2D Mesh SIMD 구조에서의 병렬 행렬 곱셈의 수치적 성능 분석

論 文
10-1-2

An Analytical Evaluation of 2D Mesh-connected SIMD Architecture for Parallel Matrix Multiplication

김 정 길*
Cheong-Ghil Kim

Abstract

Matrix multiplication is a fundamental operation of linear algebra and arises in many areas of science and engineering. This paper introduces an efficient parallel matrix multiplication scheme on $N \times N$ mesh-connected SIMD array processor, called multiple hierarchical SIMD architecture (HMSA). The architectural characteristic of HMSA is the hierarchically structured control units which consist of a global control unit, N local control units configured diagonally, and N^2 processing elements (PEs) arranged in an $N \times N$ array. PEs are communicating through local buses connecting four adjacent neighbor PEs in mesh-torus networks and global buses running across the rows and columns called horizontal buses and vertical buses, respectively. This architecture enables HMSA to have the features of diagonally indexed concurrent broadcast and the accessibility to either rows (row control mode) or columns (column control mode) of 2D array PEs alternately. An algorithmic mapping method is used for performance evaluation by mapping matrix multiplication on the proposed architecture. The asymptotic time complexities of them are evaluated and the result shows that paralle matrix multiplication on HMSA can provide significant performance improvement.

Keywords : OpenCL, many-core GPU, parallel programming, convolution

I. Introduction

In recent years, microprocessors are becoming increasingly parallel machines. Multi-core processors with media extension ISA (Instruction Set Architecture) already has been the dominant trend in processor design and GPUs are getting popular as a computing platform for general purpose workloads as the main engine increasing the performance of microprocessors [1-3]. As this trend continues, the parallel programming of core computations could be the best solution for substantially improving application performance [4].

Typically, parallel machines can be abstracted as the number of CPUs or PEs (processing elements) connected with each other using communication networks. Such that they have been implemented on the basis of two major architectural models: SIMD (Single-Instruction stream Multiple-Data streams) and MIMD (Multiple-Instruction streams Multiple-Data streams) [5]. A distinct difference between SIMD and MIMD architecture is the control organization. SIMD machines have a central control unit that broadcasts a single stream of instructions to PEs for concurrent synchronous execution. In contrast, MIMD machines are distributed; each PE has an associated control unit. Thus, PEs on MIMD machines are stand-alone serial process-

접수일자 : 2011년 02월 08일
심사일자 : 2011년 02월 10일
수락일자 : 2011년 03월 20일
*교신저자, E-mail : cgkim@nsu.ac.kr

ors, with a control unit, a datapath, and local memory.

As for workloads of current microprocessors, typical applications are shifting towards data intensive and vector-oriented media type ones which encompass matrix computations in many situations. It is intuitive that parallel processing is required and SIMD approaches provide a good match to those applications.

To accelerate the computations of data intensive applications, 2D SIMD array has been implemented on various hardware architectures since the first SIMD machine project, ILLIAC IV [6]. They were introduced in the form of special purpose processor or coprocessor and intelligent memory systems of a workstation or server [7-10]. Furthermore, the increasing demands for multimedia application have brought architectural enhancements on general purpuse processors. One of these enhancements is SIMD mode multimedia extensions. Intel's MMX (Multimedia eXtension) [11] and SSE (Streaming SIMD Extensions) [12] belong to this category. They may have relatively small scale of SIMD mode parallelism compared to the previous SIMD systems mainly formed by large hardware systems, but have brought the significant speed up of SIMD techniques for complex data parallel applications.

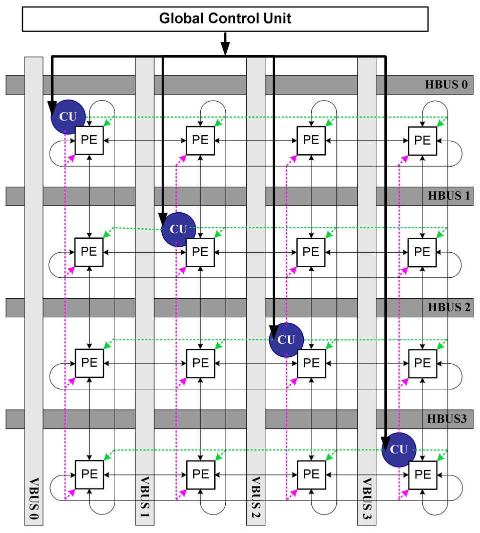
This paper presents an efficient parallel matrix multiplication and transpose scheme on HMSA [13]. For the performance evaluation of matrix multiplication is mapped onto the proposed array architecture and compared with the previous architectures in terms of the number of computation steps analytically.

The organization of this paper is as following. In Section 2, the architecture and operational model of HMSA is reviewed. Section 3 describes the parallel matrix programming paradigms on HMSA. In section 4, the experimental results will be discussed; finally the conclusion will be addressed in Section 5.

II. HMSA

1. Architectural Overview

HMSA is a modified SIMD architecture consisting of a global control unit, N local control units



(a)

그림 1. 2D Mesh SIMD 구조
Fig. 1. 2D Mesh SIMD Architecture

configured diagonally, and N^2 processing elements (PEs) arranged in an $N \times N$ array with local buses connecting with adjacent four neighbor PEs in a mesh network and global buses running across the rows and columns called vertical buses (VBUS) and horizontal buses (HBUS). Figure 1 shows an example of 4×4 PE configuration of HMSA. There are four local CUs located at the diagonal position for this given torus configuration. Both CUs and PEs have their own memory CM (Control Memory) and PM (Processing Memory), respectively.

The architectural characteristic of HMSA is the hierarchical construction of control units to provide an efficient PE control mechanism. GCU controls and synchronizes all N local CUs, and provides all the interfaces with outsides for programming and communications. The N diagonal CUs directly perform the control of PEs with two different modes, i.e., RC-mode and CC-mode. The former allows all PEs to operate on row basis, and the latter does identically on column basis.

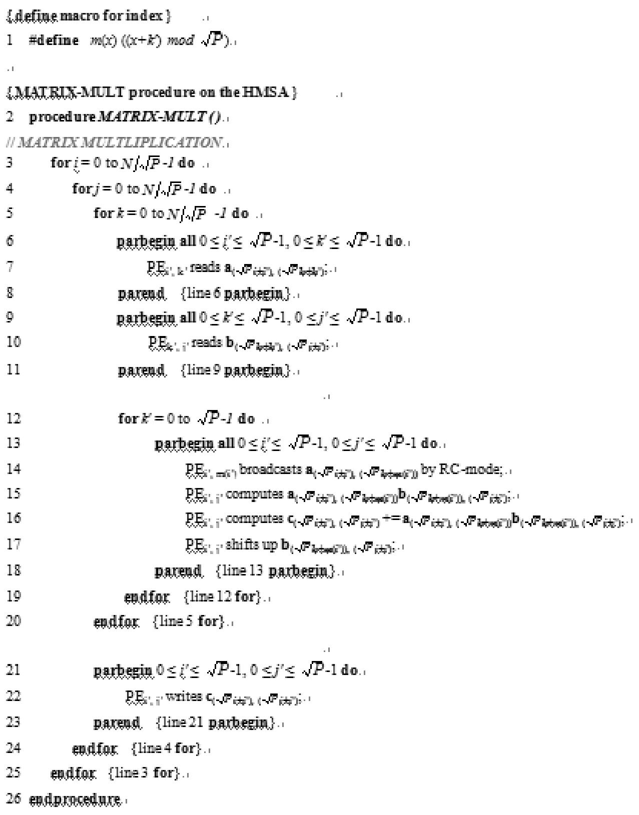
This control scheme can be efficiently utilized for the various linear algebra applications with the effective behavioral advantages in the point of performance. First, the HMSA can allow the diagonally indexed concurrent broadcast. This feature enables the efficient delivery of each operand vector to other PEs at the same time instead of sending each element of an operand vector which is a common way in two-dimensional SIMD arrays. Therefore, the matrix-by-vector products can be performed in a single cycle because the operand vector transmission operation can be overlapped with multiplication. Second, the HMSA provides a flexible process control mechanism for PE execution by the RC-mode and CC-mode.

Each PE in the HMSA is constructed as an ALU, a shifter, a set of general-purpose registers, several special-purpose registers and its dual ported memory. Special purpose registers consist of a row/column broadcasting register, a receiving register from neighborhood PEs, and a sending register to neighborhood PEs. Also two data broadcast buses, i.e., the horizontal data broadcast bus for RC-mode (HBUS) and the vertical data broadcast bus (VBUS) for CC-mode are constructed.

2. Parallel Matrix Multiplication

For the mapping processes, some assumptions of the HMSA system and matrix multiplication are made. Matrix A and B for multiplication should be already stored at the memory block of each PE. The size of each matrix and the number of processing units are assumed to be $N \times N$ and P, respectively. If N is smaller than $\sqrt{p}$, then the elements a_{ij} and b_{ij} of A and B are stored at the processor memory block of PE, $PM_{i,j}$, which is logically located at the jth column of the ith row, otherwise a different method to store the matrices at the memory is required. In that case both A and B should be divided into submatrices whose sizes should fit into the size of memory. Each row submatrix of A, $A_{i,k}$ and each column submatrix of B, $B_{k,j}$ should be stored at the same PM. The pseudo code of the proposed matrix multiplication is shown in Figure 2. To perform a matrix multiplication in the HMSA, every $PE_{i,j}$ for all $0 \le i, j \le \sqrt{p}-1$ computes in parallel by accessing the memory block $PM_{i,j}$.

The computing process is described as follows. First, every $PE_{i',k'}$ for all $0 \le i', k' \le \sqrt{p}-1$ fetches the first operands which are element $a_{i',k'}$ of submatrix $A_{i,k}$ $(0 \le i, k \le N/\sqrt{p}-1)$ in matrix A to their local registers in parallel in line 7 of Figure 2. Second, using the same method, every $PE_{k',j'}$ for all $0 \le k', j' \le \sqrt{p}-1$ fetches the second operands which are element $b_{k',j'}$ of submatrix $B_{k,j}$ $(0 \le k, j \le N/\sqrt{p}-1)$ in matrix B to their local registers in parallel in line 10. Third, every $PE_{i',m(i')}$ ($m(i') = (i' + k')$ mod) for all $0 \le i' \le \sqrt{p}-1$ concurrently broadcasts the first operand vector to each own row processor group by RC-mode and



```
{define macro for index}
1  #define  $m(x)\ ((x+k')\ mod\ \sqrt{P})$

{MATRIX-MULT procedure on the HMSA}
2  procedure MATRIX-MULT()
// MATRIX MULTLIPLICATION
3      for $i = 0$ to $N/\sqrt{P}-1$ do
4          for $j = 0$ to $N/\sqrt{P}-1$ do
5              for $k = 0$ to $N/\sqrt{P}-1$ do
6                  parbegin all $0 \le i' \le \sqrt{P}-1, 0 \le k' \le \sqrt{P}-1$ do
7                      $PE_{i',k'}$ reads $a_{(\sqrt{P}i+i'),(\sqrt{P}k+k')}$;
8                  parend  {line 6 parbegin}
9                  parbegin all $0 \le k' \le \sqrt{P}-1, 0 \le j' \le \sqrt{P}-1$ do
10                     $PE_{k',j'}$ reads $b_{(\sqrt{P}k+k'),(\sqrt{P}j+j')}$;
11                 parend  {line 9 parbegin}

12                 for $k' = 0$ to $\sqrt{P}-1$ do
13                     parbegin all $0 \le i' \le \sqrt{P}-1, 0 \le j' \le \sqrt{P}-1$ do
14                         $PE_{i',m(i')}$ broadcasts $a_{(\sqrt{P}i+i'),(\sqrt{P}k+m(i'))}$ by RC-mode;
15                         $PE_{i',j'}$ computes $a_{(\sqrt{P}i+i'),(\sqrt{P}k+m(i'))}b_{(\sqrt{P}k+m(i')),(\sqrt{P}j+j')}$;
16                         $PE_{i',j'}$ computes $c_{(\sqrt{P}i+i'),(\sqrt{P}j+j')} += a_{(\sqrt{P}i+i'),(\sqrt{P}k+m(i'))}b_{(\sqrt{P}k+m(i')),(\sqrt{P}j+j')}$;
17                         $PE_{i',j'}$ shifts up $b_{(\sqrt{P}k+m(i')),(\sqrt{P}j+j')}$;
18                     parend  {line 13 parbegin}
19                 endfor  {line 12 for}
20             endfor  {line 5 for}

21             parbegin $0 \le i' \le \sqrt{P}-1, 0 \le j' \le \sqrt{P}-1$ do
22                 $PE_{i',j'}$ writes $c_{(\sqrt{P}i+i'),(\sqrt{P}j+j')}$;
23             parend  {line 21 parbegin}
24         endfor  {line 4 for}
25     endfor  {line 3 for}
26 endprocedure
```

그림 2. HMSA에서의 행렬 곱셈 알고리즘
Fig. 2. Algorithm for matrix multiplication on HMSA

then every $PE_{i',j'}$ for all $0 \le i', j' \le \sqrt{p}-1$ multiplies the broadcasted operand by the second operand and accumulates the result product as shown in lines 14-16. After finishing the MAC (multiply-and-accumulate) operation, every $PE_{i',j'}$ for all $0 \le i', j' \le \sqrt{p}-1$ shifts up the second operand as shown in line 17. Finally, every $PE_{i',j'}$ for all $0 \le i', j' \le \sqrt{p}-1$ writes MAC operation results which are element $c_{i',j'}$ of submatrix $C_{i,j}$ $(0 \le i, j \le \sqrt{p}-1)$ in matrix C to their local memory block

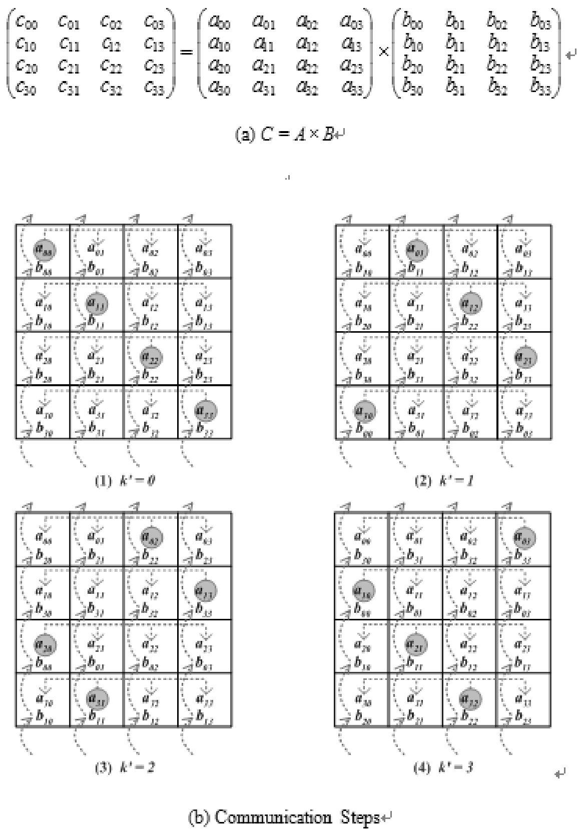


그림 3. C=A×B 행렬 곱셈의 데이터 이동
Fig. 3. Communication steps of matrix multiplication C=A×B

$PM_{i'j'}$ in parallel in line 22. The above processes are repeated until finishing the computation for the entire matrices. The communication steps in HMSA system for $C = A \times B$, with $N = 4$ and $P = 16$, is illustrated in Figure 3.

For the above processes, all local CUs in the HMSA system can concurrently broadcast each operand vector to those of PEs belonging to each local CU by RC-mode. Because the flexible control mode of HMSA system can be performed by CC-mode as well, it can effectively supported the operation which needs the transposed matrix multiplication. This efficient mechanism is applicable to any other linear algebra problem using the pattern similar to matrix multiplication. Therefore, the number of computation steps required to perform a matrix multiplication by HMSA can be obtained as

$$\left(\frac{N}{\sqrt{P}}\right)^2\left[\frac{N}{\sqrt{P}}\left(2T_{load}+\sqrt{P}(T_{broadcast}+T_{\mu lt}\right.\right. \quad \cdots(1)$$
$$\left.\left.+T_{add}+T_{send})\right)+T_{store}\right]$$
$$=4\frac{N^3}{P}+2\frac{N^3}{P\sqrt{P}}+\frac{N^2}{P}$$

In the above equation, T_{load} is the time to read the data from the memory block; $T_{broadcast}$ to perform operand broadcasting; T_{mult} to perform multiplication; T_{add} to perform addition; T_{send} to perform operand shifting; T_{store} to write the data to the proper memory block. Each of T_{load}, $T_{broadcast}$, T_{mult}, T_{add}, T_{send}, and T_{store} is assumed to take one unit time.

III. Performance Evaluation

Cannon's algorithm [14, 15] and Fox's algorithm [14, 16] are widely used as a parallel matrix multiplication. Here, they are applied to a torus and compared with the proposed algorithm in terms of the number of computation steps. The communication patterns of the two algorithms are shown in Table 1.

표 1. 통신방식
Table 1. Communication patterns

Algorithm	Cannon	Fox
Pre and Post processing	skewing of A and B	none
Computation of C	locally	locally
Movements of A	horizontal rollings	horizontal broadcast
Movements of B	vertical rollings	vertical rollings

First, Cannon proposed a memory efficient parallel algorithm, in which two $N \times N$ matrices A and B are divided into square submatrix of size among the P processors of a torus. In a phase of the algorithm execution, data in the two input matrices are aligned in such a way that the corresponding square submatrices at each processor can be multiplied together locally. In order to achieve such an alignment, the submatrix of A is rolled leftward and the submatrix of B is rolled upward. After these processes, the submatrices are multiplied and then the results are added to the partial results. Therefore, for the execution of this algorithm, the steps of performing dot product calculations, shifting matrix A to the west, and shifting matrix B to the north are required. The number of computation steps required to perform this algorithm can be obtained as

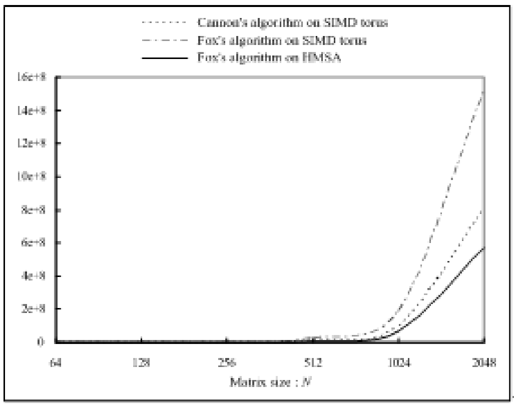
$$\begin{aligned}&(\frac{N}{\sqrt{P}})^2[\frac{N}{\sqrt{P}}(2(T_{load}+(\sqrt{P}-1)T_{send})\cdots \quad (2)\\&+\sqrt{P}(T_{\mu lt}+T_{add}+2T_{send}))+T_{store}\\&=6\frac{N^3}{P}+\frac{N^2}{P}\end{aligned}$$

Second, Fox's algorithm is another well-known memory-efficient parallel algorithm for multiplying dense matrices. Both $N \times N$ matrices A and B are partitioned among P processors so that each processor initially stores   blocks of each matrix. This algorithm uses one-to-all broadcasts of the blocks of matrix A in processor rows, and the single-step circular upward shifts of the blocks of matrix B along processor columns. Therefore, the steps of performing dot product calculation, broadcasting matrix A in a horizontal direction, and shifting matrix B to the north neighbors are required. Therefore, the number of computation steps required to perform a matrix multiplication can be found as

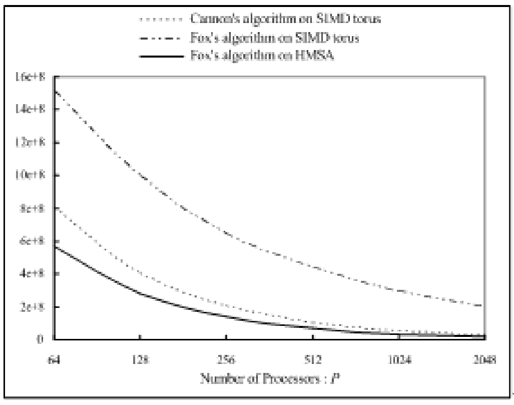
$$\begin{aligned}&(\frac{N}{\sqrt{P}})^2[\frac{N}{\sqrt{P}}(2T_{load}+\sqrt{P}(\sqrt{P}\,T_{broadcast}\cdots \quad (3)\\&+T_{\mu lt}+T_{add}+T_{send}))+T_{store}\\&=3\frac{N^3}{P}+2\frac{N^3}{P\sqrt{P}}+\frac{N^3}{\sqrt{P}}+\frac{N^2}{P}\end{aligned}$$

In the Equations (2) and (3), each of T_{load}, $T_{broadcast}$, T_{mult}, T_{add}, T_{send}, and T_{store} is assumed to take one unit time. This assumption was also made for the proposed algorithm. Under the same number of processors, the comparisons in terms of the number of computation steps are naturally fair.

Consequently, the number of computation steps on the HMSA with its corresponding algorithms is superior to those of the others, owing to the hierarchical construction of control units in multiple SIMD array architecture. This can significantly reduce the number of computation steps. According to Figure 4, the performance is improved in proportion to both the number of processors and the size of a matrix. When the number of processors is 64, Figure 4(a) shows the number of computation steps as the size of a matrix from 64 × 64 to 2048 × 2048. The number of computation steps increases drastically as the size of a matrix increases from 512 × 512 to 2048 × 2048. In Figure 4(b) when the size of a matrix is 2048 × 2048, the number of computation steps decreases drastically until the number of processor is about 512, and from about 512 processors to 2048 the number of computation steps decreases steadily.



(a) The computation steps as the matrix size: N changes (P = 64).



(b) The computation steps as the number of processors: P changes (N = 2048).

그림 4. Fox와 Cannon 알고리즘과의 성능비교

Fig. 4. Comparison with fox's and Cannon's algorithm

Therefore, both Figures 4(a) and (b) show that performance of HMSA is better than others. The HMSA system can reduce 29.1% ~ 62.2% of the number of computation steps required by Cannon's algorithm and Fox's algorithm mapped on the SIMD torus.

IV. Conclusion

This paper presented an efficient parallel matrix multiplication scheme on $N \times N$ mesh-connected SIMD array processor, called multiple hierarchical SIMD architecture (HMSA) with the features of . This architecture enables HMSA to have the features of diagonally indexed concurrent broadcast and the accessibility to either rows (row control mode) or columns (column control mode) of 2D array PEs alternately. An algorithmic mapping method is used for performance evaluation by mapping matrix multiplication on the proposed architecture. The asymptotic time complexities of them are evaluated and the result shows that parallel matrix multiplication on HMSA can provide significant performance improvement.

The result of performance evaluation is shown that the HMSA is effective to compute matrix-by-matrix and matrix-by-vector operations. Therefore, the HMSA provides a new platform for computing various computation-intensive data parallel applications.

[References]

[1] S. Akhter and J. Roberts, *Multi-Core Programming: Increasing Performance through Software Multi-threading*, Intel Press, 2006.

[2] K. Hirata and J. Goodacre, "ARM MPCore; The streamlined and scalable ARM11 processor core," *Proceedings the 2007 Asia and South Pacific Design Automation Conferenc,* pp. 747-748, 2007.

[3] K. A. Hawick, A. Leist, and D .P. Playne, "Mixing Multi-Core CPUs and GPUs for Scientific Simulation Software," Computer Science, Massey University, Tech. Rep. CSTN-102, 2009.

[4] M. Garland, "Sparse matrix computations on many-core GPU's," *Proceedings of the 45th ACM/IEEE Design Automation Conference*, pp. 2-6, 2008.

[5] D. E. Culler and J. P. Singh, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann, 1998.

[6] S. G. Shiva, *Pipelined and Parallel Computer Architectures*, HarperColling, New York, 1996.

[7] T. Blank, "The MasPar MP-1 architecture," *Compcon Spring '90. 'Intellectual Leverage'. Digest of Papers. Thirty-Fifth IEEE Computer Society International Conference*. pp. 20-24, March 1990.

[8] 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," *Proceedings of Custom Integrated Circuits '99*, pp. 631-634, 1999.

[9] R. Smith, K. Fant, D. Parker, R. Stephani and W. Ching-Yi, "An asynchronous 2-D discrete cosine transform chip," *Proceedings of Int'l Symp. Asynchronous Circuits and Systems*, pp. 224-233, 1998.

[10] N. Bagherzadeh, Chaves Filho, Guangming Lu, F.J. Kurdahi, M. H. Lee, and H. Singh, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *IEEE Transactions on Computers,* vol. 49 (5), pp. 465-481, May 2000.

[11] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro,* vol. 16 (4), pp. 42-50 , 1996.

[12] S. Tseng, Y. Kuo, Y. Ku, and Y. Hsu, "Software Viterbi Decoder with SSE4 Parallel Processing Instructions for Software DVB-T Receiver," *Proceedings of the 2009 IEEE Int'l Symposium on Parallel and Distributed Processing with Applications*, pp. 102-105, 2009.

[13] C. G. Kim, S. J. Lee, and S. D. Kim, "2-D Discrete Cosine Transform (DCT) on Meshes with Hierarchical Control Modes," *Lecture Notes in Computer Science*, 3522, pp. 675-682, 2005.

[14] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[15] L. E. Cannon, A Cellular Computer to Implement the Kalman Filter Algorithm, Ph. D. Thesis, Montana State University, 1969.

[16] G. C. Fox, S. W. Otto, and A. J. G. Hey, Matrix Algorithms on a Hypercube I: Matrix Multiplication, Parallel Computing, vol. 4, pp. 17-31, 1987.

Biography

Cheong Ghil Kim

1987 B.S. in Computer Science, at University of Redaldns, U.S.A.

2003 M.S. in Computer Science, at Yonsei University, Korea

2006 Ph.D. in Computer Science, Yonsei University, Korea

2006~ 2008 PostDoc and Research Professor at the Dept. of Computer Science, Yonsei University

2008 ~ Current Professor at Namseoul University

<Research Areas> Mobile Embedded Systems, Parallel Processing

<e-mail> cgkim@nsu.ac.kr