when the images are large, this implementation requires a large memory and this reduces performance. In addition, some embedded devices such as Personal Digital Assistants (PDAs) do not have a large memory to store whole images. Additionally, as image pixels already stored in memory in two pass algorithm, this technique is not applicable for on-line processing. Consequently, the straightforward implementation is not appropriate for applications such as searching image databases by content using PDA devices and on-line processing [15]. Therefore, we use a single-loop implementation to overcome this problem.

3.2 Similarity Measurements

Equations (3), (4), and (5) define the SSD, SAD, and histogram intersection similarity measurements, respectively [4].

$$SSD(f_q, f_t) = \sum_{i=0}^{n-1} (f_q[i] - f_t[i])^2. \quad (3)$$

$$SAD(f_q, f_t) = \sum_{i=0}^{n-1} (|f_q[i] - f_t[i]|). \quad (4)$$

$$intersection(f_q, f_t) = \sum_{i=0}^{n-1} min(f_q[i] - f_t[i]). \quad (5)$$

Here $f_q = f_q[0], f_q[1], ..., f_q[n]$ represents the query feature vector, $f_t = f_t[0], f_t[1], ..., f_t[n]$ denotes the target feature vector, and n is number of features in each feature vector.

3.3 SIMD Extensions

In order to increase the performance of multimedia applications, GPPs vendors have enhanced their Instruction Set Architectures (ISAs) with SIMD extensions. These ISA extensions use the Subword Level Parallelism (SLP) concept [8]. A subword is a smaller precision unit of data contained within a word. In SLP, multiple subwords are packed into a word and then whole the word is processed. In other words, there are several data elements in a word and all elements in that word can be processed in SIMD style.

Intel's MMX [9] was the first SIMD extension to process at most eight 8-bit data elements simultaneously. This extension supported only integer data types. It was followed by Streaming SIMD Extensions (SSE) and SSE2 from Intel [10] to support integer as well as floating-point media instructions. The SSE and SSE2 can process two 128-bit wide operands in a single instruction.

4 Single-loop Approach to Compute Mean and Deviation

This section discusses the single-loop approach to compute the mean and the standard deviation and its vectorization using the SSE2 extension.

4.1 Single-loop Algorithm

In the single-loop approach both the mean and the standard deviation features are computed in a single pass. The following equations illustrate how these features can be computed in a single pass for an $N \times M$ image.

$$\begin{aligned}
\delta^2 NM &= \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (x_{ij}^2 - 2x_{ij}\bar{x} + \bar{x}^2) \\
&= \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - 2\bar{x} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij} + NM\bar{x}^2 \\
&= \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - 2\bar{x}NM\bar{x} + NM\bar{x}^2 \\
&= \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - NM\bar{x}^2 \\
&= \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - NM(\frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}}{NM})^2. \\
\delta^2 &= \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - (\frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}}{NM})^2. \\
\delta &= \sqrt{\frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}^2 - (\frac{\sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_{ij}}{NM})^2} \quad (6)
\end{aligned}$$

Equation (6) shows that to calculate the standard deviation, we should compute two values, the sum of squared pixel values divided by the number of pixels and the square of the sum of pixel values divided by the number of pixels.

4.2 Vectorization of Single-loop

The single-loop algorithm is mapped onto a GPP enhanced with SIMD extension. It loads several pixel values using a single load instruction and process them simultaneously. The appropriate data type for vectorization depends on the image size. The largest image size that we used in this paper is 4096×4096. Hence, we need a 40-bit data type for the sum of squared pixel values and a 32-bit data type for the sum of pixel values as the following equations show.

$$\sum_{i=0}^{4095} \sum_{j=0}^{4095} (255)^2 < (2^{12})^2(2^8)^2 = 2^{40}. \quad (7)$$

$$\sum_{i=0}^{4095} \sum_{j=0}^{4095} 255 < (2^{12})^2 2^8 = 2^{32}. \quad (8)$$

We use 64-bit precision instead of 40-bit precision because the latter precision does not match the available data types. We implemented the single-loop approach using MMX as well as SSE2. The MMX implementation uses 4×16-bit SIMD instructions and the SSE implementation uses 8×16-bit instructions. In the SSE2 implementation, in each iteration of the inner loop, eight pixels are processed. First, eight pixel values are loaded into a media register using a single SIMD instruction. Then, the sum of squared pixel values and the sum of pixel values are computed by two `pmaddwd` instructions. In the inner loop, 32-bit subwords are employed, while in order to avoid overflow, 64-bit subwords are used in the outer loop. In other words, in the outer loop, four 32-bit subwords are converted to two 64-bit subwords.

5 Vectorization of Similarity Measurements

In this section, the vectorization of the SAD similarity measurement is explained. The SIMD implementations of the other similarity measurements are very similar.

As discussed in Section 2, SSE and SSE4 provide special-purpose instructions for the SAD function for integer values, but there are no such instructions for floating-point values. Because the elements of the feature vectors are floating-point values, we should synthesize the SAD function by other SIMD instructions such as subtraction, comparison, and logical instructions. The SSE implementation of the SAD similarity measurement has two loops. In the inner loop, there are two SIMD load instructions, one for loading four elements of the target feature vector and the other for loading four elements of the query feature vector. Then the SAD of these two 4-elements single-precision floating-point vectors is computed. After finishing the execution of the inner loop, the four partial sums in a media register are summed using six instructions. The final result is stored in an element of the output vector. This procedure is repeated for other target feature vectors as well.

6 Performance Evaluation

In this section, the performance improvement of the single-loop approach and the SIMD implementations over the straightforward and scalar implementations is evaluated.

6.1 Experimental Environments

We assume that features of the target images are stored as a two-dimensional array of single-precision floating-point elements. In other words, a feature database is considered as a two-dimensional array. Each row of this array corresponds to a target image feature vector. The number of rows is the same as the number of target images. In addition, a separate feature vector for the query image is considered. All feature vectors have 512 data elements.

Two versions of each function were implemented: one in C and one in assembly using SIMD extensions. The different versions of each function employ the same algorithm and data types. Each program consists of three parts, for reading the input data, for performing the computation, and for storing the calculated data. Only the computation part was implemented in MMX, SSE2, and SSE and only the time taken by this part is reported. The single-loop algorithm has been implemented using 4- and 8-way parallel SIMD instructions by the MMX and SSE2 extensions, respectively. The similarity measurement algorithms have been implemented using the 4-way parallelism supported by the SSE extension. In addition, we have used the same input data for all C and SIMD implementations.

Processor	Intel Pentium 4
CPU Clock Speed	3.0GHz
L1 Data Cache	8 KBytes, 4-way set associative, 64 Bytes line size
L2 Cache	512 KBytes, 8-way set associative, 64 Bytes line size, On Chip

Table 1. Parameters of the experimental platform.

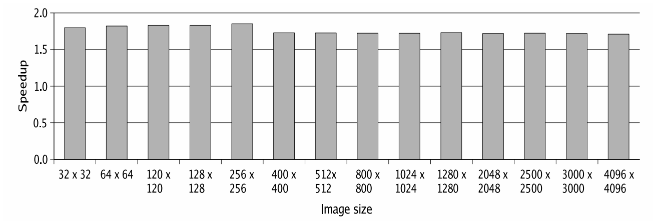

Figure 1. Speedup of the single-loop implementation over the double-pass implementation on the Pentium 4 processor for various image sizes.

All C programs were compiled using gcc with optimization level *-O2*. As experimental platform a 3.0GHz Pentium 4 processor was employed. The main architectural parameters of this system are summarized in Table 1.

All programs were executed on a lightly loaded system. Performance was measured using the IA-32 cycle counter [6]. In order to eliminate the effects of context switching and compulsory cache misses, the *K-best* measurement scheme and a *warmed up* cache were used.

6.2 Performance Evaluation Results

Figure 1 depicts the speedup of the single-loop implementation over the double-pass implementation on the Pentium 4 processor for various image sizes. The average speedup is 1.78. As the figure shows, the image size does not significantly impact the performance.

Figure 2 depicts the speedup of the 8-way parallel SSE2 implementation of the single-loop approach over the scalar version. The speedup ranges from 6.21 to 14.49. The following observations can be drawn from this figure. First, the speedups are higher for small images ($N \leq 512$) than

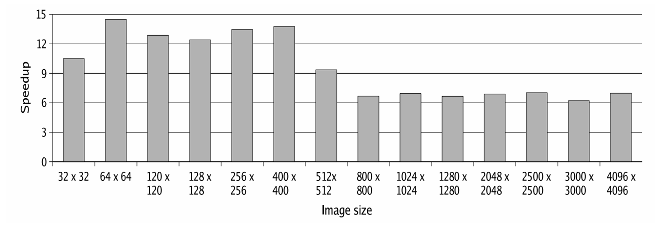

Figure 2. Speedup of the 8-way parallel implementation of the single-loop algorithm over the scalar version.

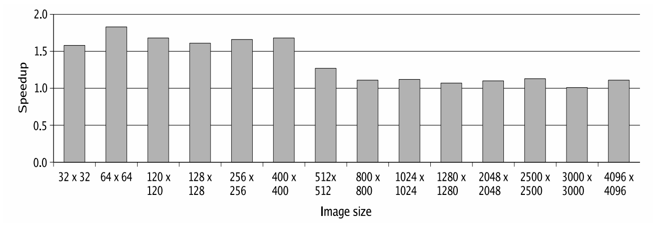

Figure 3. Speedup of the 8-way parallel implementation of the single-loop approach over the 4-way parallel implementation.

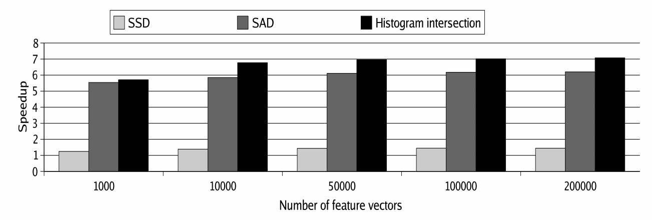

Figure 4. Speedup of the SIMD implementations of the different similarity measurements over the corresponding scalar implementations for different number of feature vectors. Each feature vector has 512 data elements.

for larger images. Second, since the SIMD implementation performs eight operations in one instruction, the expected maximum speedup is 8. However, the attained speedups for small images ($N \leq 512$) are larger than 8 and for other image sizes they are smaller than 8. These behaviors can be explained as follows. When $N \leq 512$, all reads almost hit either the L1 or the L2 data caches. Hence the speedups obtained for these image sizes are the speedups resulting from SIMD vectorization. SIMD instructions reduce the number of memory accesses, ALU instructions, and instruction decoding compared to the scalar implementation. In addition, SIMD instructions eliminate a significant number of loop overhead instructions, which increment or decrement index and address values. Additionally, this implementation uses short vector load and store instructions (16 bytes) while the C implementation loads two bytes in each load instruction. When $N > 512$ the speedup decreases because the SIMD implementation has become *memory-bound*. This means that the SIMD implementation incurs more memory stall cycles than the scalar implementation. This is because the SIMD implementation performs more ALU and multiplication operations in a single instruction and this significantly reduces the CPU component of the execution time. In other words, due to the SIMD implementation, the time spent in the computation part is relatively lower than the time spent in memory accesses.

To validate this claim, Figure 3 depicts the speedup of the 8-way parallel implementation of the single-loop approach over the 4-way parallel implementation. The speedup for small images ($N < 512$) is larger than 1.5, while for larger image sizes the average speedup is 1.11. This means that the performance does not scale very well for large image sizes compared to small image sizes.

In general, the straightforward and single-loop implementations need 28.17 and 16.01 cycles, respectively, to process one image pixel, while the 4- and 8-way SIMD implementations require 2.34 and 1.87 cycles, respectively.

Figure 4 depicts the speedups of the 4-way SIMD implementations of the different similarity measurements over the corresponding scalar versions for various number of feature vectors. The speedups range from 1.25 to 1.45 for the SSD, from 5.54 to 6.20 for SAD, and from 5.71 to 7.08 for histogram intersection. The speedups of the two latter implementations are larger than the former implementation. This is because there are conditional instructions in the scalar implementations of the SAD and histogram intersection functions, while in the C implementation of the SSD function, there are no such instructions. Conditional instructions can reduce the number of instructions per cycle (IPC) and this limits performance. Our results show that the C implementation of the SSD function is on average 5.53 and 4.59 times faster than the C implementations of the SAD and histogram intersection functions, respectively. In addition, the SIMD implementation of histogram intersection is on average 1.35 and 1.05 times faster than the SIMD implementation of SAD and SSD functions, respectively.

The SIMD implementation of the histogram intersection has four SIMD instructions in the inner loop, two SIMD load instructions and two SIMD ALU instructions, while the SIMD implementation of the SAD uses 13 instructions. This is the main reason why the speedups for the histogram intersection function are larger than the SAD. The speedups for the SSD are less than the other functions because it is memory-bound as has been discussed.

We have optimized the C implementations of the SAD

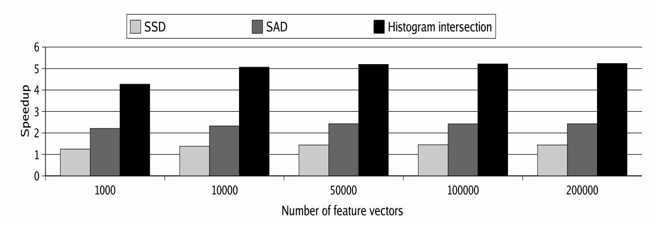

Figure 5. Speedup of the SIMD implementations of different similarity measurements over corresponding optimized scalar versions for different feature vectors that each of them has 512 data elements.

and histogram intersection functions. The conditional instructions have been replaced by unconditional instructions. The optimized C versions of the SAD and histogram intersection are 2.53 and 1.34 times faster than the corresponding unoptimized versions. Figure 5 depicts the speedups of the SIMD implementations of different similarity measurements over the corresponding optimized scalar versions. For the SAD function, the speedup ranges from 2.21 to 2.33, while the speedups for histogram intersection range from 4.27 to 5.24. The speedups of the SSD function are the same as the previous figure. As can be seen the speedups of the SAD and histogram intersection reduced compared to Figure 4.

In general, vectorized versions of the similarity measurement functions are memory intensive because they require two feature vectors: one for query image and the other for the target images.

7 Conclusions

We optimized and vectorized typical feature extraction algorithms, the mean and standard deviation, and some similarity measurement functions such as Sum-of-Squared-Differences (SSD), the Sum-of-Absolute Differences (SAD) and histogram intersection in this paper. Feature extraction and similarity measurement are two important functions in content-based image retrieval systems. We used the single-loop approach that has one pass to compute both mean and standard deviation features together. This technique yields a speedup of up to 1.85 over the straightforward implementation. The straightforward implementation has two passes, one to compute the mean and the other to compute the standard deviation. We vectorized the single-loop algorithm using the MMX and SSE2 extensions. The vectorized versions improve performance by a factor of up to 14.49. In addition, we vectorized the SSD, SAD, and histogram intersection functions. The vectorized versions provide a maximum speedup of 1.45, 2.33, and 5.24 for the SSD, SAD, and histogram intersection, respectively, over the optimized scalar implementations.

References

[1] J. H. Ahn, M. Erez, and W. J. Dally. Scatter-Add in Data Parallel Architectures. In *Proc. 11th Int. Symp. on High-Performance Computer Architecture*, pages 132–142, February 2005.

[2] Y. Chung and V. K. Prasanna. Parallelizing Image Feature Extraction on Coarse-grain Machines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(12):1389–1394, December 1998.

[3] R. Datta, J. Li, and J. Z. Wang. Content-based Image Retrieval: Approaches and Trends of the New Age. In *Proc. 7th ACM Int. Workshop on Multimedia Information Retrieval*, pages 253–262, November 2005.

[4] S. Deb. *Video Data Management and Information Retrieval*. IRM Press, 2005.

[5] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh. *Feature Extraction, Foundations and Applications*. Springer, 2006.

[6] Intel Corporation. *The IA-32 Intel Architecture Software Developer's Manual Volume 3 System Programming Guide*, 2004. Order Number: 253668.

[7] L. Kotoulas and I. Andreadis. Parallel Local Histogram Comparison Hardware Architecture for Content-based Image Retrieval. *Journal of Intelligent and Robotic Systems*, 39(3):333–343, March 2004.

[8] R. B. Lee. Subword Parallelism with MAX-2. *IEEE Micro*, pages 51–59, August 1996.

[9] A. Peleg, , and U. Weiser. MMX Technology Extension to the Intel Architecture. *IEEE Micro*, pages 42–50, August 1996.

[10] S. K. Raman, V. Pentkovski, and J. Keshava. Implementing Streaming SIMD Extensions on the Pentium 3 Processor. *IEEE Micro*, pages 47–57, July-August 2000.

[11] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Limitations of Special-Purpose Instructions for Similarity Measurements in Media SIMD Extensions. In *Proc. ACM Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, October 2006.

[12] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. SIMD Vectorization of Histogram Functions. In *Proc. 18th IEEE Int. Conf. on Application-Specific Systems Architectures and Processors (ASAP)*, July 2007.

[13] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Versatility of Extended Subwords and the Matrix Register File. *ACM Transactions on Architecture and Code Optimization (TACO)*, 5(1), May 2008.

[14] M. Stricker and M. Orengo. Similarity of Color Images. In *Proc. SPIE Storage and Retrieval for Image and Video Databases*, pages 381–392, February 1995.

[15] Z. Xiong and T. S. Huang. Subband-based, Memory-efficient JPEG2000 Images Indexing in Compressed-domain. In *Proc. IEEE Symp. on Image Analysis and Interpretation*, pages 290–294, April 2002.

[16] D. Zhang and G. Lu. Evaluation of Similarity Measurement for Image Retrieval. In *Proc. IEEE Int. Conf. on Neural Networks and Signal Processing*, pages 928–931, December 2003.