DBpia

論 文

대형 Sparse 선형시스템 방정식을 풀기위한 효과적인 병렬 알고리즘

正會員 蔡 洙 煥* 正會員 李 珍**

An Efficient Parallel Algorithm for Solving Large Sparse Linear Systems of Equations

Soo Hoan CHAE* Jin LEE** *Regular Members*

要 約 본 논문에서는 불규칙하게 분포된 non-zero 원소를 가진 대형 sparse 행렬로서 표시되는 선형시스템의 해를 능률적으로 얻기위한 반복 병렬 알고리즘에 대하여 기술하고, 이 알고리즘을 수행하는데 적절한 컴퓨터로서 dataflow 컴퓨터 구조를 제안하였다. 이 알고리즘에서는 Jacobi 반복법을 사용하였으며 행렬의 내적을 구하는데 소요되는 시간을 단축함으로서 병렬 수행시간을 단축시켰다.

ABSTRACT This paper describes an intelligent iterative parallel algorithm for solving large sparse linear systems of equations, and proposes a static dataflow computer architechture for the implementation of the algorithm. Implemented with the Jacobi iterative method, the intelligent algorithm reduces the parallel execution time by reducing the individual inner product operation time.

I. Introduction

Fast and efficient computer architectures are in high demand in many real-time application areas such as military research, medical research, and computer simulation. In order to satisfy the demand, parallel processing architecture has emerged as an important research field for computer professionals. Various kinds of parallel processing architectures have appeared in the past two decades [1-6]. The rapid advances in parallel processing architecture have stimulated much research aimed at developing parallel algorithms.

In solving linear systems of equations, the properties of linear systems determine what algorithm is appropriate for the problem. Many parallel algorithms have been developed for well structured cases such as banded matrices,

*韓國航空大學 電子計算學科
Dept. of Computer Science, Hankuk Aviation College
**韓國航空大學通信情報工學科
Dept. of Telecommunication and Information Eng., Hankuk Aviation College.
論文番號 : 89-37(接受 1989. 4. 19)

www.dbpia.co.kr

and symmetric matrices [7-9]. But, unstructured and sparse matrices are often found in network analysis and power system analysis. [10-11].

Without concerning the structure of matrices, the problem to solve large sparse linear systems of equations with diagonal dominant matrices is considered in this paper. In solving the problem, direct methods and iterative methods can be used. In general, iterative methods make more efficient use of memory space and time when solving large linear systems than direct methods [12]. Also, massive parallelism can be detected in iterative methods and global communicaton demands of iterative methods are fewer than those of direct methods [13]. Of iterative methods, Jacobi iterative method is chosen for implementing an intelligent algorithm for solving the problem because the characteristics of the Jacobi iterative method is inherently parallel.

Ⅱ. Jacobi Iterative Method

A linear system of equations can be expressed as

$$Ax=b, \qquad (1)$$

where A is a known $n\times n$ matrix, x is a unknown $n\times 1$ vector, and b is a known $n\times 1$ vector. The matrix A can be splitted into D-L-U, where D is the principal diagonal part, L is the strictly lower triangular, and U is the strictly upper triangular. Here, D, L, and U are $n\times n$ matrices. Eq. (1) can be transformed into

$$\begin{aligned}(D-L-U)x&=b\\ x&=D^{-1}(L+U)x+D^{-1}b\\ &=Tx+c, \qquad (2)\end{aligned}$$

where, $T=D^{-1}(L+U)$ and $c=D^{-1}b$.

When the matrix T and the vector c are partitioned into m blocks horizontally, the iterative method can be expressed as follows. Here, a norm, $||\cdot||$ is used for testing for the convergence.

```
Begin
k=0
while k<maximum-number-of-iterations
      do
for all 1≦i ≦m do
x_i^k = T_i *x^(k-1) + c_i
end for all
if |x^k - x^(k-1)| <tolerance then terminate
else x^(k-1)=x^k; k=k+1
end while
End
```

Two procedures, a computation procedure and a control procedure, are needed to realize the method. Main program reads the input data, initiates the procedures, and outputs the result, The computation procedure has the following functions:

* Calculate the inner product of the submatrix T_i with the vector x^{k-1}.
* Add the result of the inner product with c_i to obtain x_i^k.
* Send x_i^k to the control procedure.

The control procedure has the following functions:

* Send initial values x^0 if it is the first time, otherwise send the updated values, $x^{(k-1)}$ to the computation procedure.
* Collect the updated values, x_i^k from the computation procedure, and test for convergence. If convergence exists or the updated number of iteration exceeds the maximum limit, send this information to the main program and to the computation

www.dbpia.co.kr

procedure so that the execution is termin ated.

Ⅲ. Intelligent Iterative Parallel Algorithm

Massive parallelism can be detected in the inner product operation of Jacobi iterative method. Data granularity is the main point of view in the intelligent iterative parallel algorithm (is called the intelligent algorithm in the followings). The given matrix and vector, T and b can be divided into units of various sizes.

1) Data granularity

* Large-grain implementation: the basic unit in the large-grain implementation is a submatix. The matrix T is partitioned into several submatrices horizontally. Each processor calculates the inner product of a submatrix, T_i of T with $x^{(k-1)}$, and adds the result and the corresponding known subvector, c_i of c in order to obtain x_i^k, x^k is obtained by collecting all subvectors from each processor.
* Medium-grain implementation: Each row of the matrix, (T) is a basic unit of the medium-grain implementation Each processor executes the inner product of a row vector of T with $x^{(k-1)}$, and adds the result to the corresponding element of the known vector(c) to obtain an element of x^k. x^k is obtained by collecting all elements from each processor.
* Fine-grain implementation: The basic unit of the fine-grain implementation is a scalar operation such as an addition operation or a multiplication operation. Each vector operation is partitioned into many scalar operations.
* Proposed implementaion: The fine-grain implementation exploits massive parallelism, but communication delays may be serious to do vector operations. The large-grain or the medium-grain implementation cannot detect massive parallelism comparing to the fine-grain implementation, but they do not require high communication demand among processors for vector operations. The intelligent algorithm determines the data granularity of each row according to the given conditions (number of the nonzero elements in the row, communication delay, and start-up time of pipelined processors). In other words, the intelligent algorithm divide a row vector into one or more subvectors, or scalars depending on the given conditions. Therefore, the proposed implementation (rule based method) is between the fine-grain implementation and the medium-grain implementation.

2) Procedure

The procedure of the intelligent parallel algorithm consists of a broadcast phase which send global data to each processor, a compute phase, an aggregate phase which collect local data into one global data, and a test phase for global convergence (Fig. 1).

The main idea of the algorithm is to find optimal data granularity for the inner product operation under the following assumptions:

(1) The execution times (t_e) of a scalar operation such as addition and multiplication are same.

(2) The speeds of stages in a pipeline are equal.

(3) The clock period of the inner product pipeline which is consists of a multiplication part and an addition part is $2(t_e / k_s)$, where k_s denotes

www.dbpia.co.kr

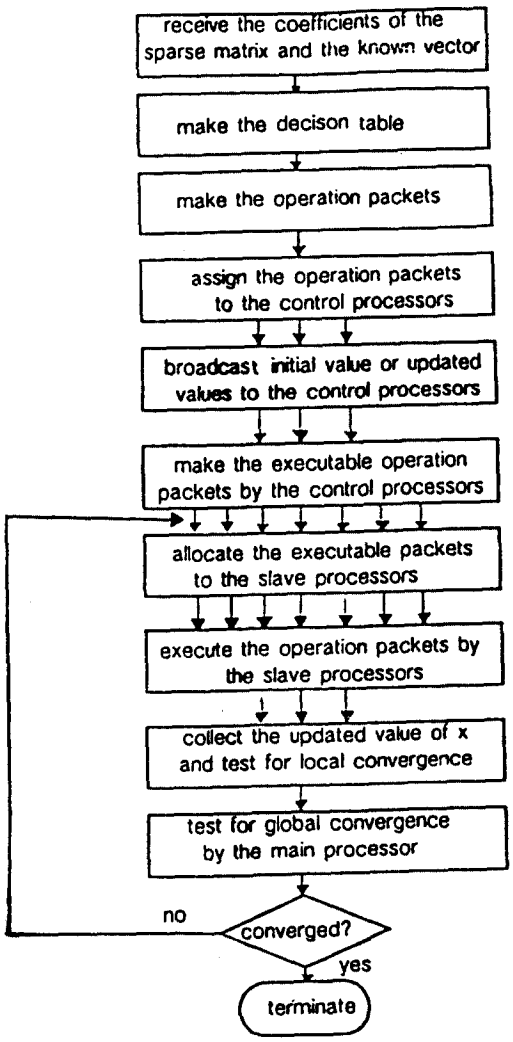


Fig. 1. Flow-chart of the intelligent algorithm

the number of stages fo the pipeline. Vector operations can be processed on a pipeline processor or many scalar processors in parallel. When pipelining is applied to vector processing, set-up time (time taken to structure the pipeline and prepare the streams of vector operands) and flush time (time to empty the pipeline at the conclusion of the operations and set the termination condition) are required. Here, the overhead time is called start-up time.

If a vector operation with length p is executed on a pipeline with k_s stages and the clock period of the pipeline is t, the execution time required is $(p+k_s-1)\ t$[14]. When pipelined processors are used for implementing the medium-grain approach (temporal method), the execution time of the inner product on a pipelined processor (t_{pp}) is obtained by

$$t_{pp}=\text{start-up time}+(p+k_s-1)\ 2t_e/k_s, \quad (3)$$

where p denotes the vector length.

For implementation of the fine-grain approach, many scalar processors are simultaneously used to do inner product (spatial method). In this case, after each processor finishes its operation, the result is sent to another processor to do next operation. Therefore, communication time is an important factor in counting parallel execution time. Assuming that all scalar processors execute binary operations, the parallel execution of the spatial method to do an inner product operation of vector a and b with length, p is shown in Fig. 2.

Therefore, the execution time of the fine-grain approach (t_{sp}) is obtained by

$$t_{sp}=(\lceil \log_2 p \rceil)t_o+(\lceil \log_2 p \rceil+1)\ t_e, \quad (4)$$

where, to denotes the communication time.

Depending on the given conditions, the algorithm chooses the temporal method, spatial method, or rule-based method by calculating the execution times required. The intelligent algorithm creates a decision table for given range of the row vectors of the matrix T to reduce the time that decides the optimal length of each row vector for each iteration. The decision table is used for a rule base.

www.dbpia.co.kr

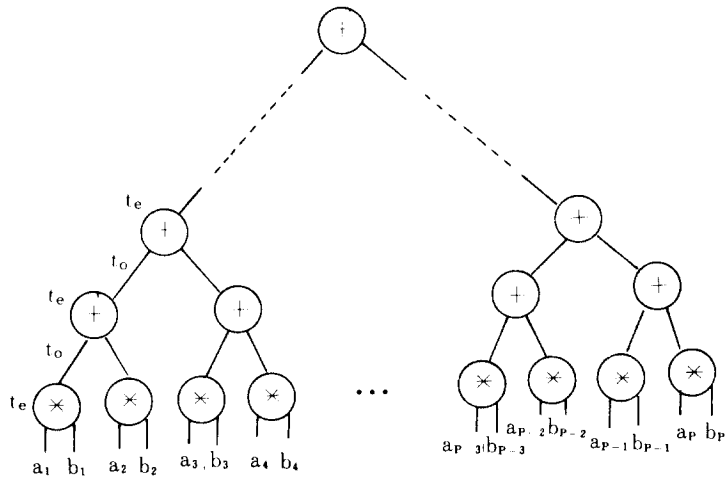


Fig. 2. Inner product operation of vector a and b

The procedure to make the decision table is shown in the followings:

```
for p=minimum-length to maximum-length do
for k=1 to ⌈ log2 p ⌉ do
expand-binary-tree;
calculate-parallel-execution-time
choose-best-case;
end for k
end for p
```

The expand-binary-tree routine expands the binary tree by splitting a row vector until it cannot be divided by the following algorithm:

```
if (p(k) mod 2) is 0 then p(k+1)1=p(k) / 2;
                          p(k+1)2=p(k) / 2
                     else p(k+1)1=p(k) / 2;
                          p(k+1)2=p(k) / 2+1,
```

where k is the number of iteration.

The calculate-parallel-execution-time routine calculates the parallel execution times for all cases made by the expand-binary-tree routine using Eq. (3), Eq. (4), or both equations. The choose-best-case routine compares the execution times and chooses the best case.

The intellignet algorithm decides the data granularity based on the decision table. According to the decision table, if it is more efficient to use a row vector with whole length, the algorithm do not divide the row vector and make a vector operation packet. If it is more efficient to divide the row vector into individual scalar elements, the algorithm makes many scalar operation packets. If it is best to divide a row vector into several subvectors, the algorithm divides the row vector into several subvectors and makes several vector operaion packets and several scalar operation packets.

Ⅳ. Evaluation

The parameters in evaluation are the order (size) of sparse matrices, sparse factor (rate of nonzero elements for all elements), start-up time of the pipelined processor, and communication delay of packet communication network.

The parameters are given at the testing

www.dbpia.co.kr

time. A random number generator of IMSL is used to make sparse matrices for simulation according to the given sparse factor.

The intelligent algorithm (rule-based method) is compared to the medium-grain implementation (temporal method) and the fine-grain implementation (spatial method) by simulation. The sizes of sparse matrices varies from 300 ×300 to 2100×2100 with the interval, 200 and it is assumed that the number of stages of the inner product pipeline is 8.

1) Execution time

The execution time of inner product operations for an iteration, which was simulated sequentially, is shown in Fig. 3 and Fig. 4. When the start-up time becomes large, the execution time of the rule-based method approaches that of the temporal method. Conversely, if the communcation delay becomes large, the execution time of the rule-based method approaches to that of the spatial method. However, the rule-based method is the best in the execution time.

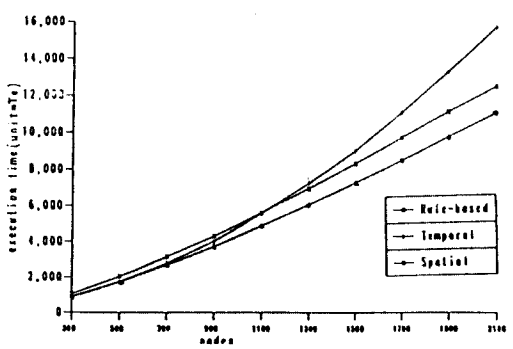


Fig. 3. Execution time on the conditions; sparsity factor= 0.01, t_o / t_e= 0.0, and start-up time=0.0

When the number of processors is unlimited, the parallel execution time simulated is shown

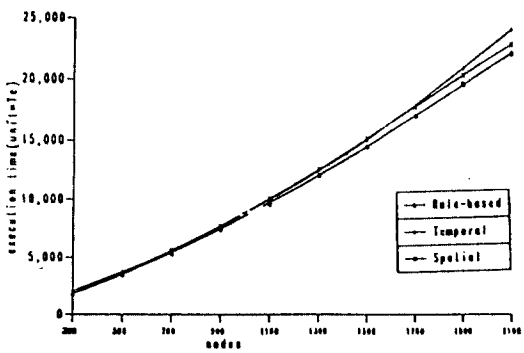


Fig. 4. Execution time on the conditions; sparsity factor= 0.01, t_o / t_e=1.0, and start-up time=4.0

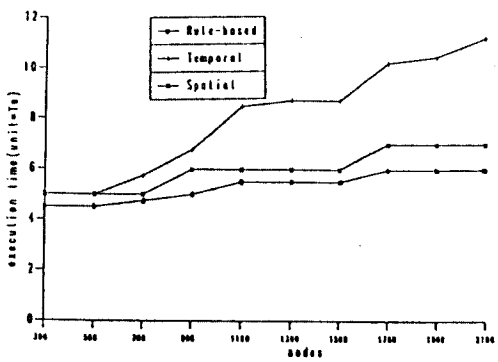


Fig. 5. Parallel execution time on the conditions; sparsity factor=0.01, t_o / t_e=0.0, and start-up time=0.0 when the number of procesors is unlimited

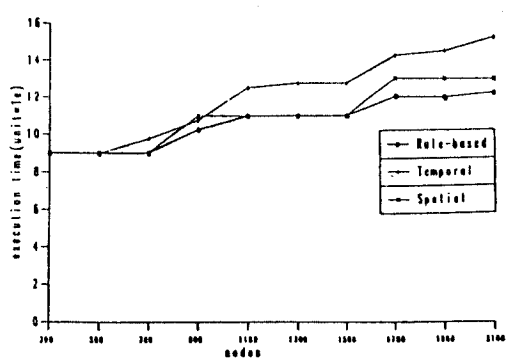


Fig. 6. Parallel execution time on the conditions; sparsity factor=0.01, t_o / t_e=1.0, and start-up time=4.0 when the number of processors is unlimited

www.dbpia.co.kr

in Fig. 5 and Fig. 6. The parallel execution time is decided by the execution time of the row with the maximum number of nonzero elements for the temporal method and the spatial method. But this phenomenon does not occur in the rule-based method, since the row vector may be divided into several subvectors. Therefore, the parallel execution time of the rule-based method is less or equal to those of the other methods.

2) Speed-up rate

The speed-up rate is defined as $s_p = t_{seq} / t_{par}$, where t_{seq} and t_{par} denote sequential execution time and parallel execution time, respectively. The speed-up rate is shown in Fig. 7 and Fig. 8. From this evaluation, the rule-base method becomes more advantageous as sparse matricxes become larger.

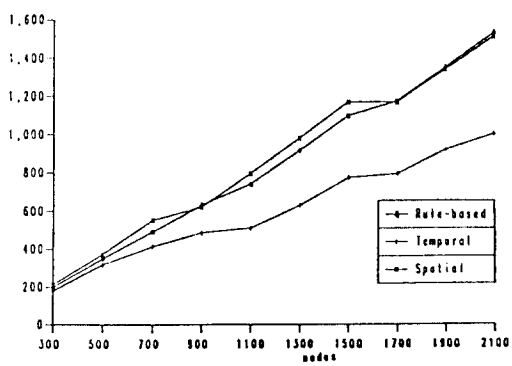


Fig. 7. Speed-up rate on the conditions: sparsity factor= 0.01, t_o / t_e=0.0, and start-up time=0.0

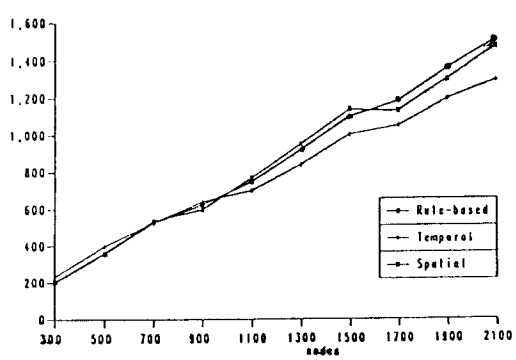


Fig. 8. Speed-up rate on the conditions: sparsity factor= 0.01, t_o / t_e=1.0, and start-up time=4.0

V. Proposed Computer Architecture

The intellignet iterative algorithm exploits massive parallelism in the inner product operation and requires scalar operations and vector operations. Therefore, heterogeneous operation and asynchronous operation are critical factors in choosing the proper architecture for the implementation of the massive paralellism.

Of many parallel computer architectures, dataflow computer architectures have the capability not only to suport massive parallelism exploited but to execute operation packets asynchronously. Dataflow computer architectures are based on asynchrony and functionality to do parallel operation. Asynchrony results from the execution mechanism in which an operation is executable when all input data tokens of execution graph have arrived. In other words, the conventional parallel architectures are based on the ordered execution of instructions with control flow while dataflow computer architectures are based on the availability of the operands without any control flow. Functionality results from the mechanism that permits any two enabled operations to be executed concurrently without side-effects, as the mechanism of functions.

Dataflow computer architectures are classified as static architectures and dynamic architectures. In static architectures, instructions are loaded into memory before the computation begins and each instruction is executed once immediately after its operands have arived. In dynamic architecture, an instruction can

www.dbpia.co.kr

be executed several times because each instruction is created at running time by tagging. Therefore, additional hardware is required to attach and to match tags of reentry.

The directed graph from the Jacobi iterative method can be acyclic for an iteration. Therefore, each operation can have a unique address which is assigned by the compiler. This causes that only one operator exists on an input link of the dataflow graph. Hence, static dataflow computer architectures are suitable to implement the intelligent algorithm.

In static dataflow computer architectures, a number of processores are conected through

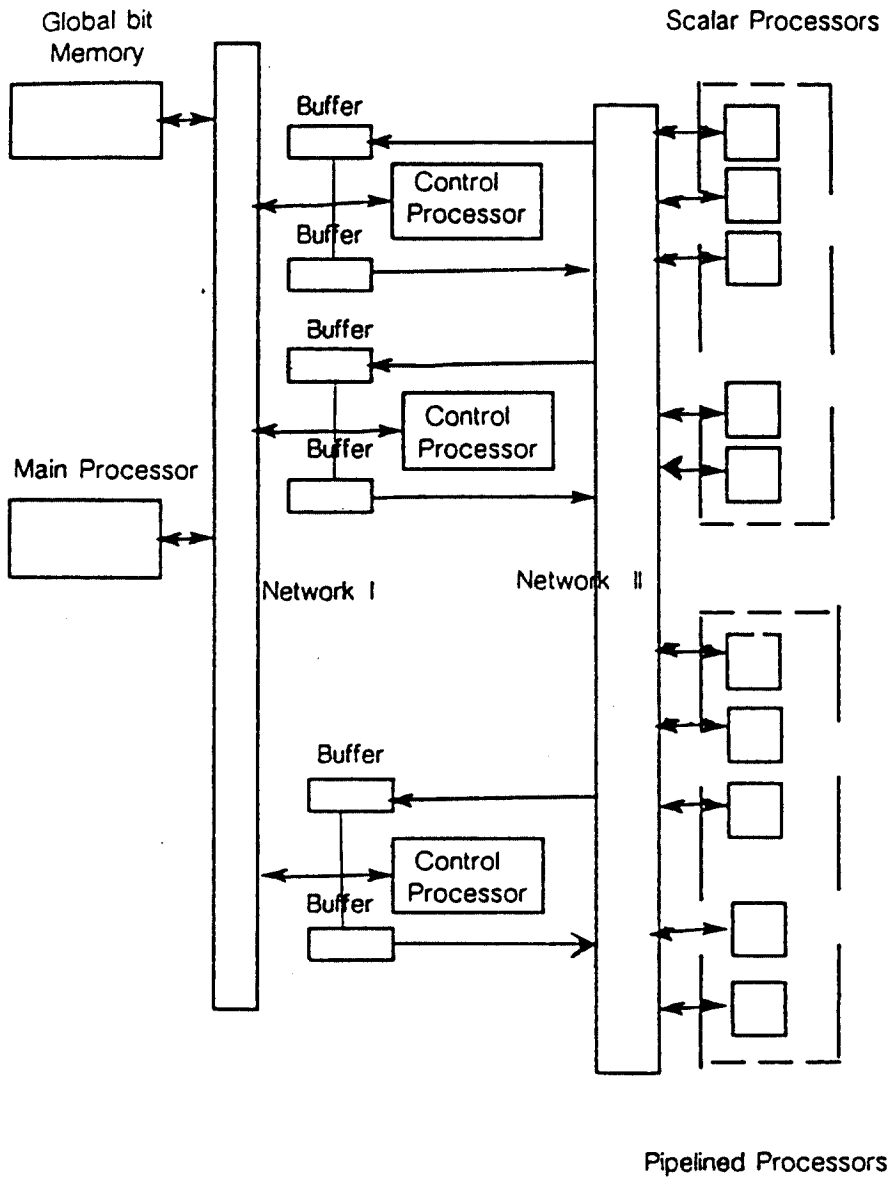


Fig. 9. System configuration

www.dbpia.co.kr

a routing network. A dataflow program is divided into individual instructions to be distributed over the processors. The instructions are stored in the memory sections of processors. Each instruction becomes executable when its operands arrive. Each processors executes the executable instructions and sends the result to the designated next processor through the routing network.

The intellingent alogorithm creates scalar operation packets and vector operation packets. Therefore, the pipelined vector processor pool and the scalar processor pool are needed to do heterogeneous operations as shown in Fig. 9. Main processor and control processors executes the control procedure. Slave processors in the processor pool accomplish the computation procedure. The configuration shown in Fig. 9 is proposed to distribute the burden of the main processor to the control processors and to improve the utilization of slave processors. The intelligent algorithm requires synchronization after an iteration for testing for global convergence. In order to reduce the time required for the rendezvous operation, buffers are used between a sending processor and a receiving processor

Control processors test for local convergence and send the result to the main processor. The result in information is whether it is converged or not. Therefore, it is desirable to use a global bit memory in order to reduce the packet communication time between the control processors and and the main processor.

Ⅵ. Conclusion

The exploitation of parallelism occupies a key position in parallel procesing systems. The other important factor to be considered in parallel processing systems is communication delay. The intelligent algorithm is a method to optimize two factors based on the given conditions in order to reduce parallel execution time and the proposed computer architecture is a proper parallel computer architecture in solving linear systems of equations. The idea can be extended to solve nonlinear systems of equations.

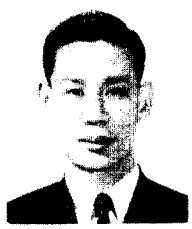
參 考 文 獻

1. David J. Kuck and Richard A. Strokes, "The Burroughs Scientific Processors", IEEE Trans. Comput., pp. 363-375, May 1982.
2. Kai Huang, "Advanced parallel processing with supercomputer architecture", Proc. IEEE, pp. 1348-1378, Oct. 1987.
3. H.T. Kung, "Why systolic architecture?," Computer, pp.37-46, Jan. 1982.
4. Vason P.Srini, "An architectural comparision of dataflow systems," Computer, pp. 66-88, Mar. 1985
5. J.R.Gurd. C.C. Kirkham, and I. Watson, "The Manchestor prototype dataflow computer," Comm. ACM, pp. 34-52, 1985.
6. Philp C. Treleaven, David R, Brownbridge, and Richard P. Hopkins, "Data-driven and demand-driven computer architecture," Computing Surveys, pp. 93-143, Mar. 1982.
7. Torstein Opsahl and dennis Parkinson, "An algorithm for solving sparse sets of linear equations with almost tridiagonal structure on SIMD computers," Proc. Int'l. conf. Parallel Processing, pp. 369-374, 1986.
8. H.H. Wang, "A parallel method for tridiagonal equations," ACM Trans. Math. Soft. pp. 170-183, Jan. 1981.
9. Guang R. Gao, "A pipelined soultion method of tridiagonal linear equation systems," Proc. Int'l. Conf. Parallel Processing, pp.84-91, 1986.

www.dbpia.co.kr

10. H. Taoka, S.Abe, "Fast solution of sparse network equations with an array processor," IEEE Tran. Power Systems, pp. 215-221.
11. E. Chiricozzi and A. D'Amico (Ed), Parallel Processing and Applications, North-Holland, pp. 3-19, 1988.
12. Richard L. Burden, J. Douglas Faires, and Albert C. Reynolds, Numerical Analysis, Prindle, Weber & Schmidt, 1985.
13. Leah H. Jamieson, Dennis B.Gannon, and Robert J.Douglass, (Ed), The Characteristics of Parallel Algorithms, MIT Press, 1987.
14. Jean-Loup Baer, Computer Systems Architecture, Computer Science Press, 1980.

蔡 洙 煥(Soo Hoan CHAE) 正會員
1950年10月28日生
1973年 2 月：韓國航空大學 電子工學科 卒業(工學士)
1973年 7 月～1977年 8 月：空軍技術將校
1977年 8 月～1983年 7 月：金星通信研究所
1983年 8 月～1985年 5 月：美國알라바마大學電算學科(理學碩士)
1985年 5 月～1988年12月：美國알라바마大學電氣工學科 (工學博士) (電子計算機構造專攻)
1989年 3 月～現在：韓國航空大學 電子計算學科 助教授



李 珍(Jin LEE) 正會員
1935年 5 月17日生
- 韓國航空大學通信工學科 卒業
- 漢陽大學校 大學院(工學碩士)
- 慶熙大學校大學院(工學博士)
- 現在：韓國航空大學通信情報工學科教授, 本學會 副會長

www.dbpia.co.kr