High-Speed Codebook Design by the Multipath
Competitive Learning on a Systolic Memory Architecture

Kentaro Sano † Chiaki Takagi Kenichi Suzuki Tadao Nakamura
Graduate School of Information Sciences, Tohoku University
01 Aramaki Aza Aoba, Aoba-ku, Sendai 980-8579, JAPAN

Abstract: - So far, we have proposed a dedicated processor based on a systolic memory architecture to accelerate competitive learning (CL) for optimal codebook design. While this processor achieves high-speed codebook design, it has essential problems: limitation of hardware resources and lack in flexibility of codebook size. To solve these problems, we present a multipath CL algorithm that allows the processor to flexibly handle the various size of a codebook. The multipath CL on the FPGA-based prototype of the processor could design codebooks with comparable MSEs to the software simulation while the prototype processor achieved 2000 times faster processing speed than a general-purpose processor.

Key-Words: - Vector Quantization, Codebook Design, Competitive Learning, Systolic Memory Architecture

1 Introduction

Vector quantization (VQ) [1] is one of the effectual techniques for lossy data compression. The quantization process of VQ essentially causes distortion from the original data, and therefore the optimal codebook is required to minimize the average distortion. Competitive learning (CL) algorithms including Kohonen CL algorithm [2] have been applied to optimal codebook design. However, the long computation time for their repetitive competition process has limited practical use of optimal codebook design especially for large-scale data.

To solve this problem, we have proposed a dedicated processor based on a systolic memory architecture [3] for codebook design. This architecture provides a high-speed platform for MMPDCL (minimax partial distortion competitive learning) algorithm [4] that is an improved CL algorithm to avoid the under-utilization problem of Kohonen CL algorithm. This architecture is a combination of a systolic array for nearest-neighbor search [5] [6], a functional memory on the systolic array [7], and a special structure for MMPDCL execution. We have also introduced deferred update of a codebook to exploit as much parallelism as possible on the systolic array.

However, the proposed processor still has essential problems: limitation of hardware resources and lack in flexibility of codebook size. The fixed size of the systolic array cannot design a codebook with too many codewords. This paper presents a multipath CL algorithm that allows the fixed systolic array to flexibly handle a codebook with a larger number of codewords. Within the multipath CL algorithm, sub-codebooks can sequentially be designed by the fixed systolic array for partitioned input-vector subspaces. We implemented an FPGA-based prototype system of our proposed processor. Comparing with simulation results, we will evaluate its processing speed and quality of codebooks designed by the multipath competitive learning on the prototype system.

This paper is organized as follows. Section 2 briefly reviews MMPDCL algorithm for codebook design. Section 3 describes the multipath CL algorithm applied to our processor based on the systolic memory architecture. Section 4 discusses performance of the multipath codebook design performed by the implemented prototype processor, comparing with software simulation. Finally, Section 5 gives conclusions and future work.

2 Competitive Learning Algorithms for Codebook Design

2.1 Codebook design for VQ

A vector quantizer maps vectors in \( k \)-dimensional Euclidean space \( \mathbb{R}^k \) into a finite number of representative vectors in \( \mathbb{R}^k \). Such representative vectors are referred to as codewords, and a set of codewords is referred to as a codebook. A vector quantizer is specified by a codebook \( Y = \{ y_1, ..., y_N \} \). A set of input vectors is quantized by replacing input vectors \( x \) with appropriate \( y \). This replacement causes distortion \( d(x, y) \), which is commonly defined as \( \| x - y \| \). Given a codebook, quantization of
each input vector is performed so that distortion is minimized: $Q(x) = y_i$, whose distortion is minimum for $x$. Let $p(x)$ be probability distribution of $x$. The mean squared error (MSE) between input vectors and their corresponding codewords is defined as:

$$MSE = \sum_{i=1}^{N} \int_{S_i} d^2(x, y_i)p(x)dx = \sum_{i=1}^{N} D_i \quad (1)$$

where $D_i$ is the contribution of $y_i$ to MSE, called partial distortion.

To obtain an optimal vector quantizer, we have to find such codewords $Y$ that minimizes MSE. The partial distortion theorem [1] denotes that each partial distortion makes an equal contribution to the minimized MSE with sufficiently large $N$. Usually we have to heuristically solve this minimization problem by a learning procedure with a given set of input vectors, because no analytical solution is generally available for most of applications where $p(x)$ is unknown.

2.2 MMPDCL algorithm for codebook design

So far, various CL algorithms have been proposed to design optimal codebooks by solving the minimization problem of an MSE. Kohonen CL algorithm [2], which is a basic and representative CL algorithm, repeats the following learning-steps: Prior to learning, codebook $Y$ is initialized with some distribution of vectors. Then, for input vector $x_k$, such codeword $y_i*$ is found as a winner that $d^2(x_k, y_i*)$ is minimum. The process to find a winner is called competition. Next, winner $y_i*$ is moved toward the input vector:

$$y_i* := y_i* + \alpha(x_k - y_i*) \quad (2)$$

where $\alpha$ is a learning rate, which is a positive constant for codebook adaptability to inputs.

As the learning steps proceed, codewords are approximating the distribution of input vectors. However, Kohonen CL algorithm has an inherent problem, i.e., the underutilization problem. Inadequate distribution of initial codewords falls into local optimality having some codewords extremely distant from any inputs, which cannot be utilized for quantization. MMPDCL algorithm [4] is an improved CL algorithm that was designed to avoid this problem by introducing the equidistortion principle.

Instead of $d^2(x_k, y_i*)$, MMPDCL algorithm uses the following distortion biased with an online partial distortion:

$$d^2_{pd}(x, y_i) = pd_i(t - 1)e^{-m/T} + \|x - y_i\|^2 \quad (3)$$

where $t$ denotes a time that is incremented every input, and $m$ is the sweep index. Term $pd_i(t)$ is an online partial distortion at time $t$. A sweep is one full learning sequence for the whole set of input vectors. $T(>0)$ is a constant to heuristically be determined as 10% of the total sweeps. The exponential term, called an attenuation term, attenuates the effects of partial distortions as learning proceeds. The online partial distortion term of Eq.(3) prevents codewords with a large partial distortion from winning competition, and indirectly evens out partial distortions. Consequently, this equidistortion mechanism removes underutilized codewords that have a small partial distortion.

After winner $y_i*$ is selected by competition, it is updated according to Eq.(2). We empirically found that a constant learning rate is also effective, while the original MMPDCL algorithm adopts a variable rate [4]. After the winner is moved, its online partial distortion is updated as:

$$pd_i*(t) = pd_i*(t - 1) + \|x_k - y_i*\|^2. \quad (4)$$

3 Multipath Competitive Learning on Systolic Memory Architecture

3.1 MMPDCL Processor based on systolic memory architecture

We have proposed and developed the MMPDCL processor for high-speed codebook design [3]. This processor is based on a systolic memory architecture that has the following advantages: parallel processing on an array of processing elements (PEs), wide memory-bandwidth of PEs coupled with their local memory, and the structure without centralized resources. PEs with local memory essentially match exploiting data parallelism of competitive learning. This architecture also mitigates a bottleneck of data supply from external memories by re-use of input data passing through the array.

Fig.1 shows an overview of the MMPDCL processor. This processor consists of three components: a codebook memory, partial distortion (pd) calculation units and comparators. The codebook memory is a systolic array of functional memory cells, called codeword cells. Each codeword cell has simple logic circuits so that each column computes distortion $d^2_{pd}$ between a codeword and an input vector. The computed distortion is used by the pd calculation units to compute $d^2_M$. Finally, the codeword with the minimum $d^2_M$ is selected by the array of comparators.

3.1.1 Codeword cells

Each codeword cell stores an element of a codeword, and each column of the array corresponds to a codeword. Let $m$ denote dimension of a codeword. Each column contains $m$ codeword cells. The number of columns, $n$, is equal to the number of codewords, i.e., the size of a codebook. Fig.1 is an example where $m = 4$ and $n = 4$. The elements of input vectors are input to the array in the skewed fashion...
The update control in Fig.2 controls writing register $y_{kj}$ for codeword update based on Eq.(2). The winner ID output by the most right comparator is returned to the most upper-left codeword cell, and propagated to the successive cells in both row and column directions. When the update control circuit receives a winner ID that matches the codeword ID of the cell, register $y_{kj}$ is written with $y_{ij} + \alpha (x_{(k-d)j} - y_{ij})$. This update is carried out for an input vector $(n + m + 1)$ cycles after its first element came to the column of the updated codeword. We referred to this as the deferred update, where the successive $(n + m)$ input vectors are input to the array and processed until the deferred update begins. To supply input vectors $x_{(k-d)j}$ for the deferred update, we put a shift buffer with $d = (n + m + 1)$ entries at the beginning of each delayed propagation line of an input-vector element.

3.1.2 Partial distortion calculation units

Each column is connected to a $pd$ calculation unit. Fig.3 shows a block diagram of the $pd$ calculation unit. This $pd$ calculation unit stores an online partial distortion, $pd_i$, of each codeword. A distortion between a codeword and an input vector is input as $d_k^2$. Then biased distortion $d_{M2i}$ is computed by adding $d_k^2$ to $pd_i$ after $pd_i$ is attenuated by bit-shifting [3]. The shift amount, $shamt$, is a counter to approximate the exponential attenuation term in Eq.(3) with bit-shifting of $pd_i$ in a fixed-point format. Finally, $d_{M2i}$ and $d_i^2$ are output to a comparator.

Based on Eq.(4), the online partial distortion, $pd_i$, is updated by using the winner ID and its $d_k^2$ sent from the most right comparator. When a $pd$ calculation unit receives the winner ID that is the same as the unit’s ID, $pd_i$ is updated by using $d_k^2$. Since $d_k^2$ is the squared distortion of the winner that is not moved yet, it should be multiplied by $(1 - \alpha)^2$.

3.1.3 Comparators

Comparators are connected forming a 1D array to search the winner with minimum $d_{M}^2$. The comparator has six inputs; a codeword ID, original distortion $d_k^2$ and biased distortion $d_{M2i}$ from the corresponding $pd$ calculation unit, and those from the previous comparator. Each comparator compares $d_{M2i}$ of its column with the output of the previous comparator. If $d_{M2i}$ of the column is smaller, its ID, $d_k^2$ and $d_{M2i}$ of the column are output to the next comparator. If not, those of the previous comparator are output to the next.
Thus, the final comparator outputs the ID and \( d^2 \) of the winner.

### 3.2 Multipath competitive learning

The systolic memory architecture has problems: limitation of hardware resources and lack in flexibility of codebook size. The codebook memory has to be an array with the same number of columns as codebook size. We cannot implement too big array if hardware resources are limited. In addition, fixed implementation of the systolic array cannot be utilized for different size of a codebook. To solve these problems, we introduce multipath competitive learning based on the space partitioning method.

The concept of multipath competitive learning is very simple: partition the input-vector space into subspaces, and then independently perform competition in each subspace to design subsets of a codebook. Fig.4 shows the flow chart of the multipath CL algorithm, which consists of the four stages: \textit{input-vector space partitioning}, \textit{codebook initialization}, \textit{competition for one input} and \textit{codebook update}.

Within the first two stages, the input-vector space is partitioned into subspaces, and codewords are distributed to sub-codebooks of the subspaces. Then, MMPDCL algorithm is performed independently for a sub-codebook in each subspace. After all the sub-codebooks are obtained, they are merged into the final codebook.

For input-vector space partitioning, we utilize the two-level competitive learning method shown in Fig.5. Here, let \( n_i \), \( n_c \) and \( n_p \) denote the number of input vectors, the number of codewords and the number of subspaces, respectively. In the two-level competitive learning method, the \( n_i \) input vectors are firstly partitioned by codebook design with \( n_p \) codewords. This is done by the 1st-level competitive learning. Then we obtain \( n_p \) Voronoi regions of the codewords that partition the input vectors into \( n_p \) subsets. For each subset of input vectors, we independently design a sub-codebook with \( n_c/n_p \) codewords. We refer to this sub-codebook design as the 2nd-level competitive learning.

After all the sub-codebooks are designed, we merge them into the final codebook with \( n_c \) codewords. Thus the multipath competitive learning allows a small systolic array to design a codebook with a larger number of codewords.

### 4 Performance Evaluation

To evaluate the performance of the proposed MMPDCL processor, we have implemented its prototype system. In the following subsections, we will show the processing speed of the prototype system and MSEs of codebooks designed by the multipath competitive learning on the prototype system, comparing with simulation results.

#### 4.1 FPGA-based prototype system

We implemented the prototype system of the proposed MMPDCL processor on an FPGA-based prototyping platform, the DiNi Group DN5000K10S. This platform is a PCI-connectable board having an FPGA, ALTERA Stratix EP1S80F1508C7 with 79040 logic elements (LEs) that can emulate up to 600,000 gates. Stratix EP1S80 also contains nine 64KB SRAMs with 32bit-wide read/write ports.

Fig.6 depicts the overview of the prototype system, which was implemented as a PCI target device of a host PC. The prototype system consists of a PCI controller, sixteen input-vector memories of SRAMs, a systolic memory array of \( 16 \times 16 \) codeword cells, a 1D array of six-
teen pd calc. units, a 1D array of sixteen comparators, a control logic and a control register. The sixteen input-vector memories are virtually implemented with the eight SRAMs, i.e., an SRAM with a 32bit-wide port corresponds to two input-vector memories with 16bit-wide ports. These implemented units took 78687 LEs, more than 99% LEs of the FPGA chip.

The input-vector memories and the control register are mapped to the memory address space of the PCI target device, and able to be read and written by the host PC. The input-vector memories are also connected to the systolic memory array. The read data from the input-vector memories can be input to the rows of the array, and the outputs from the rows can be written to the input-vector memories. The control logic controls the systolic memory array and the input-vector memories based on the values of the control register.

The systolic memory array has four operation modes; idle mode, codeword write mode, codeword read mode and competitive learning mode. The array is in the idle mode when the system is booted up. In the codeword write mode, we can write the codeword registers with values read from the input-vector memory. In this mode, appropriate values are firstly set to the input vector registers of the codeword cells by shifting the values read from the input-vector memories. Then the values are simultaneously written to the corresponding codeword registers. In the codeword read mode, the codeword registers can be read to the input-vector memories by firstly reading their values to the input vector registers and shifting them. In the competitive learning mode, input vectors are read from the input-vector memories, and input to the systolic memory array while all the codeword cells perform distance calculation and codeword update.

The whole prototype system could operate at 33MHz that is the same frequency as PCI bus. The implemented array of $16 \times 16$ codeword cells can design a codebook with 16 codewords. The input vectors and codewords are expressed in a fixed point format of 8 bits-integer and 4 bits-decimal. The cumulative distortion, $d_2$, and the biased distortion, $d_2^B$, are expressed in 18 bits-integer and 4 bits-decimal, and 24 bits-integer and 4 bits-decimal, respectively. We empirically used a constant learning rate of 0.125 that can be implemented as 3-bit right shift of a fixed point value.

### 4.2 Software simulation

For performance comparison, we also conducted the following software simulation:

1. Single-path MMPDCL in floating point
2. Multipath deferred MMPDCL in fixed point.

Simulation 1 designs a codebook by the normal single-path MMPDCL algorithm. Simulation 2 designs a codebook by the multipath MMPDCL algorithm using deferred update and a constant learning rate, which corresponds to software simulation of the multipath codebook design on the MMPDCL processor. For simulation 2, 0.125 is used for a constant learning rate. As described in Section 3.1.1, the number of delay cycles for the deferred update was set $(n + m + 1)$ where $n = 16$ and $m = 16$ to emulate the implemented array.
By designing codebooks for VQ-based image compression, we compare performance of the prototype system with the software simulation. Fig.7 shows a test image, Lena, which is a 8-bit gray scale image with a resolution of $512 \times 512$ pixels. This image was divided into $4 \times 4$-pixel blocks forming 16384 16-dimensional input vectors. We used a codebook with 256 codewords to give a compression rate of about 0.078 (original image:256KB, codebook:4KB, block indices for compressed image:16KB). Since the systolic memory array of the prototype system has only 16 columns, we applied the multipath competitive learning to the system by partitioning the codebook into 16 sub-codebooks with 16 codewords. The multipath MMPDCL of simulation 2 was also performed with the same partitioning. We performed 128 sweeps of competition for all the experiments.

4.3 Performance comparison

4.3.1 Processing speed

To design a codebook with 256 codewords, the multipath competitive learning with 16 partitioned sub-codebooks requires 128 sweeps of 16384 input vectors for the 1st-level competitive learning. Furthermore, 128 sweeps of 16384 input vectors is also required for the 2nd-level competitive learning because 16384 input vectors are totally used to design the 16 sub-codebooks. Since the systolic memory array with $16 \times 16$ codeword cells performs competition for an input vector every clock cycle, $4.19 \times 10^5 (= 128 \times 16384 \times 2)$ cycles are required to finish the multipath competitive learning on the array. Accordingly, the prototype system running at 33MHz takes 0.127 sec to design the codebook by the multipath competitive learning.

On the other hand, the single-path MMPDCL algorithm using floating point calculation took $2.66 \times 10^2$ sec to design a codebook on Intel Celeron processor running at 2.0 GHz. Although there is difference between floating-point calculation and fixed-point calculation, the multipath competitive learning on the prototype system achieved about 2000 times faster codebook design in comparison with the general-purpose processor. This high-speed processing was due to the massive parallelism and regularity of the systolic memory array, and wide band-width of local memory access inside the array. The low operation frequency and the structure without centralized resources are expected to give performance scalable to the increasing number of transistors available in deep submicron CMOS technology generations.

4.3.2 Codebook quality

Fig.8 shows MSEs for the single-path MMPDCL (simulation 1), the multipath deferred MMPDCL (simulation 2) and the multipath MMPDCL on the prototype system. The MSE of the codebook designed by the single-path MMPDCL was reduced to about 618 as the number of sweeps increases. The multipath deferred MMPDCL finally gave an MSE of about 679, which is slightly larger than that by the single-path MMPDCL. The difference in MSE is mainly due to the fixed-point calculation, a constant learning rate, deferred update, and multipath design with sub-codebooks.

On the other hand, the multipath MMPDCL on the prototype system resulted in an MSE of about 751. Although this value is also different from the MSE by the multipath deferred MMPDCL, we made sure that the prototype system could design codebooks with comparable MSEs. The difference in MSE is probably caused by different precision of the fixed-point calculation between the prototype system and the software simulation. Simulation 2 does not completely emulate the data-path of the implemented hardware. Our future work contains improving the precision of
a. Single-path MMPDCL.

b. Multipath deferred MMPDCL.

c. Prototype system (multipath).

Figure 9: Reconstructed images.

the fixed-point calculation on the array.

Fig.9 shows reconstructed images with the designed codebooks. Regardless of the MSE increase, the prototype system gave the reconstructed image with comparable quality to the other images. On this image, perceptibly severe degradation is not seen in comparison with the image obtained with the single-path MMPDCL (simulation 1). Although the prototype system can be improved in terms of precision, the obtained results show that the implemented system is also able to design codebooks applicable to VQ-based image compression applications.

5 Conclusions

In this paper, we have presented the multipath competitive learning to allow the fixed systolic array of the MMPDCL processor to design a codebook with a various number of codewords. The FPGA-based prototype system of the MMPDCL processor achieved 2000 times faster codebook design than a general-purpose processor. Comparison with the software simulation has shown that the multipath MMPDCL on the prototype system could design codebooks with comparable MSEs to the simulation results of the multipath deferred MMPDCL. The codebook designed by the prototype system has good quality applicable to VQ-based image compression applications. Our future work includes detailed evaluation with various types of input data, improvement of precision and extension of the systolic memory array to design larger codebooks.

Acknowledgments

This research was partially supported by Grant-in-Aid for Young Scientists(B) KAKENHI(#15700040).

6 References