FPGA versus Subword Parallelism Implementations for a VQ Problem

Dr. Mahmoud Shuker Mahmoud
Al-Mansour University College

Dr. Laith Baker Salman
Al-Mansour University College

Abstract:

Vector Quantization (VQ) is a widely used algorithm in image data compression, voice compression, and more generally in signal processing. VQ is a generalization of scalar quantization and it is a codebook-based method. Unfortunately, designing a codebook that best represents the set of input vectors is an NP-hard problem. One of the successful solutions to this problem is to parallelize it. In recent years, high performance computing system have become more and more widespread, especially with the advent of highly flexible Field Programmable Gate Array (FPGA) and relatively cheap general purpose processors supported with SIMD instructions (MMX, SSE).

FPGAs are used in situation where the implemented algorithm is highly parallel. Arrays of processing units can be built in a single FPGA chip to perform the required process. The SIMD media ISA extensions for general-purpose processors has usually been to utilize Sub-word Level Parallelism (SLP) with existing hardware.

In this paper, two methods for parallelizing VQ are proposed. The first is a hardware-based parallelism using FPGA; and the second is a software-based parallelism using SIMD instructions. Finally, a comparison between the two proposed methods is obtained.

1- Vector Quantization

Vector Quantization (VQ) is a widely used algorithm in image data compression, voice compression, speech and hand written character recognition (in general statistical pattern recognition), and generally in signal processing. Vector Quantization (VQ) is a generalization of scalar quantization to the quantization of a vector. VQ is an NP-hard problem. VQ is usually used in signal processing in order to output a signal that is a compressed version of the original one. It is applied on problems where it is not necessary that the decompressed signal is an exact copy of the original, i.e. it is a lossy technique [1].

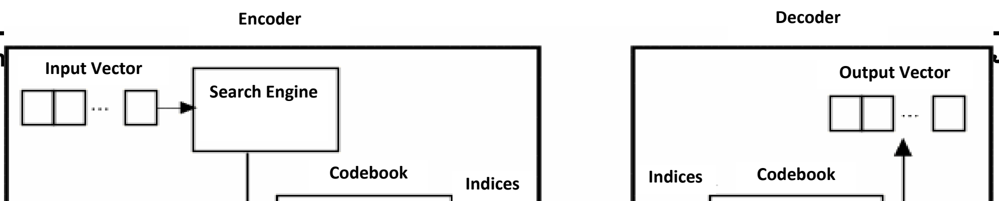
1.1- Background

A vector quantizer is composed of two basic operations. The first is the encoder, and the second is the decoder. Both encoder and decoder have a common *codebook*, which consists of a finite number of *codewords*. The encoder takes an input vector and outputs the index of the codeword that offers the best match (lowest distortion). In this case the lowest distortion can be found by different methods such as evaluating the Euclidean distance between the input vector and each codeword in the codebook as in equation (1) below:

$$d(v_1, v_2) = \sqrt{\sum_{i=1}^{k} (v_{1i} - v_{2i})^2} \qquad \text{.......(1)}$$

where v_1, and v_2 are k-dimensional vectors.

Once the closest codeword is found, the index of that codeword is sent through a channel (the channel could be computer storage, communications channel, and so on). When the decoder receives the index of the codeword, it replaces the index with the associated codeword. Figure (1) shows a block diagram of the operation of the encoder and decoder [1].



The dimension of the codewords and size of the codebook determine the compression ratio. If *N* is the codebook size then an index needs $\lceil \log_2 N \rceil$ bits for its binary representation. If *n* is the number of vectors and *p* the dimension, in bits, of each vector, then the compression ratio is given by:

$$\frac{n\lceil \log_2 N \rceil}{np} = \frac{\lceil \log_2 N \rceil}{p} \qquad \text{......(2)}$$

Beside VQ encoding/decoding process, the codebook generation is an important part of VQ technique. The codebook generation is usually done using a finite set of training vectors, which are samples of the original signal. This is done due to the fact that, in some cases, the signal is not completely known or it is too large to allow acceptable computational time.

VQ creates a codebook to minimize the distance between signal vectors (or training vectors) and codewords so that the difference, or distortion, between the original and the decompressed signal is also minimized. With a finite set of training vectors, the average distortion of a codebook is:

$$av\_distortion = \frac{\sum_{i=1}^{M} d(v_i, cw(v_i))}{M} \qquad \text{......(3)}$$

where *cw(v)* is the codeword that corresponds to v_i and *M* is the training set size. The function *d* is the Euclidean Distance.

There are two conditions that must be satisfied so that a VQ is optimal in the sense of minimizing the distortion:

- **The *Nearest Neighbor* Condition defines the optimal VQ encoding. Each vector must be mapped to the nearest codeword.**
- **The *Centroid* Condition requires that each codeword must be the centroid of the vectors that map to it.**

The most commonly used algorithm for VQ is driven by these two conditions and it's called the K-means Algorithm, also known as LBG algorithm (for Linde-Buzo-Gray). It is an iterative method based on the optimal codebook conditions. It has also been proved that the K-means algorithm converges in a finite number of iterations [1, 2].

1.2- SIMD Parallelization

Efficient exploitation of data parallelism on a larger scale has also been demonstrated architecturally in SIMD (Single Instruction Multiple Data) massively parallel architectures. SIMD computers were the first systems to be implemented with a massive amount of processors, and were among the first systems to provide computational power in the GFLOP range. However, because of their high cost, SIMD computers are more and more superseded by less expensive SIMD technologies available on most general-purpose processors [3, 4].

The basic idea is to use SIMD type of parallelism in a single processor to allow simultaneous processing of single instruction on several small registers, which actually are part of a single large register. This concept has been called SWAR (SIMD Within A Register). Most of modern processors have been designed with special instructions to improve this technique such as Intel's MMX, SSE, SSE2, and SSE3, Sun's VIS, HP's MAX, Compaq's MVI, MIPS's MDMX, Motorola's AltiVec, and AMD's 3DNow [3, 4].

1.3- VQ Sequential Implementation

This section outlines the sequential implementation of the codebook. For a codebook of size L; the K-means algorithm (where K=L) divides the set of training vectors {*v (n)*} into L clusters C_i in such a way that the two necessary conditions for optimality are satisfied.

Algorithm (1) illustrates the four basic steps for sequential implementation of the K-means algorithm. In this algorithm, *m* is the iteration index and C_i *(m)* is the ith cluster at iteration *m*, with cw_i *(m)* its centroid.

Step-1: **Initialization**

Set *m*=0.

Choose a set of initial code vectors to create initial codebook C_i. The initial codewords can be randomly chosen from the set of input vectors.

$cw_i(0),\ 1 \le i \le L$

Step-2: **Classification**

Given a codebook, C_m, classify the set of training vectors $\{v(n), 1 \le n \le M\}$ into the clusters C_i by the nearest neighbor rule.

$v \in C_i,\ \text{iff } d(v,cw_i) \le d(v,cw_j),\ \text{all } j \ne i,\ 1 \le j \le L$

This is done by taking each input vector and finding the Euclidean distance between it and each codeword. The input vector belongs to the cluster of the codeword that yields the minimum distance.

Step-3: **Code Vector Updating**

m=m+1.

Update the code vector of each cluster by computing the centroid of the corresponding training vectors in each cluster. This is done by obtaining the average of each cluster and will yields a new codebook C_{m+1}.

$cw_i(m)=Cent(C_i(m)),\ 1 \le i \le L$

Step-4: **Termination Test**

Compute the average distortion for C_{m+1}. If it has changed by a small amount since the last iteration (below a certain threshold) stop. Otherwise set *m=m+1* and go to the Classification Step.

Algorithm (1) Sequential K-means Algorithm

1.4- VQ Parallel Implementation

Unfortunately, designing a codebook that best represents the set of input vectors is NP-hard. That means that it requires an exhaustive search for the best possible codewords in space, and the search increases exponentially as the number of codewords increases; where for each iteration, it requires that each input vector be compared with all the codewords in the codebook. One of the successful solutions to this problem is to parallelize it using SIMD instructions available in most general purpose processors.

1.4.1- Subword-Level Parallelism

From the algorithm of the codebook design given in previous section; one can deduce that the most intensive computation part (hot spot), that can make use of Subword-Level Parallelism (SLP), is the classification step and more specifically, Euclidean distance calculation. The reason is that to satisfy the nearest neighbor condition, the distance between each input vector and all code vectors should be evaluated; and this process must be repeated for a number of iterations to generate the satisfactory codebook. Therefore parallelizing Euclidean distance calculation using SSE technique improves the performance of the parallel codebook design.

Figure (2) shows a single iteration Data Flow Graph (DFG) for the implementation of the Euclidean distance *d(v,cw)*, with SSE. After completing all iterations (vector size/4), the resulting four values stored at xmm_3 register should be accumulated to obtain a single final result, and this can be done using MOVUPS and SHUFPS SSE instruction. A simple sequential code was added to the SSE based function to overcome the case when the input vectors to the function are not multiples of four [4].

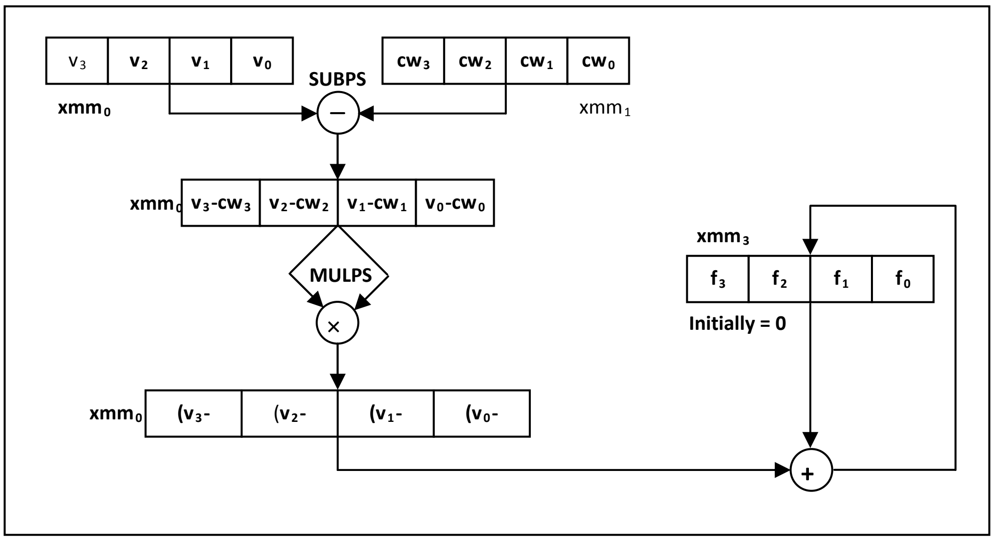


Figure (2) A Single Iteration of Euclidean Distance Calculation DFG

Beside the performance benefit obtained from parallelizing Euclidean distance calculation with SSE in the codebook design; the encoder of the VQ system can also make use of it by minimizing the time required for best matching of the input vectors.

1.4- SIMD Experimental Results and Analysis

The VQ system designed in this part of the paper was used for gray level image compression. The input images are divided into small blocks. Each block is considered as a training vector. Therefore, the total size of the training samples is equal to the size of the input image divided by the number of blocks. The sequential and parallel K-means algorithms for codebook design of the VQ system were tested for different image of sizes (512×512, 1024×1024 pixels), different codebook sizes (256, 1024 codewords) and different input vector size (2×2, 4×4 pixels).

Four cases are considered, these are:

- **Case 1: image size = 512×512, codebook size = 256 and codeword size = 2×2**
- **Case 2: image size = 512×512, codebook size = 1024 and codeword size = 2×2**
- **Case 3: image size = 1024×1024, codebook size = 256 and codeword size = 2×2**
- **Case 4: image size = 512×512, codebook size = 256, and codeword size = 4×4**

Finally, a 0.1 threshold value was used as an acceptable difference between current and previous average distortion to terminate codebook updating process.

Figure (3) shows the speedup gain in the encoding process when using SSE based function for Euclidean distance calculation for all the cases mentioned previously.

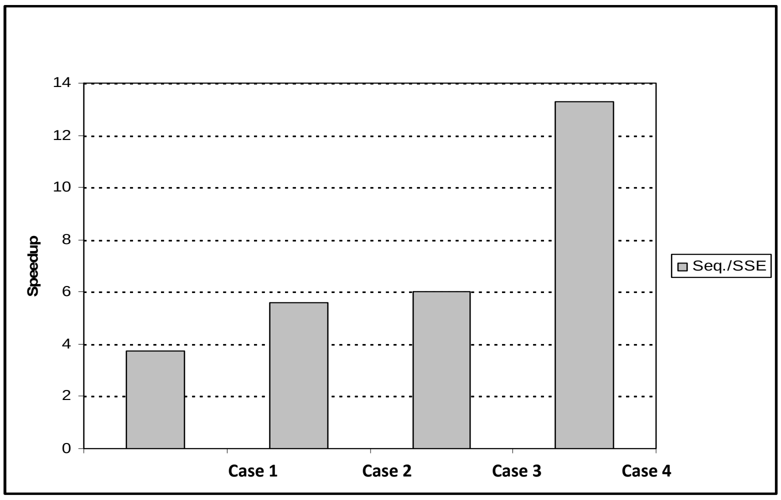


Figure (3) Speedup for VQ Encoding Based on SSE Routine

Figure (3) shows the effect of increasing codebook size (case 2), increasing image size (case 3) or increasing input vector size (case 4) on the performance of the SSE based distance function for encoding process. Also the speedup achieved by parallelizing (case 4) is more than parallelizing other cases. As said previously, this is because of the increase in the computations achieved by the SSE based function.

Finally, we should mention that using VQ for gray level image compression in the first three cases yields a Compression Factor (*CF*) of approximately 4, and a Peak Signal to Noise Ratio (*PSNR*) approximately 75. While for the last case, the *CF* equal to 18.084 and the *PSNR* are 61.634.

2- Hardware Implementation of the VQ Algorithm

The other alternative way of implementing the VQ algorithm is by building a hardware that is capable of implementing the algorithm in a very fast way. This is done using FPGAs. FPGAs are special ICs that can be programmed using special programming tools to perform a predetermined task.

This way of implementation will achieve high level of parallelism because one can build as many multipliers, adders, magnitude comparison units, or any other arithmetic or logical units as he want (provided that the chip has enough number of gates)[5, 6].

2.1- FPGAs Review

The word FPGA stands for (Field Programmable gate array). These are ICs that consist of raw AND-OR combinations, Flip Flops, Look up tables, and other logical basic units. The FPGA chip usually has an internal metal routes that can connect various logical input and output nodes inside the chip.

The main advantage of using FPGAs in implementing systems is the *reconfigurability* feature that means when the design is to be changed (whether this change is an error correction or a development in the structure of the design), the change can be done easily by software and the same chip will perform the new modified or corrected job. This is an important feature that reflects the power of FPGAs [5, 7].

2.2- FPGA Implementation of the VQ Algorithm

The block diagram in Figure (4) below was proposed to implement the VQ algorithm using FPGAs.

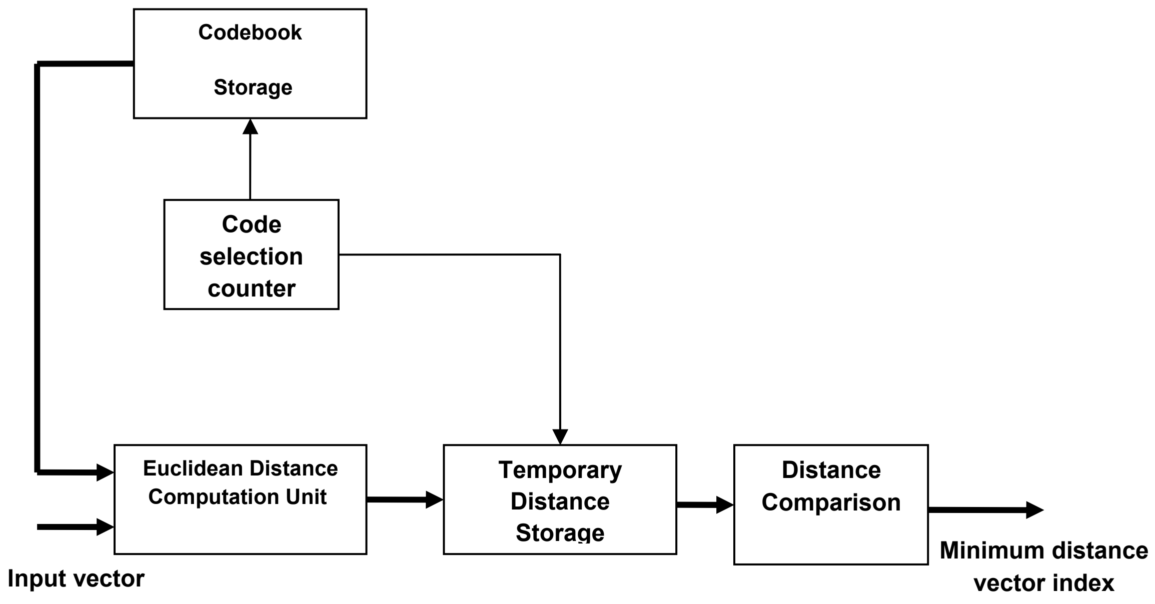

Codebook Storage
Code selection counter
Euclidean Distance Computation Unit
Temporary Distance Storage
Distance Comparison
Input vector
Minimum distance vector index

Figure (4) The Main Structure of the FPGA Implementation

The block diagram in figure (4) consists of the following main parts:

- Codebook storage.
- Euclidean Distance Computation Unit.
- Temporary Distance Storage.
- Distance Comparison.
- Code selection counter.

The Codebook storage stores the values of the centroid of the clusters and these values are compared sequentially to the input vector. The code selection counter selects the corresponding vector and the result of comparison (the distance between the input vector and the current codebook vector) is stored in the Temporary Distance Storage Unit. This unit holds the temporary values of the computed distances and prepares them for the comparison to find the minimum one. The comparison starts after all distances are computed and is performed by the Distance Comparison Unit which uses a tree-based smaller value decision making technique to find the minimum value.

It is important to emphasize that parallelism is exploited in the distance computation part of the algorithm (i.e. the input vector components are subtracted from the Codebook vector components in parallel and the differences are squared also in parallel) while the scanning of the Codebook is done sequentially. This is because building a number of Distance Computation Units equal to the Codebook size is not practical. However this *Subword* parallelism achieved good speedup as will be presented in the implementation results next.

2.2.1- Codebook Storage

The Codebook consists of 32 codes each one has 16 vector component (16 bytes). So as a hardware implementation it is a 32×128 bit memory. The indexing is made using the 5 bits coming from the Code selection counter and the counter is driven by the main clock operating with a maximum frequency to be determined from the implementation results. Figure (5) shows the arrangement of the Codebook.

2.2.2- Euclidean Distance Computation Unit

This unit consists of 16 subtractors followed by 16 squaring units each subtractor and its corresponding squarer work on one component of the input and code book vectors.

The outputs of all squaring units are added together and this will result in the magnitude square of the distance. However; since, comparing the squared magnitudes is similar to comparing the magnitudes themselves, there is no need to compute the square root of the output of the adder array. Figure (6) shows the internal diagram of the Euclidean Distance Computation Unit. The

output of this unit will be stored in a temporary storage array consists of 32 registers each representing the distance between the input vector and one of the code book vectors and the location of storage is controlled also by Code selection counter [8].

2.2.3- Distance Comparison Unit

This unit is responsible for comparing the stored distances between the input vector and the code book vectors. The comparison follows a tree-based technique in which each two values are compared by a magnitude comparator to find the smaller value.

The smaller value with its index are passed to the next level in the tree and so on until the smallest value is found. Figure (7) shows the Distance comparison unit.

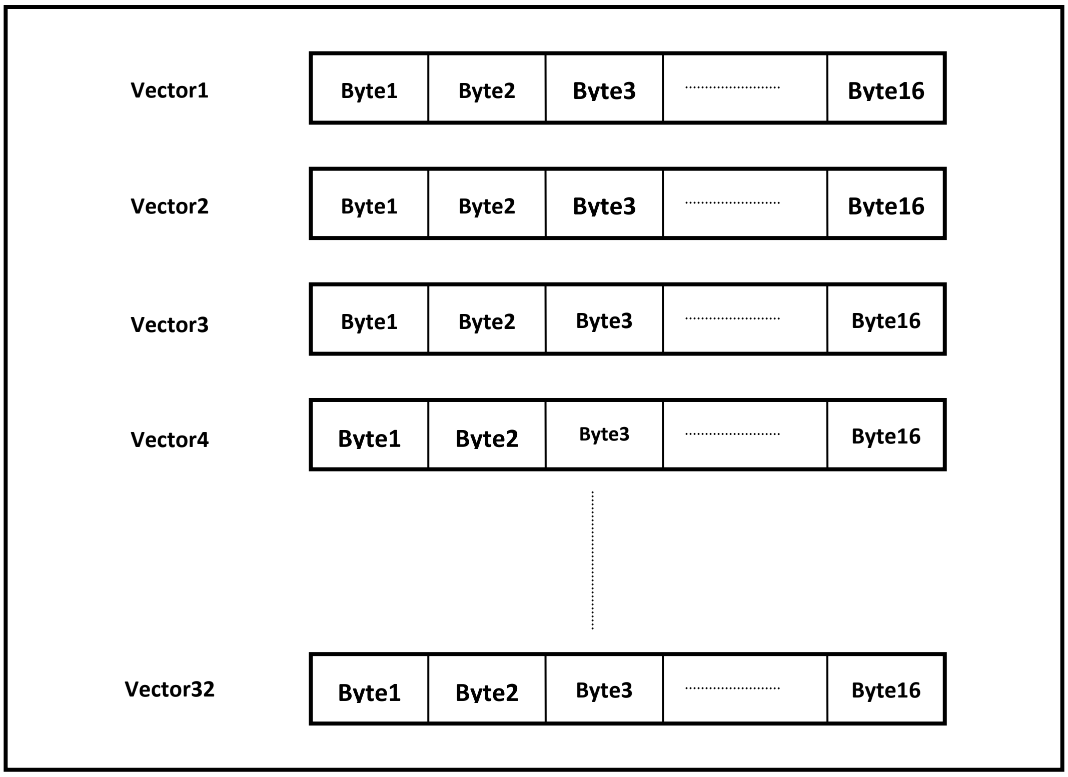


Figure (5) Code Book Storage Unit

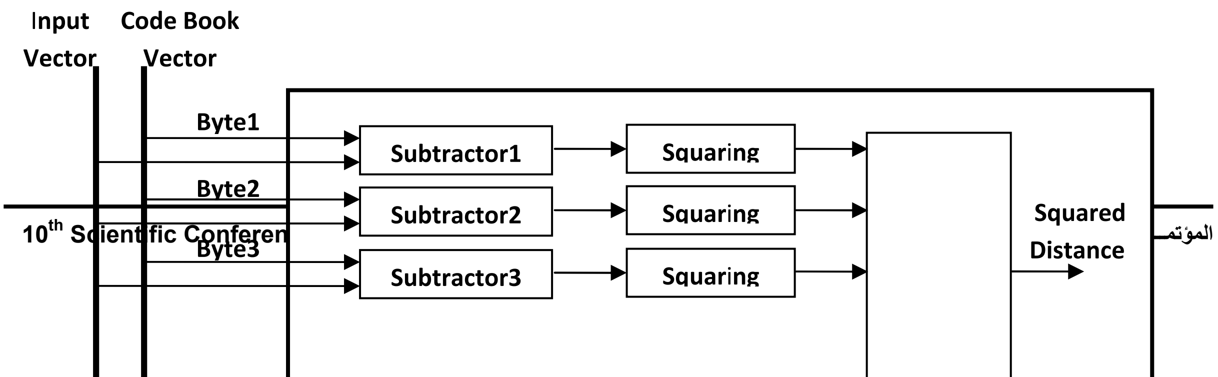


Figure (6) Euclidean Distance Computation Unit

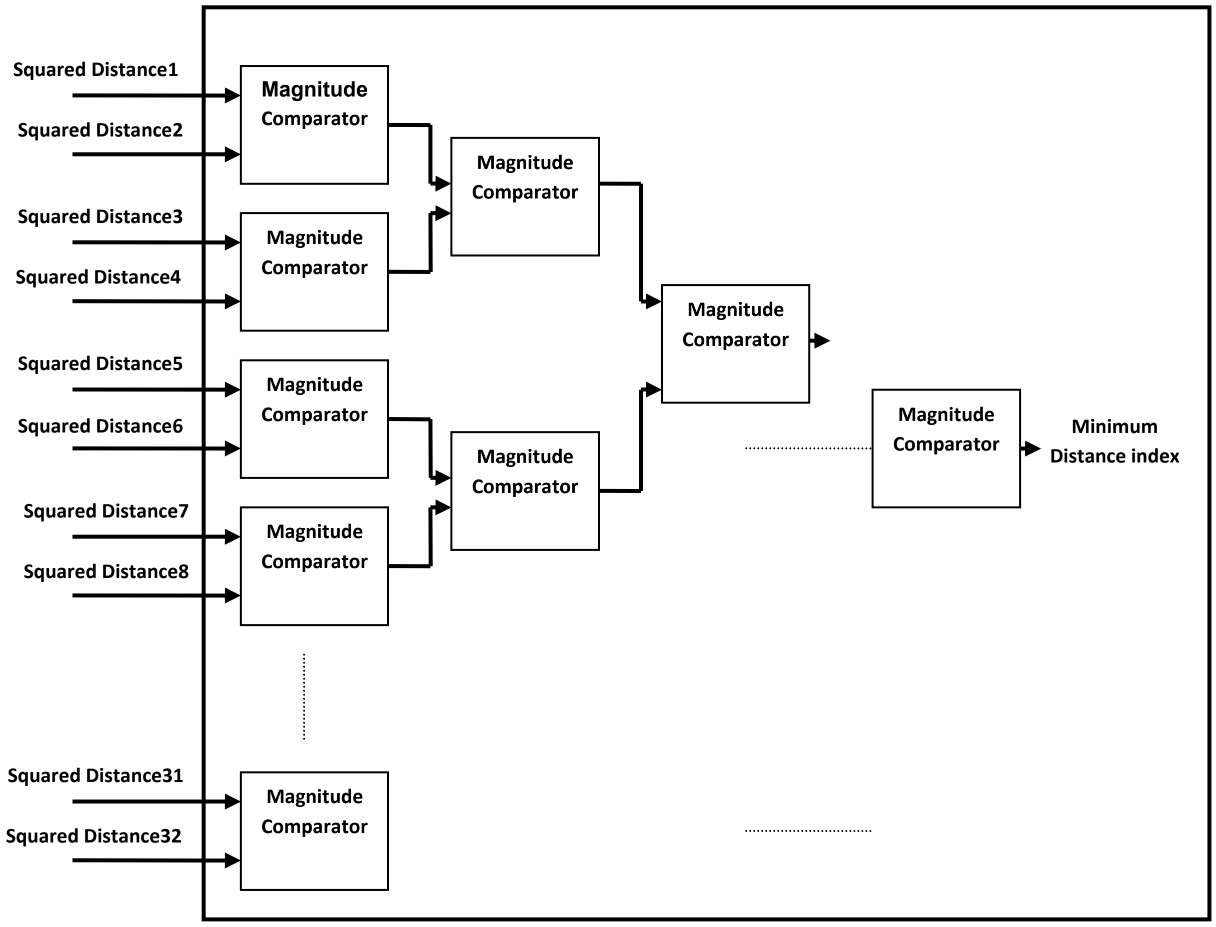


Figure (7) Distance Comparison Unit

2.3- Implementation Results

The block diagrams above were implemented using xillinx foundation 2.1i tools and the implementation was based on a schematic drawing of the system. The virtex family of chips were sufficient to fulfill the design size. The system was build from scratch (i.e. from basic logic gates in a hierarchical manner until the design desktop). The implementation results in a variety of reports describing how the system fit on the chip. The most important parameter obtained from these reports is the maximum combinational path delay which is about 160 ns meaning the system can compare 32 code book vectors to the input one in about 5 us and the total operation frequency is about 200 KHz.

3- Conclusions

VQ is a widely used algorithm in signal processing applications. Therefore parallelizing this algorithm will be very useful to get better speedup in comparison with serial implementation. In this paper two methods where used to parallelize the VQ algorithm.

The first one is a software based technique that makes use of the SIMD within a processor register (or what is called subword level parallelism). The Stream SIMD Extension (SSE) instructions available in most general purpose processors where used to parallelize the VQ algorithm. The implementation shows that the technique give better results (more speedup over sequential implementation) when there is a massive amount of computation; i.e. when the size of input data, codebook size, and/or the training vector size is large.

The second technique adopted in this paper for parallelizing the VQ is a hardware based technique. This technique uses the FPGA to implement the algorithm. The FPGA based implementation results in a hardware that is limited in its speed of operation by the maximum combinational path delay in the implementation structure. For the case of a (4X4) vectors, 32 entry codebook, the delay was found to be 160 ns so the whole codebook is scanned in 32X160 ns which equals 5.12 us. If the same parameters are used in a SIMD based algorithm, the result was found to be 134 us for the completion of the algorithm.

As a first conclusion the FPGA may seem the better solution to this kind of problems. However, by increasing the codebook size and the size of the data, the FPGA should process that increase in a sequential way because it is difficult to build larger arrays of multipliers and larger storage units. In the case of SIMD implementation more parallelism is possible and a better speed up will be achieved. So the suitable choice depends on the amount of data and the VQ selected parameters.

References

1. "Vector Quantization", Available at: http://www.mqasem.net/vectorquantization/vq.html.

2. D. Salomon, "Data Compression: The Complete Reference", Springer, Inc. Publisher, 2nd Edition, 2000.

3. L. Baumstark, J. Wills, "Exposing Data-Level Parallelism in Sequential Image Processing Algorithms", IEEE Proc. of the 9th Work Conf. on Reverse Engineering, 2002.

4. "IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture", Intel Corporation, Order No. 253665-015, April 2005.

5. "FPGA and CPLD Architectures" IEEE DESIGN & TEST OF COMPUTERS.

6. Zainalabedin Navabi, Analysis and Modeling of Digital Systems, McGraw-Hill, Inc, 1993.

7. http://www.xilinx.com/support/documentation/user_guides/ug073.pdf.

تنفيذ خوارزمية تكميم المتجهات بطريقة مصفوفة البوابات المنطقية القابلة للبرمجة مقابل طريقة توازي الكلمة الجزئية

د. ليث باقر سلمان
كلية المنصور الجامعة

د. محمود شكر محمود
كلية المنصور الجامعة

المستخلص :

تستخدم خوارزمية تكميم المتجهات بشكل واسع في ضغط بيانات الصور و بيانات الصوت و بشكل اعم في معالجة الإشارات. إن تكميم المتجهات هو تعميم لتكميم الكمية العددية و هي طريقة تعتمد على مبدأ سجل الرموز. رغم ذلك فان تصميم سجل رموز يمثل مجموعة المتجهات المدخلة هو عملية معقدة. و أحد الحلول الناجحة لهذه المشكلة هو من خلال أسلوب العمل المتوازي. في السنوات الأخيرة أصبحت أنظمة الحساب عالية الكفاءة اكثر انتشارا خاصة بوجود مصفوفة البوابات المنطقية القابلة للبرمجة عالية المرونة و بوجود المعالجات العامة الاستخدام المعززة بايعازات SIMD (MMX, SSE).

تستخدم مصفوفة البوابات المنطقية القابلة للبرمجة في الحالات التي تكون فيها الخوارزمية ذات طبيعة متوازية. حيث يمكن بناء مجموعة من وحدات المعالجة المتشابهة في الشريحة الواحدة لغرض القيام بالمعالجة المطلوبة. ان امتداد ISA الخاص بايعازات SIMD للمعالجات العامة الغرض تم تصميمه لاستثمار القدرة على العمل على التوازي SLP مع المكون المادي الموجود.

في هذا البحث تم اقتراح طريقتين لتنفيذ خوارزمية تكميم المتجهات. الطريقة الأولى هي طريقة مبنية على أساس المكون المادي لمصفوفة البوابات المنطقية القابلة للبرمجة و الطريقة الثانية مبنية على أساس برنامج يستخدم ايعازات SIMD وتم إجراء مقارنة بين الطريقتين.