논문 98-02-02

Performance Comparison of Two Parallel LU Decomposition Algorithms on MasPar Machines

Young-Tae Kim*
(金 榮 泰*)

Abstract

This paper presents a performance study of two LU decomposition algorithms on two massively parallel SIMD machines: the 16K processor MasPar MP-1 and the 4K processor MasPar MP-2. The paper presents experimental results and an analysis of the algorithms to explain the results. While the *blocked* and the *nonblocked* algorithms for LU decomposition have been studied individually by others, we compare the two algorithms and identify the tradeoffs between them. Our analysis of the blocked algorithm shows how the block size affects the interprocessor communication cost and the memory read/write overhead. The analysis in this paper is useful to determine an optimum block size for the blocked algorithm.

I. Introduction

This paper studies two efficient parallel LU decomposition algorithms with partial pivoting on the MasPar machines with SIMD architecture: a *nonblocked* algorithm and a *blocked* algorithm. To understand different performance of the algorithms, we compare them and identify the tradeoffs between them. The parallel nonblocked and blocked algorithms perform identical floating point operations, but they have different memory read/write patterns and different communication patterns. The experimental results show the tradeoffs between the two algorithms, and the analysis explains the results and also helps in arriving at an optimal value of block size of the blocked algorithm.

The data partition described in this paper enables the two parallel LU decomposition algorithms efficient. Parallel algorithms for LU decomposition [2], [3], [4], [5], [6], [7], [9], [12] use either a one-dimensional *block* data partition which partions a matrix into row (column) submatrices or a two-dimensional *block* data partition which partions a matrix into submatrices along the row and column directions. The algorithms in this paper use a two-dimensional *scattered* data partition which partitions a matrix into submatrices and elements of each submatrix are evenly scattered among processors. This data partition provides better performance by evenly distributing the computations among all the processors.

We describe the algorithms in Section II, and present the performance comparisons and their analysis in Section III. Conclusions are in Section IV.

* 國立 江陵大學敎 컴퓨터科學科
(Dept.of Computer Science, Kangnung National Univ.)
接受日: 1998年7月20日, 修正完了日: 1998年12月21日

II. Description of the Algorithms

First, we briefly explain the data partition used in this paper and next, we describe the parallel nonblocked and blocked LU decomposition algorithms.

2.1 2-D Scattered data partition

Both algorithms use the two-dimensional *scattered* data partition for coefficient matrices. Fig 1 illustrates the 2-D scattered data partition of a 4×4 matrix A, onto a 2×2 processor array (PE). Fig 2 shows the submatrix of A stored on a single processor (PE0).

PE0	PE1	PE0	PE1
PE2	PE3	PE2	PE3
PE0	PE1	PE0	PE1
PE2	PE3	PE2	PE3

Fig. 1. 2-D Scattered partition of a 4×4 matrix on 2×2 PE array.

$a_{0,0}$	$a_{0,2}$
$a_{2,0}$	$a_{2,2}$

Fig. 2. Elements of the 4×4 matrix A on PE0.

2.2 Nonblocked LU decomposition algorithm

Assuming a matrix A is a $N\times N$ square matrix, the PE array is $P\times P$, and $N=M\times P$, the nonblocked LU decomposition algorithm has seven major steps as described in Fig 3. The k^{th} loop for the nonblocked algorithm illustrates for $N=16$ and $P=4$ in Fig 4. Each square represents a PE array (Fig 4 shows sixteen copies of a 4×4 PE array), and a small square within the square represents a processor. Two arrays l^k and u^k represent the k^{th} column of the L matrix and the k^{th} row of the U matrix, respectively. A^k represents the $(N-k-1)\times(N-k-1)$ submatrix of A where the upper left corner element is $a_{k+1,k+1}$ and the lower right corner element is $a_{N-1,N-1}$, and r^k represents the k^{th} row of A. To update the submatrix A^k, l^k is broadcasted to the right, and u^k is broadcasted down as shown in Fig 4. The update of A^k can now perform locally with no communication between PEs. Fox described a similar parallel LU algorithm for banded matrices in [4].

```
for k = 0 to N − 1 {
  find p such that |a_{p,k}| = max{|a_{k,k}, l^k|}; /* find the pivot row */
  temp = r^k;                                      /* interchange
  r^k = r^p;                                          the pivot row
  r^p = temp;                                         with the current row */
  broadcast r^p down;                              /* broadcast the pivot row */
  a_{k,k} = 1.0/a_{k,k};                           /* invert coefficient a_{k,k} */
  l^k = a_{k,k} * l^k;                             /* calculate and save multipliers */
  broadcast l^k right;                             /* broadcast multiplier column */
  A^k = A^k − l^k * u^k;                           /* update submatrix */
}
```

Fig. 3. The parellel nonblocked LU decomposition.

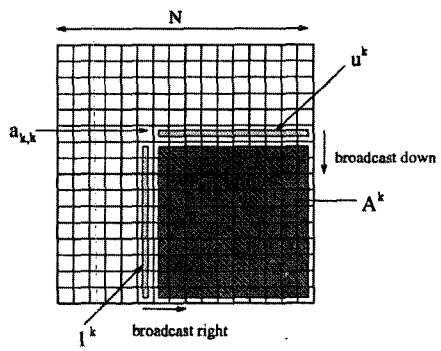

Fig. 4. The k^{th} step of nonblocked LU decomposition algorithm.

2.3 Blocked LU decomposition algorithm

Assuming that A is partitioned into B^2 submatrices (each of size $N_B \times N_B$), the blocked LU decomposition algorithm has B steps as shown in Fig 5. Each loop consists of three algorithms: (1) nonblocked LU decomposition, (2) triangular system solver, and (3) matrix-matrix multiplication. We briefly describe the three algorithms separately. Fig 6 illustrates the k^{th} loop for the blocked LU decomposition algorithm. Let L^k and U^k denote the k^{th} column-submatrix of L and the k^{th} row-submatrix of U, respectively. The nonblocked LU algorithm on the rectangular matrix $[A_{k,k}, L^k]^T$(see Figs 5 and 6) decomposes the matrix with partial pivoting. In this step, entire rows of A, not rows of $[A_{k,k}, L^k]^T$, are interchanged for the partial pivoting. The triangular system solver updates the matrix U^k. Figs 7 and 8 show the parallel triangular system solver used to update U^k. See the detail of the parallel triangular system solver in [10]. Let a^i be the subarray of $A_{k,k}$, and let B^i be the submatrix of U^k as shown in Fig 8. As was done in the nonblocked LU decomposition, a^i is broadcasted to the right, and b^i is broadcasted down. Then $B^i = B^i - a^i \times b^i$ can perform with no communication. The third step in the blocked algorithm is $A^k = A^k - matrix\_mult(L^k, U^k)$. We use a parallel matrix-matrix

multiplication routine for this step. Note that the blocked algorithm is same as the nonblocked algorithm when $B=1$. The correctness of the blocked LU decomposition algorithm is shown in [5].

```
for k = 0 to B − 1 {
   nonblocked_LU_decomp([A_{k,k}, L^k]^T);        /* rectangular matrix decomposition */
   U^k = triangular_system_solver(A_{k,k}, U^k);
   A^k = A^k − matrix_mult(L^k, U^k);
}
```

Fig. 5. The parellel blocked LU decomposition.

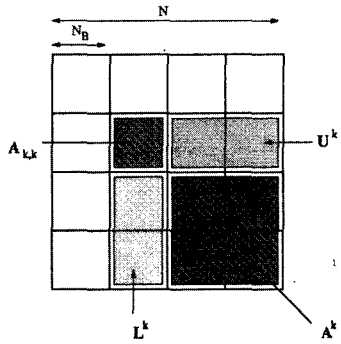

Fig. 6. The k^{th} step of blocked LU decomposition algorithm.

```
for i = 0 to N_B − 1 {
   broadcast a_i right;
   broadcast b_i down;
   B^i = B^i − a^i * b^i;
}
```

Fig. 7. The parallel triangular system solver.

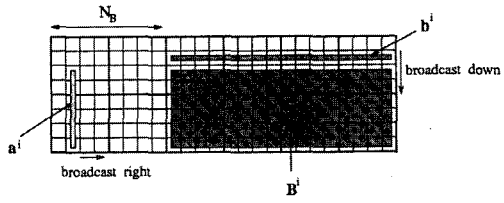

Fig. 8. The i^{th} step of parallel triangular system solver algorithm.

III. Performance Comparison

In this section, we compare the performance of the two algorithms by the experimental results on a MasPar 16K processor MP-1 and a 4K processor MP-2. To explain the results, we develop analytical timing formulas of analyzing the performance of the two algorithms. We first describe the MasPar machines and next discuss the performance comparison by communication cost, memory read/write overhead, and total execution time.

3.1 MasPar machines

First, we describe the relevant characteristics of the MP-1 and MP-2 machines to better understand the performance results on those machines. MP-2 processors can perform floating point operations four to five times faster than MP-1 processors, and communication and memory read/write on the MP-2 is one or two times faster than on the MP-1. Table 1 and 2 show measured cycle times for several instructions and communication [11].

Table 1. Memory read/write and floating point operation cycles on MasPar Machines.

	Operation	MP-1 Cycles	MP-2 Cycles
T_{load}	Load	85	40
T_{store}	Store	74	35
T_{mult}	Floating Point Multiply	225	41
T_{div}	Floating Point Division	325	75
T_{add}	Floating Point Addition	127	26
T_{neg}	Floating Point Negation	36	10
T_{cmp}	Floating Point Comparison	84	33

Table 2. Communication costs on MasPar machine.

General description		MasPar specific description		
routing scheme	communication cost	primitive (32-bit message)	communication cost MP-1	MP-2
Pipelined	$T_{X_s}+dT_{X_p}+mT_{X_t}$	`Xnetp[d]`	$58+d$	$48+d$
		`Xnatc[d]` (Copy)	$84+d$	$48+d$
Non-pipelined	$T_{X_s}+dmT_{X_t}$	`Xnet[d]`, $d=1$	43	40
		`Xnet[d]`, $d>1$	$19+35d$	$13+33d$

T_{X_s} : startup time
T_{X_p} : time to fill the pipeline
T_{X_t} : transmission time
d : distance
m : number of messages

3.2 Communication cost

Interprocessor communication cost is an important factor affecting the parallel performance. We analyze the communication costs for the two LU algorithms. The graphs in Figure 9 and 10 show the behaviors of communication cost of the two algorithms. The communication cost initially increases as the number of blocks increases and finally levels off. Note that starting points in the graphs represent the nonblocked algorithm since the algorithm has only one block. The two LU decomposition algorithms use different communication patterns. The nonblocked LU decomposition needs communication, which is a one-to-all broadcast along either a row or a column of processors, for pivoting and for broadcasting a pivot row and a multiplier column. The blocked decomposition uses nearest-neighbor communication in addition to the broadcast communication. We have used the Cannon's matrix multiplication algorithm in the blocked LU decomposition. The Cannon's algorithm uses the nearest-neighbor communication where elements are shifted from one processor to the next along either a row or a column with wrap-arounds at the end [1].

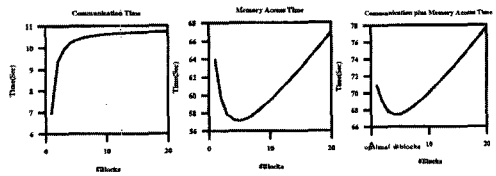

Fig. 9. The communication and memory read/write time comparison of problem size $5K\times5K$. on $16K$ MP-1(P=128).

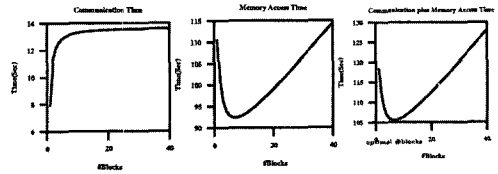

Fig. 10. The communication and memory read/write time comparison of problem size $5K\times5K$. on $16K$ MP-2(P=64).

The following formula determines the communication time for both algorithms:

$$f_{comm}(B) = P[\frac{1}{B^2}(M^2(-\frac{1}{2}mT_{Xt}-\frac{1}{2}T_{Xs}-\frac{1}{2}PT_{Xp}))$$
$$+\frac{1}{B}(M^2(-\frac{1}{2}mT_{Xt}-\frac{1}{2}T_{Xt}+\frac{3}{2}PT_{Xp})$$
$$+M(-\frac{1}{2}mT_{Xt}-\frac{1}{2}T_{Xs}-\frac{1}{2}PT_{Xp}))$$
$$+M^2(4mT_{Xt}+4T_{Xs}+PT_{Xp})+M((2(\log_2 P)+\frac{3}{2})T_{Xs}$$
$$+(2(P-1)m+\frac{3}{2}P)T_{Xp}+\frac{3}{2}mT_{Xt})]$$

Note that when $B=1$, the formula is for the nonblocked algorithm. We provide the detail derivation of the above formula in Appendix. The formulas are verified with the experimental results in [8].

3.3 Memory read/write overhead

Accounting for memory read/write overhead is important of the execution time if it is a significant portion. We analyze the memory read/write behaviors of the two LU decomposition algorithms. The graphs in Figs 9 and 10 show memory read/write behaviors in the two algorithms. The major difference comes from the "update submatrix" part in each step. As we have seen earlier, in the nonblocked algorithm, the submatrix $\boldsymbol{A}^k$ is updated by multiplication of the $(N-k-1)\times 1$ array $\boldsymbol{l}^k$ and the $1\times(N-k-1)$ array $\boldsymbol{u}^k$, and the totality of updating steps requires $O(N^3)$ operations. Thus, the efficiency of the algorithm depends upon the efficient "update submatrix" in each step. On the other hand, matrix-matrix multiplication plays an important role in the blocked algorithm since $\boldsymbol{A}^k$ is updated by the matrix-matrix multiplication $\boldsymbol{L}^k\times\boldsymbol{U}^k$ (see Fig 5). Note that the blocked algorithm makes use of matrix-matrix multiplication while the nonblocked algorithm uses vector-vector multiplication to update submatrices.

The following formula determines the memory read/write time overhead for the two algorithms:

$$f_{mem}(B) = P[B(M^2(\frac{1}{3}T_{load}+\frac{1}{3}T_{store}))$$
$$+(M^3(\frac{2}{3}T_{load})+M^2(5T_{load}+\frac{7}{2}T_{store})$$
$$+M(\frac{7}{3}T_{load}+(\frac{7}{6}+\frac{1}{P})T_{store}))$$
$$+\frac{1}{B}(M^3(\frac{1}{2}T_{store})+M^2(-\frac{1}{3}T_{load}-\frac{1}{3}T_{store})+M(-\frac{1}{2}T_{load}))$$
$$+\frac{1}{B^2}(M^3(-\frac{1}{6}T_{store})+M^2(-\frac{1}{2}T_{load}))]$$

Likely the communication formula, this formula calculates the memory read/write time for the nonblocked algorithm when $B=1$. Appendix includes the detail derivation of the formula. The formulas are verified with the experimental results in [8].

3.4 Analysis of total execution time

Figs 9 and 10 show communication time and memory access time in the blocked algorithm using $5K\times 5K$ size on MasPar MP-1 and MP-2 respectively. The right-most graphs in the figs also show the optimal number of blocks with the smallest communication cost plus memory read/write overhead. The graphs not only show how the the communication plus memory read/write overhead is different, but also are useful to determine the optimal block size for the blocked algorithm. The reduction of using the blocked algorithm with the optimal block size is significant on MP-2 while it is small on MP-1. This can be verified with the formulas given in the above sections and the values in the Tables 1 and 2. Since the two algorithms have the same number of floating operations, the different communication cost and memory read/write overhead affect the total execution time on the two m achines. The graphs in Fig 11 show comparison of the algorithms with three large matrix sizes on two MasPar machines. The blocked algorithm with the optimal block size outperforms the nonblocked algorithm on 4K

MP-2 while the two algorithms show almost identical performance on 16K MP-1. We also compare the two algorithms by total execution time and MFLOPS in Table 3.

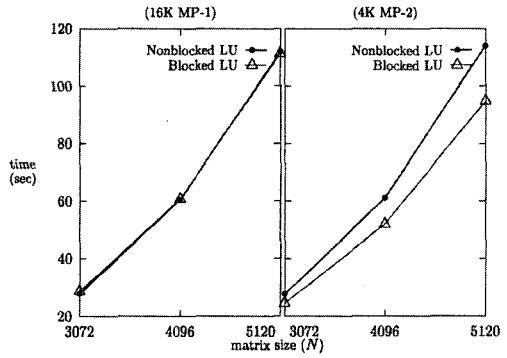

Fig. 11. The blocked algorithm outperforms the nonblocked algorithm on 4K MP-2 while they show almost identical performance on 16K MP-1.

Table 3. Excution time and MFLOPS.

N	16K MP-1				4K MP-2			
	execution time		MFLOPS		execution time		MFLOPS	
	non-blocked	blocked	non-blocked	blocked	non-blocked	blocked	non-blocked	blocked
1024	**1.89**	2.08	**190**	172	**1.71**	1.82	**210**	197
2048	**9.52**	10.02	**301**	286	9.39	**8.72**	305	**329**
3072	**27.89**	28.59	**347**	338	27.73	**24.47**	349	**395**
4096	**60.49**	60.71	**379**	377	61.10	**51.97**	375	**441**
5120	111.92	**111.34**	400	**402**	113.96	**94.71**	393	**473**

The bold-faces mean that the algorithm is faster than the other one.

IV. Conclusions

This paper supports three important conclusions: (i) We implement the nonblocked LU decomposition algorithm and next implement the blocked algorithm by introducing triangular system solver and matrix-matrix multiplication to the nonblocked algorithm. The 2-D scattered data partition for coefficient matrices makes these algorithms efficient. (ii) The blocked algorithm reduces memory read/write overhead by using matrix multiplication while it requires more interprocessor communication cost than the nonblocked algorithm. Thus, if the reduction of the memory read/write overhead is larger than the additional communication cost in the blocked algorithm, the blocked algorithm is faster than the nonblocked algorithm as we have seen the results on 4K processor MasPar MP-2. (iii) The comparison study in this paper helps us determine the optimal block size for the blocked algorithm by providing the minimal value of the memory read/write overhead reduction plus communication cost.

Appendix: Analytical Formula for the Parallel Blocked LU Decomposition Algorithm

$N\times N$: matrix size

$B\times B$: number of blocks

$N_B\times N_B$: block matrix size ($N_B=N/B$)

$P\times P$: PE array size

M : N/P

M_B : N_B/P

m : number of messages

$$\left(\frac{\textit{word size in bits}}{\textit{communication channel bandwidth in bits}}\right)$$

1. Parallel nonblocked LU decomposition (for the parallel nonblocked algorithm alone, plug $B=1$)

(a) Find pivot (local - sequential comparison)

$$\left(\sum_{i=B-1}^{0}\left(\sum_{j=iM_B+M_B}^{iM_B+1} j\right)\right)P(T_{load}+T_{cmp}+T_{neg}+T_{cmp})$$

(b) Find pivot (global - logarithmic comparison)

$$B(N_B((\log_2 P)(T_{Xs}+T_{cmp})$$
$$+(P-1)mT_{Xp})+N_B((\log_2 P)T_{Xs}+(P-1)mT_{Xp}))$$

(c) Interchange the pivot row with the current row and broadcast

$$B(2M_B^2P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}+\frac{P}{2}T_{Xp})$$
$$+\left(\sum_{i=M_B}^{1} i\right)P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}+PT_{Xp}))$$

(d) Coefficient inversion BN_BT_{div}

(e) Save inverted coefficient BM_BT_{store}

(f) Calculate(and save) and broadcast multiplier

$$\left(\sum_{i=B-1}^{0}\left(\sum_{j=iM_B+M_B}^{iM_B+1} j\right)\right)$$

$$P(T_{load}+T_{store}+T_{mult}+mT_{Xt}+T_{Xs}+PT_{Xp})$$

(g) Update submatrix

$$B\left(\sum_{i=M_B}^{1} i^{2)}\right)P(2T_{load}+T_{store}+T_{mult}+T_{add})$$

$$+\left(\sum_{i=B-1}^{0} i\right)\sum_{i=M_B}^{1} i)M_B$$

$$P(2T_{load}+T_{store}+T_{mult}+T_{add})$$

2. Parallel triangular system solver

(a) Broadcast $\boldsymbol{a}^k$ to right

$$\left(\sum_{i=B-1}^{1} i\right)N_B M_B(T_{load}+mT_{Xt}+T_{Xs}+PT_{Xp})$$

(b) Broadcast $\boldsymbol{b}^k$ to down

$$(B-1)P\left(\left(\sum_{i=M_B}^{1} i\right)(T_{load}+mT_{Xt}+T_{Xs}+PT_{Xp})\right)$$

(c) Update submatrix ($\boldsymbol{B}^k = \boldsymbol{B}^k - \boldsymbol{a}^k \times \boldsymbol{b}^k$)

$$\left(\sum_{i=B-1}^{1} i\right)\left(\sum_{i=M_B}^{1} i\right)M_B P(2T_{load}+T_{store}+T_{mult}+T_{add})$$

▷ $\boldsymbol{a}^k$ is kept in a register so that a T_{load} operation can be saved.

3. Parallel matrix-matrix multiplication

(a) Pre-skew input matrices(A and B)

$$2\left(\sum_{i=B-1}^{1} i\right)(M_B^2 P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}))$$

▷ For the shift communication on MasPar, we used xnet[1].

(b) Dot product calculation of a row of A and a column of B

$$\left(\sum_{i=B-1}^{1} i^2\right)M_B^2 P(M_B(2T_{load}+T_{mult}+T_{add})$$

$$+(T_{load}+T_{store}))$$

▷ $2T_{load}$s are for A and B .

(c) Shift each column of A to west

$$\left(\sum_{i=B-1}^{1} i\right)(M_B^2 P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}))$$

(d) Shift each row of B to north

$$\left(\sum_{i=B-1}^{1} i\right)(M_B^2 P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}))$$

(e) Post-skew A and B

$$2\left(\sum_{i=B-1}^{1} i\right)(M_B^2 P(T_{load}+T_{store}+mT_{Xt}+T_{Xs}))$$

References

[1] P. E. Bajørstad, F. Manne, T. Sorevik, and M. Vajtersic, "Efficient Matrix Multiplication on SIMD Computers," *SIAM J. Matrix Anal. Appl.*, vol. 13, no. 1, pp. 386-401, Jan. 1992.

[2] J. J. Dongarra, and D. W. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers," *SIAM Review*, vol. 37, no. 2, pp. 151-180, June 1995.

[3] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Vorst, *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, Philadelphia, 1990.

[4] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on concurrent Processors Vol. 1*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[5] T. Freeman, and C. Phillips, *Parallel Numerical Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[6] K. A. Gallivan, R. J. Plemmons, and A. H. Sameh, *Parallel Algorithms For Dense Linear Algebra Computations*, IAM, Philadelphia, 1990.

[7] G. A. Geist, and C. H. Romine, "LU Factorization Algorithms on Distributed-Memory Multiprocessor Architectures," SIAM J. Sci. Stat. Comput., vol. 9, no. 4, pp. 639-649, July 1988.

[8] Y. Kim, M. Fienup, J. Clary, and S. C. Kothari, "Parametric Micro-level Performance Models for Parallel Computing," Tech. Rep. TR94-23, Department of Computer Science, Iowa State University, 1994.

[9] G. Laszewski, M. Parashar, A. G. Mohamed, and G. C. Fox, "On the Parallelization of Blocked LU Factorization Algorithms on Distributed Memory Architectures," *Proceedings of '92 Conference on Super Computing*, 1992.

[10] G. Li, G. and T. F. Coleman, "A Parallel Triangular Solver For A Distributed-Memory Multiprocessor," *SIAM J. Sci. Stat. Comput.*, vol 9, no. 3, pp. 485-502, May. 1988.

[11] MasPar Computer Corporation, *MasPar Assembly Language Reference Manual*, Sunnyvale, CA, 1990.

[12] B. V. Purushotham, A. Basu, P. S. Kumar, and L. M. Patnaik, "Performance Estimation of LU factorization on Message Passing Multiprocessors", *Parallel Processing Letters*, vol. 2 no. 1. pp. 51-60, Mar. 1992.

저 자 소 개

金 榮 泰 (會員申請中)

B.S. of Mathematics, Yonsei University (1986). M.S. of Computer Science, Iowa State University (1992). Ph.D. of Computer Science, Iowa State University (1996). Research Associate, Iowa State University (1997). Full-time lecturer, Department of Computer Science, Kangnung National University (1998 ~).

Research Interest: Parallel and Distributed System, Parallel Compiler.