IEICE TRANS. INF. & SYST., VOL.E87–D, NO.7 JULY 2004

PAPER Special Section on Hardware/Software Support for High Performance Scientific and Engineering Computing

An Acceleration Processor for Data Intensive Scientific Computing

Cheong Ghil KIM[†a)], **Hong-Sik KIM**[††], **Sungho KANG**[††], **Shin Dug KIM**[†], *and* **Gunhee HAN**[††], ***Nonmembers***

SUMMARY Scientific computations for diffusion equations and ANNs (Artificial Neural Networks) are data intensive tasks accompanied by heavy memory access; on the other hand, their computational complexities are relatively low. Thus, this type of tasks naturally maps onto SIMD (Single Instruction Multiple Data stream) parallel processing with distributed memory. This paper proposes a high performance acceleration processor of which architecture is optimized for scientific computing using diffusion equations and ANNs. The proposed architecture includes a customized instruction set and specific hardware resources which consist of a control unit (CU), 16 processing units (PUs), and a non-linear function unit (NFU) on chip. They are effectively connected with dedicated ring and global bus structure. Each PU is equipped with an address modifier (AM) and 16-bit 1.5 k-word local memory (LM). The proposed processor can be easily expanded by multi-chip expansion mode to accommodate to a large scale parallel computation. The prototype chip is implemented with FPGA. The total gate count is about 1 million with 530, 432-bit embedded memory cells and it operates at 15 MHz. The functionality and performance of the proposed processor is verified with simulation of oil reservoir problem using diffusion equations and character recognition application using ANNs. The execution times of two applications are compared with software realizations on 1.7 GHz Pentium IV personal computer. Though the proposed processor architecture and the instruction set are optimized for diffusion equations and ANNs, it provides flexibility to program for many other scientific computation algorithms.

key words: *SIMD, FPGA, artificial neural networks, diffusion equations, image processing*

1. Introduction

The scientific computations have demanded high performance 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 [1]. 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]–[4]. These SIMD machines were very large, complex, and expensive parallel supercomputers.

As the VLSI technology growing, architecturally new SIMD array processors, in which the processor and memory were integrated into a single chip to overcome the memory access bottleneck, were introduced in the form of special purpose processor or coprocessor and intelligent memory systems of a workstation or server [5]–[8]. The majority of these have the same concept to connect many simple processors together for fast parallel processing, and they are dedicated to a specific application in the lack of generalities. This application specific hardware limits the range of applications and causes the developed processor price high.

Recently, the aids of growing fabrication technology and CAD tool advance, which allow the implementation of a complex system on a chip in relatively low cost, have brought the advent of the parallel system based on DSP array processors and the parallel processor implementation optimized for scientific computing and neural networks. The former offers high flexibility, however, the computational efficiency is not high enough for certain applications [9]–[11]. The later approach implements PUs with relatively low performance arithmetic unit and local memory for each PU [12]–[15]. The memory access bottleneck is overcome by distributed memory and the computing power is obtained by parallelism. The main drawbacks of the parallel implementation are low computational efficiency in certain application and lower flexibility when it is compared with the former approach.

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; AM, 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. For example, the applications using diffusion equations and ANNs are data intensive tasks rather than computational ones since the computation tasks are simple with heavy memory access for the entire processing. Although both applications require data intensive processing, their behavioral models are different. The simulation of diffusion equations includes data transfer among PUs and the mapping of ANNs requires multiple memory indexing modes in each PU.

The rest of this paper is organized as follows. In Sect. 2, we describe the architecture and operation of the proposed processor. Section 3 shows the result of our performance evaluation. Section 4 provides the implementation result. Finally, we conclude in Sect. 5.

Manuscript received October 15, 2003.
Manuscript revised January 15, 2004.

[†]The authors are with the Department of Computer Science, Yousei University, Seoul 120–749, Korea.
[††]The authors are with the Department of Electrical & Electronic Engineering, Yonsei University, Seoul 120–749, Korea.
a) E-mail: cgkim@parallel.yonsei.ac.kr

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 Fig. 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. GRF (global register file) 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.

Figure 2 shows the block diagram of a PU. Each PU consists of 16-bit fixed point numerical arithmetic units, a 16-bit 16-word register file, 16-bit 1.5 k-word LM, special purpose registers (CR, FR, and AR), and an address modifier (AM). In addition, a 16-bit logical arithmetic unit (LALU) is embedded for basic logical operations (*AND*, *OR*, *XOR*, and *NOT*) and two special purpose logical instructions (*CONCATENATE* and *SHRINK).* For MAC (multiply and accumulate) 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.

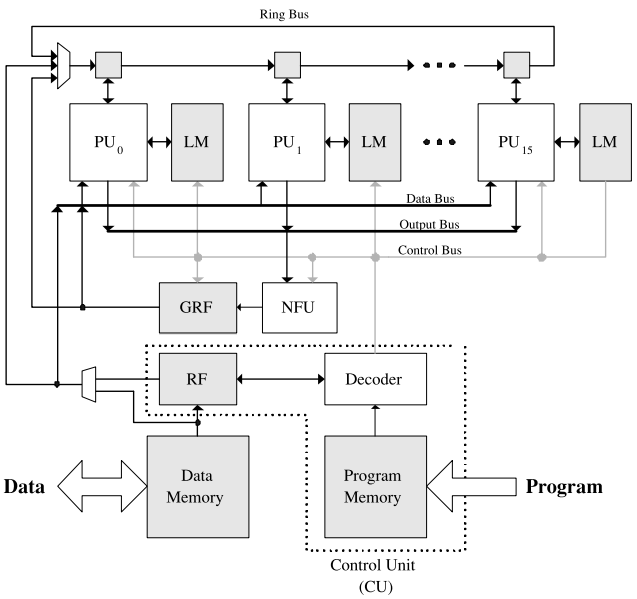


Fig. 1 Block diagram of the proposed architecture.

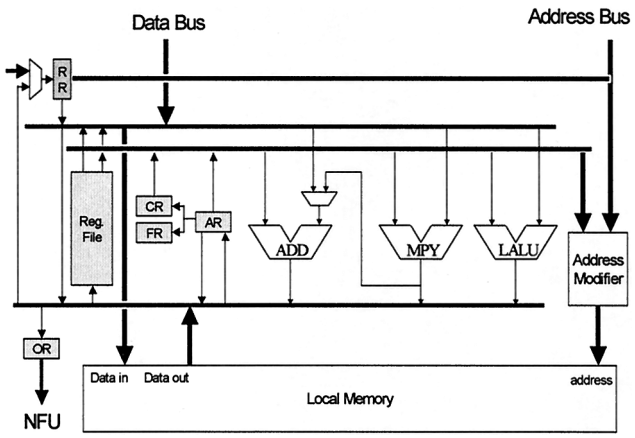
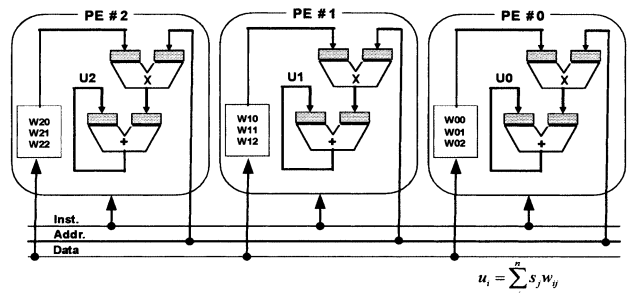


Fig. 2 Block diagram for a PU.

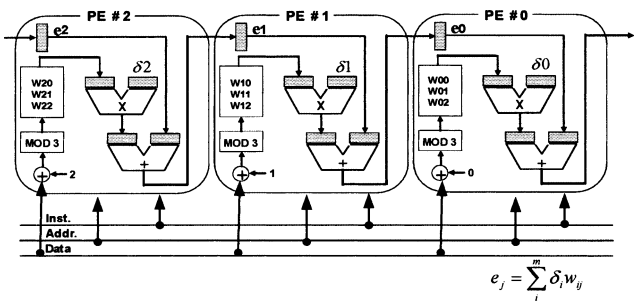
2.1 Address Modifier (AM)

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. 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 3 shows an operational model of how an AM works on MLP (multi-layer perceptron) with back-propagation. Here, a row of the forward weight matrix is allocated to each PU. The first is feed-forward operation, in which the network computes the equation, $u_i = \sum_j^n s_j w_{ij}$. The second is error back-propagation that computes the equation, $e_j = \sum_i^m \delta_i w_{ij}$. From these two equations, the



(a) Feed forward (bus-based)



(b) Backpropagation (ring-based)

Fig. 3 Row and column mode memory access with AM.

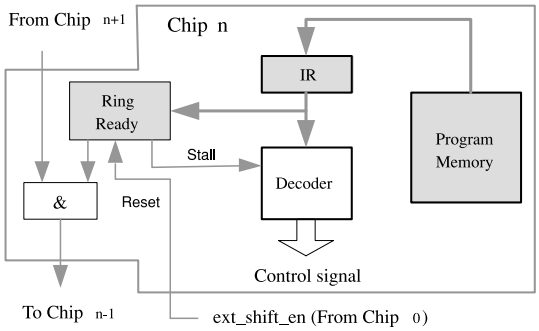


Fig. 4 Multi-chip expansion.

weights distributed over LMs should be accessed in two different modes; the row-wise for the feed-forward and the column-wise for the error back-propagation. In the proposed processor, three mechanisms, ring, bus, and AM, are used for effective memory access for BP. For the process of feed-forward operation, the address is broadcast to all PUs simultaneously through bus as shown on Fig. 3 (a) since the weights are stored in local memories in row order. In error back-propagation 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 Fig. 3 (b). Therefore, the proposed architecture enables both row and column wise memory access without many overheads.

2.2 Multi-Chip Expansion

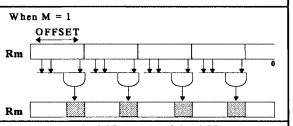
The expandability of the proposed processor is essential because most scientific computations require large sizes of parallel processing; Figure 4 shows the block diagram of multi-chip expansion. The multi-chip expansion through a register ring is used for increasing the network size. This is called 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*, 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 26 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*, *SHRINK*, *CONCATENATE*, 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. *SHRINK* and *CONCATENATE* are special purpose instructions to increase the efficiency of bitwise filtering operations for the character recognition application. *CONCATENAE* creates a new word by combining two groups of bit from two input operands. One group of bits is taken from the first operand with the size of *offset* bits from MSB, while the other is taken from the second operand with the size of (*16 bits-offset*) bits from LSB. In *SHRINK*, the input operand is sequentially divided into several sub-blocks with the size of *offset* bits and the bit-wise operation is performed for all bits in each sub-block, and the results are stored in an output operand.

Table 1 Instructions.

Instruction	Description	
LD CRn, @(disp:21), D	CRn and (RR or DBR)← DM@(disp:21 + CR0)	Load a data to data bus or ring bus from external memory If D=0 then broadcast through ring bus If D=1 then broadcast through data bus
ST CRn, @(disp:13), S	DM@(disp:13+CR0) ← OR@CRn	Store a data in an *OR* into external memory
MV Rn, Rm, P, S	Rn ← Rm	Move a data in a register R_n to a register R_m
BR CRn, C D	DBR or RR ← GM@CRn or CRn	Broadcast a data in a register GR_n of a general purpose register file to a data bus (D) or a ring bus (R)
MAC Rm, Rj, S	AR ← (Rm × Rj) + AR	Multiply R_m and R_j, then Add with a accumulator register *AR*
ADDim Rj #(imm:16) C S	Rj ← Rj + #(imm:16)	Add R_n with an immediate value If C = 0 then add in PU If C = 1 then add in CU
SUBim Rj #(imm:16) C S	Rj ← Rj – #(imm:16)	Subtract R_n with an immediate value If C = 0 then add in PU If C = 1 then add in CU
CONCANTENATE Rj, Rn, Rm, OFF, S	Rj ← Concatenate (Rm,Rj, OFF)	Rj = {Rn[15:OFF], Rm[15:16-OFF]}
SHRINK Rn, Rm, M, OFF, S	Rn ← Shrink(Rm,OFF)	
BS OPT, Rn, Rm, SH, OPT, S	Rn ← Barral shift(Rm)	Rn ← Barral shift(Rm) with shift amount of CRj
WLD Rn #(LMAddr:12) M S	Rn ← LM@(LMAddr + RF0)	Load a data in a local memory to a register R_n
WST Rn #(LMAddr:12) S	LM@(LMAddr+RF0) ← Rn	Store a data in a register R_n to a local memory
SHIFT Rn, Rm	Rn ← 1 bit Shift (Rm)	Shift a data in a register R_m with one bit to a register R_n
NFU CRn, CRm, Cj, M, D	GM@CRn and (DBR or RR) or CRn ←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)

3. Verification

The functionality and the performance of the proposed processor are verified with the character recognition application based on ANNs including image processing and the simulation of diffusion equations for oil reservoir. We have programmed those applications and estimated the speed performance.

3.1 Character Recognition

Figure 5 shows the overall architecture of the proposed char-

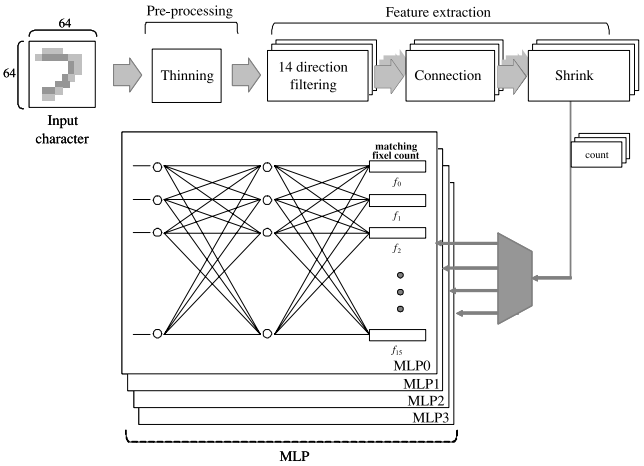


Fig. 5 Proposed pattern recognition system.

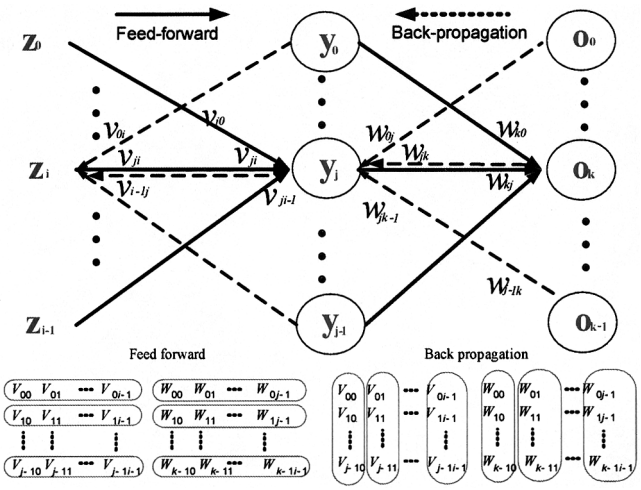


Fig. 6 MLP with back-propagation learning.

acter recognition system. Generally, the character recognition application is separated into three phases [16]. 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 adaption, then additional learning or training stage is required.

3.1.1 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 [17].

The *thinning* skeletonizes an input image 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 back-propagation learning.

3.1.2 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.

Table 2 Equations for MLP.

Feed forward stage	Back propagation stage
Input layer => Hidden layer	Output layer => Hidden layer
$net_j = \sum_{j=0}^{j-1} v_{ji} z_i$	$\delta_{ok} = (d_k - o_k) f'_k(net_k)$
$y_i = f_j(net_j)$	$w_{kj} = w_{kl} + \eta \delta_{ok} y_i$
Hidden layer => Output layer	Hidden layer => Input layer
$net_k = \sum_{k=0}^{k-1} w_{kj} y_j$	$\delta_{yj} = (\sum_{k=0}^{k-1} \delta_{ok} w_{kj}) f'_k(net_k)$
$o_k = f_k(net_k)$	$v_{ji} = v_{ji} + \eta \delta_{yi} z_i$

Figure 6 shows the general structure of MLP with memory access mode on feed-forward and back-propagation stages. Table 2 shows the equations. 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 feed-forward 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, $net_j = \sum_{j=0}^{j-1} v_{ji} z_i$. After that net_j is moved to NFU through OR (output register) to calculate the output value $y_i = f_j(net_j)$ for the hidden layer. At this time, y_i is also stored at global memory since it is going to be read again on back-propagation 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 $net_k = \sum_{k=0}^{k-1} w_{kj} y_j$ and $o_k = f_k(net_k)$.

In the back-propagation 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 $w_{kj} = w_{kl} + \eta \delta_{ok} y_i$ between the output layer and the hidden layer; $\delta_{yj} = \left(\sum_{k=0}^{k-1} \delta_{ok} w_{kj}\right) f'_k(net_k)$ and $v_{ji} = v_{ji} + \eta \delta_{yi} z_i$ between the hidden layer and the input layer. Finally, the weight is updated using δ_{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 feed-forward stage. Therefore, all PUs read weights at the same memory location and execute MAC operation. Third, on back-propagation 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 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 will 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 back-propagation stage.

3.2 Diffusion Equations

Simulation of a diffusion equation for oil reservoir is a data intensive processing which is used to predict the future oil production of an oil reservoir based on geological data of the reservoir. In the petroleum industry, the demand for fast simulation of dynamics in the reservoir is ever increasing.

The oil reservoir problem can simply be mapped into a Resistor-Capacitor (RC) network for solving a system of dynamic equations [18]. The discrete form of the parabolic PDE describing pressure transient response during a well test is given in Eq. (1). Here, p_{ij} is the fluid pressure of the cell located at (i, j).

$$\begin{aligned} 0 = {} & \frac{1}{2\Delta x^2}[(h_{i,j}k_{i,j} + h_{i+1,j}k_{i+1,j})(p_{i+1,j} - p_{i,j}) \\ & -(h_{i-1,j}k_{i-1,j} + h_{i,j}k_{i,j})(p_{i,j} - p_{i-1,j})] \\ & +\frac{1}{2\Delta y^2}[(h_{i,j}k_{i,j} + h_{i,j+1}k_{i,j+1})(p_{i,j+1} - p_{i,j}) \\ & -(h_{i,j-1}k_{i,j-1} + h_{i,j}k_{i,j})(p_{i,j} - p_{i,j-1})] \\ & +C\frac{\mu}{\Delta x \Delta y} q_{i,j} - C_1 \mu \phi_{i,j} h_{i,j} \frac{\partial p_{i,j}(t)}{\partial_t} \end{aligned} \tag{1}$$

Where: $h_{i,j}$ is the thickness of the grid (i, j)
$k_{i,j}$ is the permeability at the grid (i, j)
$\phi_{i,j}$ is the porosity at the grid (i, j)
$q_{i,j}$ is the injection/production rate at the grid (i, j)
μ is the water viscosity (0.7)
C, C_1 are conversion factors ($C = 887.178$ psi)
Δx is the grid length in the x direction
Δy is the grid length in the y direction

A special case of Eq. (1) is the pattern balancing case in which the last term in the equation is zero and the problem becomes one of steady-state pressure distribution. Note that Eq. (2) is equivalent to the original discrete reservoir equation provided that the grid size along the x and y directions are equal (i.e. $\Delta x = \Delta y$). Equation (1) can be solved iteratively with the proposed processor by defining:

$$\begin{aligned} p^k_{i,j} = {} & \alpha_{i,j}(p^-_{i-1,j} - p^-_{i,j}) - \alpha_{i-1,j}(p^-_{i,j} - p^-_{i-1,j}) \\ & + \alpha_{i,j}(p^-_{i+1,j} - p^-_{i,j}) - \alpha_{i+1,j}(p^-_{i,j} - p^-_{i+1,j}) \\ & + \alpha_{i,j}(p^-_{i,j-1} - p^-_{i,j}) - \alpha_{i,j-1}(p^-_{i,j} - p^-_{i,j-1}) \\ & + \alpha_{i,j}(p^-_{i,j+1} - p^-_{i,j}) - \alpha_{i,j+1}(p^-_{i,j} - p^-_{i,j+1}) \\ & + \chi_{i,j} \end{aligned} \tag{2}$$

$$p^+_{i,j} = p^-_{i,j} + \beta_{i,j} p^h_{i,j} \tag{3}$$

The definition for the lumped system parameters used in the equation above and their relationship to the original reservoir parameters are given in Eq. (4). Those symbols (α, β, and χ) are parameters of each cell obtained from geological exploration.

$$\begin{aligned} \alpha_{i,j} &= \frac{1}{2\Delta x^2} h_{i,j} k_{i,j} \\ \chi_{i,j} &= C\frac{\mu}{\Delta x \Delta y} q_{i,j} \\ \beta_{i,j} &= C_1 \mu \phi_{i,j} h_{i,j} \end{aligned} \tag{4}$$

```
LOOP:                                                   // load data
        MV      AR      RF0     MP      RD;     // restore original RF0 => AR
        SUBim   RF0     One     SP      RD;     // Up memory index
        WLD     RF4     A_Index MD      RD;     // load  α_{i-1,j}
        JMPT    CR11    JUMP_1  PJ      INT;
        WLD     RF5     P1_Index MD     RD;     //
        JMP     RTN_1;
JUMP_1:
        WLD     RF5     P2_Index MD     RD;     //
RTN_1:
        ADDim   RF0     Two     AP      RD;     // Down memory index
        WLD     RF6     A_Index MD      RD;     // load  α_{i+1,j}
        JMPT    CR11    JUMP_2  PJ      INT;
        WLD     RF7     P1_Index MD     RD;     //
        JMP     RTN_2;
JUMP_2:
        WLD     RF7     P2_Index MD     RD;     //
RTN_2:
        MV      RF0     AR      MP      RD;     // restore original RF0
        MPY     AR      RF4     RF5     RD;     // α_{i-1,j} * p_{i-1,j}
        ADD     RF3     RF3     AR      RD;     //
        MPY     AR      RF6     RF7     RD;     // α_{i+1,j} * p_{i+1,j}
        ADD     RF3     RF3     AR      RD;     //
        ADD     RF1     RF4     RF6     RD;     // α_{i-1,j} + α_{i+1,j}
        ADD     RF2     RF5     RF7     RD;     // p_{i-1,j} + p_{i+1,j}
```

Fig. 7 Part of oil reservoir simulation code.

However, the simulation time of a large reservoir that is divided into millions of grid blocks due to the large number of elements in typical meshes. The problem grows very steeply as the grid size increases.

Figure 7 shows a part of program code for oil reservoir simulation. Most of operations consist of data read, addition, and multiplication. On the proposed processor, the whole grid is divided into several sub sections. Each PU stores geological data for each sub section on its LM. The boundary cell data on neighboring PU can be accessed through ring bus. This mechanism deals effectively with data-transfer intensive operations. Furthermore, multi-chip expansion mode easily provides the massive parallel processing as the mesh size grows.

4. Implementation Result

Figure 8 shows the physical layout of the prototype system board using the proposed processor architecture embodied in FPGA chip. Table 3 summarizes the FPGA implementation result and the specifications of the proposed processor. The operating clock is 15 MHz. The overall size is utilizing 29,923 logic elements equivalent to 120 million gate level.

The implemented recognition system is trained with 20 sets of 17 handwritten Korean alphabets for 1,770 iterations on incremental learning mode. Figure 9 shows that 4 MLPs successively trained. It took 68 seconds for preprocessing, feature extraction, classification, and learning. In order to compare its processing time, the application was implemented by using C++ program running on 1.7 GHz Pentium IV personal computer with 512 MB DRAM, and its processing time was 19 seconds. The proposed processor showed no more than 3.5 times slower performance than PC-implementation, but nevertheless it run with relatively very slow operation clock of 15 MHz and small memory capacity of 50 KB embedded SRAM. Suppose the proposed processor is implemented as a chip using 0.18-micron process technology, it is expected to operate at 200 MHz clock speed, and then its computing power could be over 3 times faster than 1.7 GHz PC.

For diffusion equations, we estimate the computation speed according to the variation of mesh size. The following methods allow us to obtain the execution time in terms of CPU clock cycles. We use Pentium's RDTSC (Read Time Stamp Counter) [19] instruction to measure the execution time of the application implemented with C++ program on 1.7 GHz Pentium IV personal computer. The execution time was obtained by executing the code five times inside a loop, and then summed averaged execution time was selected. This was done to include all cache effects and to ensure the in-order execution of the RDTSC instruction, therefore, that execution time reported from the experiment represents the processor's best performance. In the proposed processor, the execution time information is obtained by calculating based on the processing time of one chip with 16 PUs. The

Table 3 Specification.

Specification	
FPGA implementation result	
number of logic elements	29,923
clock frequency	15MHz
operating voltage	1.5V
embedded memory	540,672 bits
CU	
program memory	32-bit * 4K-word
register file	32-bit * 16-word
PC (program counter)	
decoding logic	
16-bit adder	
NFU	
memory	6-bit * 512-word
BUS	
16-bit data bus	
16-cell 16-bit ring chain	
PU	
number of PUs/chip	16
data path	16-bit fix point
local memory	16-bit * 1.5K-word
arithmetic ALU	16-bit adder / subtracter 16-bit multiplier
logical ALU	16-bit bitwise operators (AND, OR, XOR), 16-bit barrel shifter
register file	16-bit * 8-word
special registers	CR, FR, AR
AM (address modifier)	
Instruction set	
number of instructions	26
opcode	5 bits

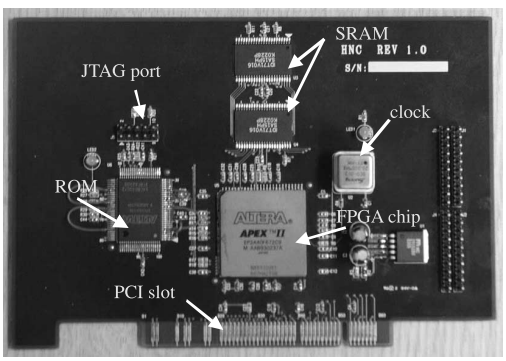


Fig. 8 System board with FPGA chip.

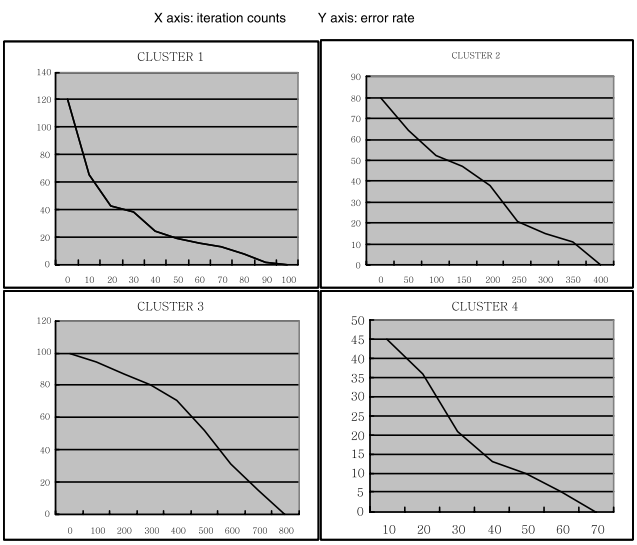


Fig. 9 Convergence result from the independent chip.

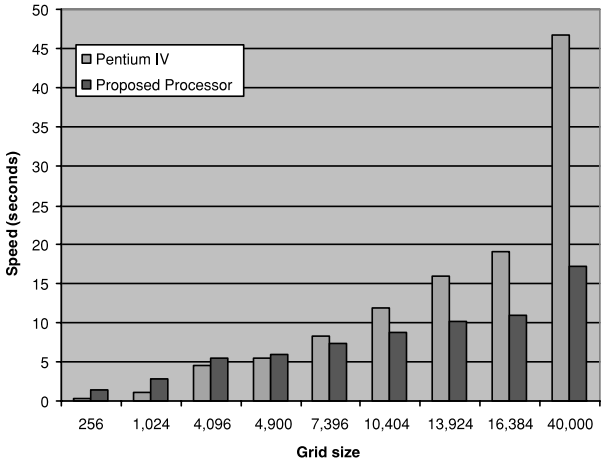


Fig. 10 Comparison of execution time.

execution time information is obtained by running the system board. The result shows that the execution time of the proposed processor outperforms PC implementation as the mesh size grows over 70 by 70 as shown in Fig. 10.

5. Conclusion

This paper proposed a high performance acceleration processor optimized for scientific computations such as diffusion equations and ANNs. Highly customized 26 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. 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. Very simple multi-chip expansion is available for massively parallel processing. The prototype chip was implemented with FPGA. The functionality and performance of the proposed processor was verified with oil reservoir simulation and character recognition application. The former shows that the proposed processor outperforms 1.7 GHz Pentium IV PC as the mesh size grows over 70 by 70. The latter shows that the propose processor has the possibility to achieve over 3 times better computation power than the PC, being implemented as a chip using 0.18-micron process technology.

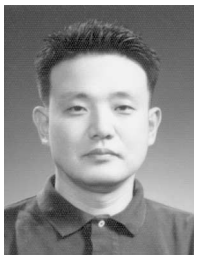
Acknowledgments

This work was supported by Next Generation Technologies Program of Commerce, Industry, and Energy in South Korea (2001-2-0975).

References

[1] S.G. Shiva, Pipelined and Parallel Computer Architectures, Harper-Collins, New York, 1996.
[2] P. Boulet and J.A.B. Fortes, "Experimental evaluation of affine schedules for matrix multiplication on the MasPar architecture," Proc. 1st International Conf. on Massively Parallel Computing Systems, pp.452–459, May 1994.
[3] J. Hicklin and H. Demuth, "Modeling neural networks on the MPP," Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation, pp.39–42, Oct. 1988.
[4] B.A. Kahle and W.D. Hillis, "The connection machine model CM-1 architecture," IEEE Trans. Syst. Man Cybern., vol.19, no.4, pp.246–252, 1989.
[5] D.G. Elliott, R. Mason, P.M. Nyasulu, and W. Snelgrove, "Minimizing the effect of the host bus on the performance of a computational RAM logic-in-memory parallel-processing system," Proc. Custom Integrated Circuits '99, pp.631–634, 1999.
[6] Y. Fujita, S. Okazaki, and N. Yamashita, "A compact real-time vision system using integrated memory array processor architecture," IEEE Trans. Circuits Syst. Video Technol., vol.5, no.5, pp.446–452, Oct. 1995.
[7] K.D. Lam, V. Pattnaik, S.-M. Yoo, J. Torrellas, W. Huang, Y. Kang, and Z. Ge, "FlexRAM: Toward an advanced intelligent memory system," Proc. International Conf. on Computer Design '99, pp.192–201, 1999.
[8] F.T. Chong, M. Oskin, and T. Sherwood, "Active pages: A computation model for intelligent memory," Proc. 25th Annual International Symposium on Computer Architecture, pp.192–203, July 1998.
[9] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer, "The ring array processor: A multiprocessing peripheral for connectionist applications," J. Parallel Distrib. Comput., vol.14, no.3, pp.248–259, March 1992.
[10] A. Gunzinger, U.A. Muller, W. Scott, B. Baumle, P. Kohler, H.R. von der Muhll, F. Muller-Plathe, W.F. van Gunsteren, and W. Guggenbuhl, "Achieving super computer performance with a DSP array processor," Proc. Supercomputing '92, pp.543–550, Nov. 1992.
[11] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: The terasys massively parallel PIM array," Computer, vol.28, no.4, pp.23–31, April 1995.
[12] M. Koyanagi, T. Matsumoto, T. Shimatani, K. Hirano, H. Kurino, R. Aibara, Y. Kuwana, N. Kuroishi, T. Kawata, and N. Miyakawa, "Multi-chip module with optical interconnection for parallel processor system," Proc. 45th IEEE International Conf. on Solid-State Circuits, Digest of Technical Papers, pp.92–93, Feb. 1998.
[13] Y. Kondo, Y. Koshiba, Y. Arima, M. Murasaki, T. Yamada, H. Amishiro, H. Mori, and K. Kyuma, "A 1.2 GFLOPS neural network chip for high-speed neural network servers," IEEE J. Solid-State Circuits, vol.31, no.6, pp.860–864, 1996.
[14] W. Eppler, T. Fischer, H. Gemmeke, T. Koder, and R. Stotzka, "Neural chip SAND/1 for real time pattern recognition," IEEE Trans. Nucl. Sci., vol.45, no.4, pp.1819–1823, 1998.
[15] G. Danese, I. De Lotto, F. Leporati, A. Quaglini, S. Ramat, and G. Tecchiolli, "A parallel neurochip for neural networks implementing the reactive tabu search algorithm: Application case studies," Proc. Euromicro Workshop on Parallel and Distributed Processing, pp.273–280, 2001.
[16] H. Kamruzzaman, "Comparison of feed-forward neural net algorithms in application to character recognition," Proc. IEEE Region 10 International Conf. on Electrical and Electronic Technology, vol.1, pp.165–169, Aug. 2001.
[17] P. Salembier, P. Brigger, J.R. Casas, and M. Pardas, "Morphological operators for image and video compression," IEEE Trans. Image Process., vol.5, no.6, pp.881–898, June 1996.
[18] R. Yentis and M.E. Zaghloul, "VLSI implementation of locally con-

nected neural networks for solving partial differential equations," IEEE Trans. Circuits Syst. I, Fundam. Theory Appl., vol.43, no.8, pp.687–690, Aug. 1996.
[19] http://developer.intel.com/design/xeon/applnots/241618.htm



Cheong Ghil Kim received his M.S. degree in Computer Science from Yonsei University in 2003. He is currently a Ph.D. candidate in the same department. His research interests include application specific computer architectures and low power multimedia communication systems.

Hong-Sik Kim received the B.S. and M.S. degrees from Yonsei University, Seoul, Korea in 1997 and 1999 respectively. He is pursuing his Ph.D. degree in electrical and electronic engineering also at Yonsei University. His research areas include the RTL testability and array processing test.

Sungho Kang received the B.S. degree from Seoul National University, Seoul, Korea in 1986 and the M.S. and Ph.D. degrees from the University of Texas, Austin in 1988 and 1992 respectively. He is currently a professor in the Yonsei University, Seoul, Korea. His research focuses on the VLSI design and testing.

Shin Dug Kim received the Ph.D. degree from the School of Computer and Electrical Engineering at Purdue University, West Lafayette, Indiana. He is currently a Professor in Computer Science at the Yonsei University, Seoul, Korea. His research interests include advanced computer architecture, Grid computing, and Internet applications.

Gunhee Han was born in 1965. He received the B.S. degree from Yonsei University, Seoul, Korea in 1990 and the Ph.D. degree from Texas A&M University, College Station, in 1997. He was with Texas A&M till 1998. He is an Assistant professor in the Department of Electrical and Electronic Engineering in Yonsei University since 1998. His research interests include CMOS Image Sensor, High speed serial communication, and $\Sigma\Delta$ modulator.