

# A SIMD Neural Network Processor for Image Processing

Dongsun Kim<sup>1</sup>, Hyunsik Kim<sup>1</sup>, Hongsik Kim<sup>2</sup>,  
Gunhee Han<sup>2</sup>, and Duckjin Chung<sup>3</sup>

<sup>1</sup> DMB Project Office, Korea Electronics Technology Institute  
455-6 MaSan-Ri, JinWi-Myon, PyungTaek-Si, KyungGi-Do 451-865, Korea  
{dskim,hskim}@keti.re.kr

<sup>2</sup> Department of Electrical & Electronic Engineering, Yonsei University, Seoul, Korea  
{hskim,gunhee}@yonsei.ac.kr

<sup>3</sup> Information Technology and Telecommunications, Inha University  
253 Younghyun-Dong, Nam-Gu, Incheon 402-751, Korea  
djchung@inha.ac.kr

**Abstract.** Artificial Neural Networks (ANNs) and image processing requires massively parallel computation of simple operator accompanied by heavy memory access. Thus, this type of operators naturally maps onto Single Instruction Multiple Data (SIMD) stream parallel processing with distributed memory. This paper proposes a high performance neural network processor whose function can be changed by programming. The proposed processor is based on the SIMD architecture that is optimized for neural network and image processing. The proposed processor supports 24 instructions, and consists of 16 Processing Units (PUs) per chip. Each PU includes 24-bit 2K-word Local Memory (LM) and a Processing Element (PE). The proposed architecture allows multi-chip expansion that minimizes chip-to-chip communication bottleneck. The proposed processor is verified with FPGA implementation and the functionality is verified with character recognition application.

## 1 Introduction

Massively parallel computation of simple operator in neural networks and image processing suggest analog implementation as an attractive choice. Analog implementation of neural-network processor has advantages in low-power and small silicon area [1,2]. Although many analog implementations have been reported and commercialized as well, the reliability of analog computation is severely degraded as the array size increases due to component mismatch. The digital implementations of neural network processor have been reported in 90s [4,5]. This approach requires large silicon area while application range of the specialized neural network processor is limited. Drastic improvement of general purpose DSP performance and price makes the DSP based implementation more attractive than neural network processor development. Late 90s and early 2000, digital neural processor development is revived because the fabrication technology and

CAD tool advance allow the implementation of a complex system on a chip in relatively low cost and several commercial products are available. However their application range is still limited. Data intensive operators in neural networks and image processing need high computing power and have a great potential for SIMD parallel processing. SIMD architectures have been adopted effectively on the applications of image processing, matrix operations, partial differential equations, artificial neural networks, multimedia processing, etc. Previously, SIMD architecture was realized in the form of massively parallel computer system starting from the first SIMD machine project, ILLIAC IV. They usually consisted of high performance host computer and data parallel unit including few hundreds or thousands of simple processing elements, memory system, and a global array control unit [2,3]. To overcome the limits of the previous SIMD processors, this paper proposes a flexible SIMD Processor with distributed memory taking full advantage of the application specific instruction set and hardware resources; Address Modifier (AM), Non-linear Functional Unit (NFU), ring and global buses, and multi-chip expansion. These features are optimally customized for achieving high computational efficiency in data intensive applications while providing flexibility.

## 2 Processor Architecture

The proposed processor employs an SIMD architecture consisting of 16 processing units (PUs), a non-linear functional unit (NFU), and a control unit (CU), which are connected through two global data buses, one control bus, and a ring bus as shown in Figure 1. The instruction program is stored in the embedded program memory; on the other hand, the data are distributed in embedded local memories (LMs) and external data memory. The global data bus and ring bus allow data broadcast and PU-to-PU data transfer. The CU generates the control signals for all PUs and allows address jump and branch functions. The NFU is a look up table memory that realizes an arbitrary non-linear function. Global Register File (GRF) is used to store data from NFU. The data in GRF are to be broadcasted to PUs through the data bus or the ring bus. Each PU consists of 32-bit fixed point numerical arithmetic units, a 32-bit 16-word register file, 16-bit 1.5K-word LM, special purpose registers (CR, FR, and AR), and an address modifier (AM). In addition, an 16-bit logical arithmetic unit (LALU) is embedded for basic logical operations (AND, OR, XOR, and NOT). For MAC operation, the result of multiplier is bypassed to adder. The adder has the ability to perform the local memory addressing by adding the offset value stored in the RF0 register and the address field of WLD (or WST) instruction. The embedded LM is used to store weights, coefficients, image, and other data according to the applications. Followings are key features of the proposed processor.

### 2.1 Address Modifier (AM)

Particularly, each PU contains an AM which enables the proposed processor to have functionalities of both column-wise data fetch and row-wise data fetch.

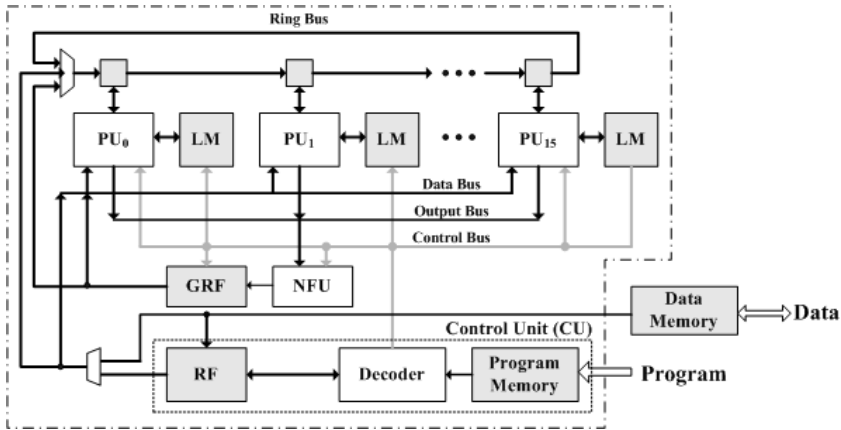


Fig. 1. Block diagram of the proposed architecture

The importance to do so is that many linear algebra applications require series of matrix-by-vector and transposed matrix-by-vector multiplications. In ANNs, the matrix contains the synaptic weights and the vector does input values or error values. The matrix element accessing direction is dependent on the Processing state. Figure 2 shows an operational model of how an AM works on multi-layer perception (MLP) with back-propagation (BP). Here, a row of the forward weight matrix is allocated to each PU. The first is feed-forward (FF) operation, in which the network computes the equation,  $u_i = \sum_j^b s_j \omega_y$ . The second is error BP that computes the equation,  $e_j = \sum_i^m \delta_i \omega_y$ . From these two equations, the weights distributed over LMs should be accessed in two different modes; the row-wise for the (FF) and the column-wise for the error BP. In the proposed processor, three mechanisms, ring, bus, and AM, are used for effective memory access for BP. For the process of FF operation, the address is broadcast to all PUs simultaneously through bus as shown on Figure 2 (a) since the weights are stored in local memories in row order. In error BP phase, the AM calculates a new address using modular operation for column base memory access. Previous error values are shifted to next PU through ring register as shown in Figure 2 (b). Therefore, the proposed architecture enables both row and column wise memory access without many overheads.

## 2.2 Multi-chip Expansion

The expandability of the proposed processor is essential because most scientific computations require large sizes of parallel processing. The multi-chip expansion through a register ring is used for increasing the network size. This is called as multi-chip ring shift operation mode which is decided by the flag attached to instructions. In this mode, the ring ready register is set to ‘1’ and the program is stalled. After all the ring ready registers in chips are set (or the signal, *ext\_shift\_en*

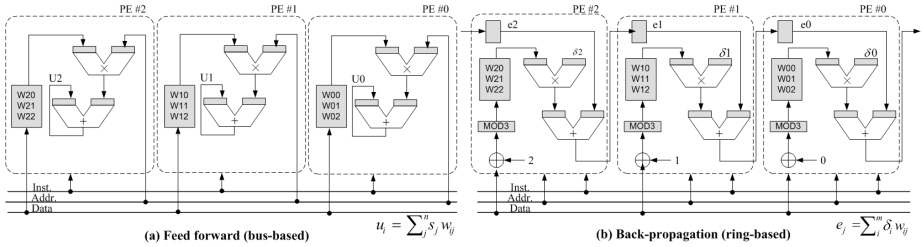


Fig. 2. Row and column mode memory access with AM

becomes ‘1’), the chip-to-chip ring shift operation is performed forming a larger ring across all the chips. This simplifies the chip-to-chip data transfer in multi-chips expansion. The contents of program memory of each chip are duplicated, and share the external data memory for multi-chip SIMD operation.

### 2.3 Instruction Set

The function of the proposed processor is programmed by means of 24 customized instructions for target applications. They include instructions for memory access, data transfers, arithmetic operations, and flow controls. Table 1 shows the instruction list including several special purpose instructions such as *BR*, *WLD*, etc. *BR* is used for broadcasting data through ring or bus. The selection is made by the flag bit appended to *BR* instruction. *WLD* loads data from an LM of its neighbor PUs or its own PU to RF, at this time, the AM can be selected to operate on column-wise or row-wise memory access.

## 3 Character Recognition System

The functionality and the performance of the proposed processor are verified with the character recognition application based on ANNs including image processing. Figure 3 shows the overall architecture of the proposed character recognition system. Generally, the character recognition application is separated into three phases [6]. The first phase is the image pre-processing using translation, dilation, rotations, thinning, and so on to bring a character to a standardized form. The second phase is the feature extraction that corresponds to linear or non-linear filtering. The third phase is the classification based on features that are obtained in the second phase. If the recognition system requires learning or adaptation, then additional learning or training stage is required.

### 3.1 Preprocessing and Feature Extraction

The pre-processing and the feature extraction consist of 4 stages; *thinning*, *image filtering*, *connection*, and *shrinking*. These operations are based on two dimensional morphological filtering [7]. The *thinning* skeletonizes an input image

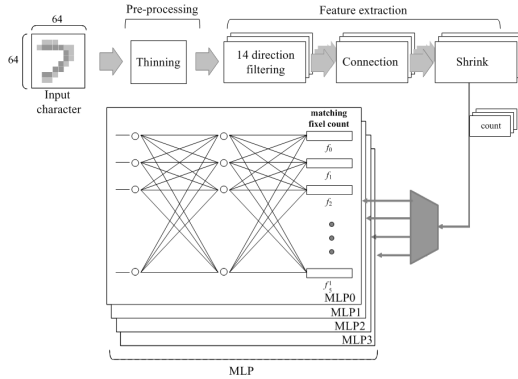
Table 1. Instruction Set

Instruction	Description	
<b>NPLD</b> CRn,@(disp:21),D	CRn and (RR or DBR) $\leftarrow DM@(disp:21+CR0)$	Load a data to data bus or ring bus from external memory (D=0: ring bus, D=1: data bus)
<b>NPST</b> CRn, @(disp:13),S	$DM@(disp:13+CR0)$ @ (disp:13),S	Store a data in an OR into external memory $\leftarrow OR@CRn$
<b>NPMV</b> Rn,Rm,P,S	$Rn \leftarrow Rm$	Move a data in a register Rn to a register Rm
<b>NBR</b> CRn,C,D	DBR or RR $\leftarrow$ GM@CRn or CRN	Broadcast a data in a register GRn of a general propose register file a data bus(D) or a ring bus(R)
<b>NMAC</b> Rm,Rj,S	$AR \leftarrow (Rm \times Rj)+AR$	Multiply Rm and Rj, then add with a accumulator register AR
<b>NPADDim</b> Rj, #(imm:16) C S	$Rj \leftarrow Rj + \#(imm:16)$	Add Rn with an immediate value (C=0: add in PU, C=1: add in CU)
<b>NPSUBim</b> Rj, #(imm:16) C S	$Rj \leftarrow Rj - \#(imm:16)$	Subtract Rn with an immediate value (C=0: subtract in PU,C=1: subtract in CU)
<b>NBS</b> OPT,Rn,Rm, SH,OPT,S	$Rn \leftarrow$ Barral shift(Rm)	$Rn \leftarrow$ Barral shift(Rm) with shift amount of CRj
<b>NWLD</b> Rn #(LMAddr:12) M S	$Rn \leftarrow LM@(LMAddr+RF0)$	Load a data in a local memory to a register Rn
<b>WST</b> Rn #(LMAddr:12) S	$LM@(LMAddr+RF0) \leftarrow Rn$	Store a data in a register Rn to a local memory
<b>NSHIFT</b> Rn,Rm	$Rn \leftarrow$ 1 bit shift (Rm)	Shift a data in a register Rm with one bit to a register Rn
<b>NFU</b> CRn,CRm,Cj,M,D	GM@CRn and (DBR or RR) or CRn $\leftarrow$ NFU OR@CRm with shift offset in Cj	NFU look-up table access with a data in an OR of a PU and broadcast NFU data to a data bus(D) or a ring bus(R)

while preserving its original shape. After that, the skeletonized image is filtered with 12 two-dimensional morphological feature filters like direction, angle, crossing, and T-crossing filters. Each feature extraction filtering is performed in each PU and the filter weight is stored in LM. The input image is broadcasted through the data bus and the filter output is stored in LM. During the *image filtering* certain lines may be broken. These broken lines have to be reconnected by the *connection* process that is a morphological dilation. Then the object in the resulting image is shrunken down until only one point remains for each object. The number of renaming point is counted for each feature filter and it represents how many corresponding features are in the input image. These numbers are used as an input vector for the classifier that is realized through MLP with BP learning.

### 3.2 Multi-layer Perception (MLP)

MLPs are well-established multipurpose classifying algorithm, and they are frequently employed in recognition systems. On the proposed system, the MLP consists of three layers; input, output, and one hidden layer. Each layer consists of 16 nodes. To improve recognition performance, the input character sets are grouped into one of four sub-nets according to the number of strokes in the character. Figure 4 shows the general structure of MLP with memory access



**Fig. 3.** Pattern recognition application using the proposed processor

mode on FF and BP stages. Each PU is assigned for one neuron on a layer. Therefore, one PU holds two weight sets, one for the first layer and the other for second layer. The FF path is processed with one input element at a time. The first input element  $z_1$  is broadcasted through the data bus and all PUs compute the corresponding synaptic weight using  $v_{ji}$  stored in each PU, and then the second input element  $z_2$  is broadcasted through the data bus. The same operations are repeated for all input elements and the result of synaptic weight for every input is accumulated at AR in each PU. This is the process between the input layer and the hidden layer, which can be express as the equation,  $net_j = \sum_{i=0}^{j-1} v_{ji}z_i$ . After that  $net_j$  is moved to NFU through output register (OR) to calculate the output value  $y_i = f_i(net_j)$  for the hidden layer. At this time,  $y_i$  is also stored at global memory since it is going to be read again on BP stage. The same sequence of computations for the second layer is repeated to calculate the  $o_k$  in output neuron using the input  $y_i$ . This process can be expressed by the equations  $net_k = \sum_{j=0}^{k-1} \omega_{kj}y_j$  and  $o_k = f_k(net_k)$ . In the BP stage, the produced output  $o_k$  is compared with the desired output  $d_k$  and an error value  $\delta_{ok} = (d_k - o_k)f'_k(net_k) = (d_k - o_k)o_k(1 - o_k)$  is propagated backward to update weight values. The process is expressed as following equations;  $\delta_{ok} = (d_k - o_k)f'_k(net_k)$  and  $\omega_{kj} = \omega_{kl} + \eta\delta_{ok}y_i$  between the output layer and the hidden layer;  $\delta_{yj} = (\sum_{k=0}^{k-1} \delta_{ok}\omega_{kj})f'_k(net_k)$  and  $v_{ji} = v_{ji} + \eta\delta_{yi}z_i$  between the hidden layer and the input layer. Finally, the weight is updated using  $\delta_{yj}$ . These operations can be summarized as follows. First, the weight is expressed in the form of two dimensional matrix and stored in local memory in row order. Second, the input values are broadcasted to all PUs through the bus on the FF stage. Therefore, all PUs read weights at the same memory location and executes MAC operation. Third, on BP stage the AM modifies the memory address and the weights are read and calculated. And the desired values are broadcasted through the ring. This allows the proposed processor to operate on both row and column mode memory access without overhead. Another feature is that the calculation of non-linear function. The non-linear function requires

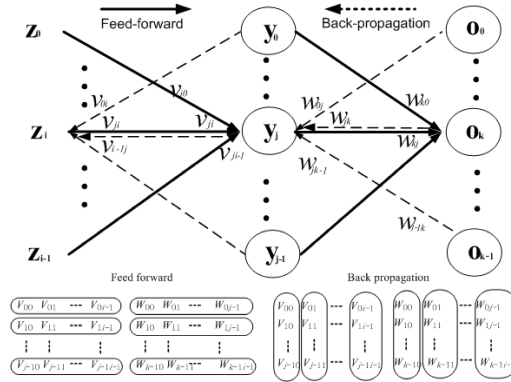


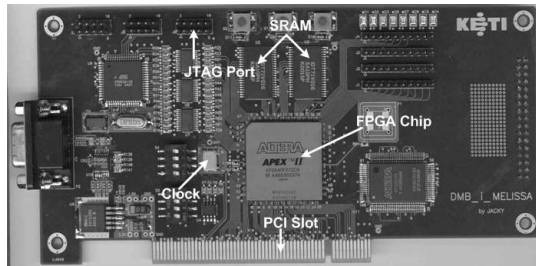
Fig. 4. MLP with back-propagation learning

complex computation or large look-up table. Implementation of such block in each PU significantly increases the hardware complexity. Therefore, only one NFU is implemented as a look-up table. In this case, the computation of non-linear function win may be a bottleneck of the over all performance. However, the proposed architecture allows effective bus management which eliminated the bottleneck to use NFU. Furthermore, the processing time also can be reduced by storing the output values between each layer in global memory because they are expected to be fetched again on the BP stage.

### 4 Implementation Result and Conclusion

Figure 5 shows the physical layout of the prototype system board using the proposed processor architecture embodied in FPGA chip. The operating clock is 55MHz and operating voltage is 3.3V. The overall size is utilizing 19,914 logic elements equivalent to 320,000 gate level. CU is implemented with 32-bit  $\times$  4K-word program memory, 32-bit  $\times$  16-word register file, program counter (PC) and 24-bit adder. NFU is consisted of 8 bit  $\times$  512-word memory and BUS is 32-cell 16-bit ring chain. The implemented recognition system is trained with 20 sets of 17 handwritten alphabets for 1,770 iterations on incremental learning mode. It took 24 seconds for pre-processing, feature extraction, classification, and learning. In order to compare its processing time, the application was implemented by using C++ program running on 2.8GHz Pentium IV personal computer with 1GB SDRAM, and its processing time was 14 seconds. The proposed processor showed no more than 1.7 times slower performance than PC-implementation, but nevertheless it run with relatively very slow operation clock of 55 MHz and small memory capacity of 256KB embedded SRAM. Suppose the proposed processor is implemented as a chip using 0.18-micron process technology, it is expected to operate at 400MHz clock speed, and then its computing power could be over 4 times faster than 2.8 GHz PC. This paper proposed a high performance acceleration processor optimized for scientific computations

such as image processing and ANNs. Highly customized 24 instructions were devised to improve the performance and the programmability of the processor on the target applications. From the architectural point of view, the characteristic of the proposed processor is SIMD with 16 PUs and special hardware resources such as NFU, a ring, and buses. Each PU includes arithmetic ALU (32-bit adder/subtractor/multiplier) and logical ALU (32-bit bitwise operators: AND, OR, XOR). The proposed architecture is suitable for applications that require heavy memory access relatively low computational complexity. Furthermore, the AM in each PU enables the proposed processor to have the ability to operate on column wise and row wise memory access, which can be exploited by many linear algebra applications.



**Fig. 5.** A Prototype system board

## References

1. Shiva S.G.: *Pipelined and Parallel Computer Architectures*. Harper-Collins, New York (1996)
2. Boulet P., Fortes J.A.B.: Experimental Evaluation of Affine Schedules for Matrix Multiplication on the MasPar Architecture. *Proc. 1st International Conf. on Massively Parallel Computing Systems*. (1994) 452–459
3. Hicklin J., Demuth H.: Modeling Neural Networks on the MPP. *Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation*. (1988) 39–42
4. Lam K.D., Pattnaik V., Seung-Moon Y., Torrellas J., Huang W., Kang Y., Zhenzhou G.: FlexRAM: Toward an Advanced Intelligent Memory System. In *proceedings of International Conf. on Computer Design99*. (1999) 192–201
5. Chong F., Oskin M., Sherwood T.: Active pages: A ComPutation Model for Intelhgent Memory. *Proc 25th Annual International Symposium on ComPuter Architecture*. (1998) 192–203
6. Salembier P., Brigger P., Casas J. R., Pardas M.: Morphofogical Operators for Image Ellld Video Compression. *IEEE Trans Image Process*. **5** (1996) 881–898
7. Yentis R., Zaghloul M. E.: VLSI Implementation of Focally Connected Neural Networks for Solving Partial Differential Equations. *IEEE Trans. Circuits Syst. I, Fundaln. Theory Appl*. **43** (1996) 687–690