실시간 영상 복원을 위한 분산 전기단층촬영 알고리즘

(A Distributed Electrical Impedance Tomography Algorithm for Real-Time Image Reconstruction)

이 정 훈 [†]　　박 경 린 [††]
(Junghoon Lee)　(Gyunglin Park)

요 약 본 논문은 전기단층촬영의 실시간 영상 복원을 위한 마스터-슬레이브 구조를 갖는 분산 전기단층촬영 알고리즘을 제안하고 그 성능을 평가한다. 영상복원은 그 수행시간이 미지수의 수에 3제곱에 비례하는 계산 위주의 응용으로서 영상의 정밀도를 위해 미지수를 증가시키면 그 수행시간이 급격히 증가한다. 마스터는 순차적인 루프에 진입하기 전에 각 컴퓨팅 노드에 독립적인 프레임 데이터를 분배하여 병렬로 기저노드를 추출하도록 하고 그 결과를 취합하여 그룹화함으로써 미지수의 수를 감소시킨다. 지역망으로 연결된 컴퓨팅 노드들은 MATLAB이 설치되어 기본적인 계산능력을 갖고 있으며 MATLAB 자료구조를 효율적으로 교환할 수 있는 명령이 동적 링크 라이브러리로 구현되어 있다. 또한 마스터에는 병렬 행렬 연산, 고속 자코비언 등이 구현되어 순차적인 부분의 계산을 효율적으로 수행한다. 구현된 각 요소들의 성능을 측정한 결과 병렬 라이브러리는 전체 복원 시간을 50% 가까이 감소시킬 수 있으며 분산 알고리즘은 4개의 노드가 협력작업을 하는 경우 주어진 대상 물체에 대해 12배 빠른 속도로 영상을 복원할 수 있다.

키워드 : 실시간 영상복원, 클러스터 컴퓨팅, 분산 알고리즘, 병렬 라이브러리, 협력 그루핑

Abstract This paper proposes and measures the performance of a distributed EIT (Electrical Impedance Tomography) image reconstruction algorithm which has a master-slave structure. The image construction is a computation based application of which the execute time is proportional to the cube of the unknowns. After receiving a specific frame from the master, each computing node extracts the basic elements by executing the first iteration of Kalman Filter in parallel. Then the master merges the basic element lists into one group and then performs the sequential iterations with the reduced number of unknowns. Every computing node has MATLAB functions as well as extended library implemented for the exchange of MATLAB data structure. The master implements another libraries such as threaded multiplication, partitioned inverse, and fast Jacobian to improve the speed of the serial execution part. The parallel library reduces the reconstruction time of image visualization about by half, while the distributed grouping scheme further reduces by about 12 times for the given target object when there are 4 computing nodes.

Key words : real-time image reconstruction, cluster computing, distributed algorithm, parallel library, cooperative grouping

1. Introduction

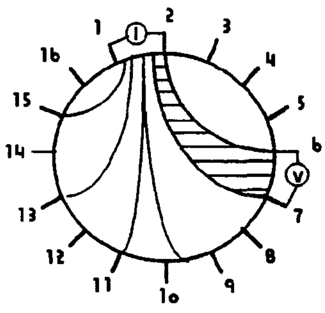
Over the past few decades, EIT(Electrical Impedance Tomography) techniques have received much attention from both theoretical and experimental points of view since they can be used as an alternative imaging modality for monitoring tool in many engineering fields[1]. This is mainly due to the relatively cheap hardware requirements, noninvasive measurement manner in sensing the signal, and reasonable temporal resolution. The application field includes heat exchangers, oil or natural gas pumping systems, fluidized beds and so on[2]. The basic assumption of the conventional EIT is that

† 정 회 원 : 제주대학교 전산통계학과 교수
jhlee@cheju.ac.kr
†† 비 회 원 : 제주대학교 전산통계학과 교수
glpark@cheju.ac.kr

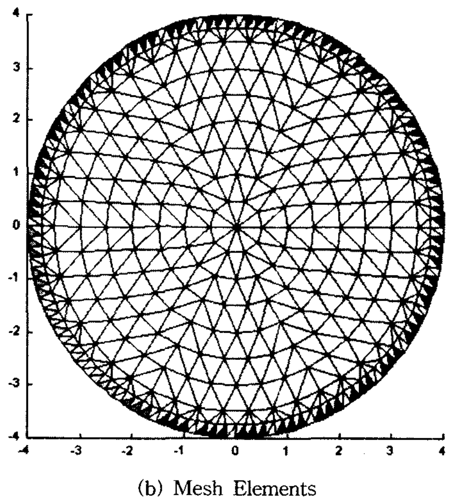
논문접수 : 2003년 4월 11일
심사완료 : 2003년 11월 5일

the distribution of internal electrical properties is stationary during the acquisition phase of full data set. To collect the necessary data, several electrodes are attached along the surface of the target object. Figure 1(a) demonstrates that 16 electrodes are evenly located around the surface of circular target.

Predefined current patterns are injected through each electrode one by one or simultaneously. For an injection from a node, the other electrodes measure the voltage values, respectively. The obtained data set builds up the equation system to be solved for image reconstruction. Based on these data, the reconstruction procedure virtually segments the cross section of the target object into a number of tiny elements as shown in Figure 1(b) and calculates their resistivities. That is, each element is an unknown variable that should be solved while the measured voltages decide the coefficients of a differential equation system.



(a) Electrode distribution



(b) Mesh Elements

Figure 1 The principle of EIT

Element size determines the resolution factor of the visualized image. However, finer element partition does not always produce a better image as EIT image reconstruction is a nonlinear ill-posed inverse problem that makes it difficult to obtain a stable and reliable result. As a result, the number of electrodes mainly affects the determination of element size. In addition, as the reconstruction procedure involves various time-consuming matrix operations such as multiplication, inverse, and Jacobian, the computation time cannot possibly catch up with the data acquisition speed, to say nothing of real-time visualization. If there are n elements in the mesh, most data structures are $n*n$ matrices, and the computation time of above-mentioned operation is known as $O(n^3)$. To the worse, when the algorithm is implemented with software tool such as MATLAB without any support of specific hardware, the computation time becomes too long.

For efficient and fast image reconstruction, the performance improvement should be pursued via parallel or distributed computing technology as well as design of a new algorithm[3]. If we tolerate some degradation in the quality of the reconstructed image, fast reconstruction scheme can be designed as an instance of the imprecise algorithm[4]. The imprecise computation technique is a way to improve the scheduling feasibility of applications where results of poor quality are better than late results. The imprecise computation model assumes that the time constraint of each task is given and that the quality of a task's result is solely dependent on the time and resources spent to produce the result.

This paper describes and proposes performance enhancement techniques for EIT, aiming at improving the speed of image reconstruction. The work includes an implementation of parallel numerical library and a proposition of distributed EIT algorithm where each node cooperates to reduce the number of elements, namely, that of unknowns. This paper is organized as follows: After issuing the problem in Section 1, Section 2 introduces the background concept of the EIT image reconstruction scheme. Section 3 focuses on the extended libraries for parallel and distributed processing. Section 4 proposes a cooperative grouping scheme on top of the extended functions. After the results are analyzed in Section 5, Section 6 finally summarizes and concludes this paper.

2. Overview of EIT image reconstruction

The EIT problem is essentially equivalent to the differential equation system where each element is unknown variable. As natural, the system may be either overdetermined or underdetermined, so it is usually impossible to find a unique solution, especially when the measured voltages contain a certain type of noise. Hence, most published algorithms belong to the iterative one and they carry out as follows repeatedly with an tentative guess of unknowns:

1. calculates how well the solution fits
2. decides how to adjust the solution set
3. modifies the current solution

This procedure will continue until a reasonable solution is found or within a limited number of iterations. Iteration limit eliminates the possibility of infinite loop, which happens when the estimation diverges as the iteration goes on. The first step is straightforward, and can be performed by substituting the current solution to the system. There exist a several number of decision criteria for this purpose, but RMSE(Root Mean Square Error) scheme is generally used. In the context of EIT, for ρ, a set of resistivity values for the respective elements, it is possible to calculate the voltage anticipated to be measured at each electrode. This is the forward solver and it is generally based on the finite element method observing the complete electrode model[5]. The second step is specific to each algorithm and a noticeable way is to use Jacobian matrix, analogous to a first order derivative. The third step also varies by the respective algorithms ranging from a simple addition to a combination of complex matrix operations. With respect to EIT system, step 2 and 3 can be described as follows: To solve the inverse problem we have used an approach in which we minimize the functional

$$\Psi(\rho) = |U - U(\rho)|^2 + \alpha^2 |L(\rho - \rho^*)|^2 \quad (1)$$

for the resistivity ρ. As a result, the present problem is to find the resistivity distribution both satisfying the above constraint and minimizing the difference between the calculated boundary potential, $U(\rho)$, and the actually measured boundary potential, U. Additionally, L is a regularization matrix, while α is a regularization constant and ρ* is an a priori guess for ρ. The solution is searched iteratively by

$$\rho_{i+1} = \rho_i + \delta\rho_i \quad (2)$$

where $\delta\rho_i$ is solved from

$$(J_i^T J_i + \alpha^2 L^T L)\delta\rho_i = J_i^T (U - U(\rho_i)) - \alpha^2 L^T L(\rho_i - \rho^*) \quad (3)$$

In (3), J denotes the Jacobian matrix of ρ with respect to U.

As can be inferred from (3), the reconstruction procedure accompanies many matrix operations that need severe computations, the EIT community has been urgently pursuing an efficient way to reduce tremendous reconstruction time, especially via parallel or distributed processing. However, there are several difficulties in developing an efficient scheme as the general tomography algorithms have strong data dependencies between the iterative loops and they contain so many 2-dimensional matrix operations. The data dependency can be partially solved by revising a tomography algorithm, but even a small modification may incurs an additional reconstruction error due to its sensitivity to resistivity model[6]. Another approaches try to reduce the computation time parallel processing on matric multiplication and inverse[7]. Though they do not infect the correctness of algorithm, their enhancement is limited by expensive hardware in case of parallel processing, by network overhead and so on.

3. The implementation of parallel and distributed functions for MATLAB

Recently, high-level languages such as MATLAB have become popular in prototyping a complex algorithm in domains such as signal and image processing[8]. MATLAB provides a very high level specification in a functional style and a rich set of built-in matrix manipulation functions. Therefore, we have implemented both our own and already published algorithms. This code includes Newton-Raphson as well as various Kalman Filters. On the other hand, it is convenient for us to develop parallel and distributed computing primitives using C programming language, as they are closely related with the underlying operating system. In addition, there exists a case where C implementation is more efficient and convenient than MATLAB implementation, especially when we should use system call.

To resolve this conflict, MATLAB opens the way for a linkage of itself with a communication library, which is usually implemented in C[9]. Figure 2 shows the principle how to extend MATLAB to a distributed and parallel environment. Prerequisite is programming interface to external software. The supported platforms range from Windows to UNIX. The function coded in C programming language is compiled into a DLL(Dynamic Link Library) via MEX utility. We will call this kind of program as *mex routine*, and it can be directly called from the MATLAB interpreter or M-file, relieving the burden of any modification on the already developed code. The mex routine should be able to correctly handle the matrix representation mechanism of MATLAB package, as its arguments and return values are exchanged accordingly.

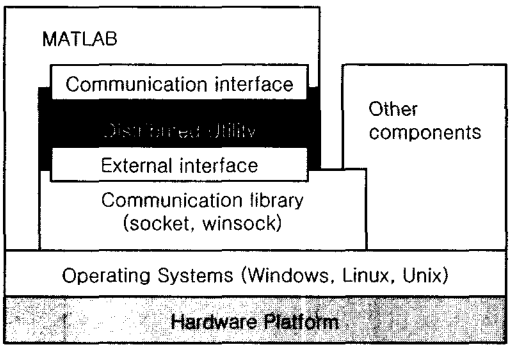


Figure 2 External interface to MATLAB

3.1 Parallel computing library

With the advent of PC including more than one CPU, multiprocessor architecture is not an uncommon programming environment even for the general users. By multiprocessor, we mean that two or more processors exchange data by accessing the common shared memory through the high-speed system bus only with a negligible overhead resulted from bus arbitration. In this computing model, the ability to partition a job and assign it to a processor is the indispensable requirement for parallel processing. Hence, currently released operating systems, running on the general-purpose PC, such as Windows 2000 and the like are able to assign the subpart of the given job to CPUs thread by thread. After all, if a programmer is to enhance the execution time based on the underlying multiprocessor framework, he (or she) should clearly define threads first and then program each thread routine.

Most MATLAB functions cannot be parallelized efficiently on the diverse multiple processor platforms, conflict with MATLAB's sophisticated memory model and architecture. However, matrix multiplication and inverse operation on MATLAB can benefit via threaded programming greatly in computation speed on the multiprocessor PC. The parallel multiplication can be simply implemented if we know the data structure of MATLAB's matrix representation, which is shown in Figure 3. Each matrix has a pointer variable and the real data part is pointed by it.

Before creating threads, we simply create two matrices, B1 and B2, and each of them contains only pointer variable but not the data themselves. B1 points to the first item of B, while B2 the midst. Notice that not A but B is partitioned because MATLAB stores each matrix item according to column-major order. This step involves no data copy. At last threads are created to execute C1 = A * B1, C2 = A * B2, respectively in parallel. After waiting the termination of each thread by synchronizing via *WaitForSingleObject* system call,

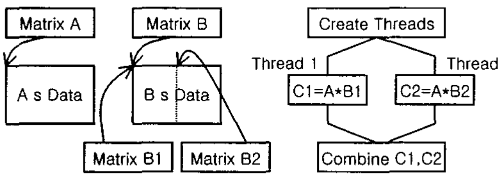


Figure 3 Threaded matrix multiplication

the main program combines the results into C matrix by copying C1 and C2 to the data area of C, which is reserved in advance for efficiency reason. As the final data copy invokes just a negligible overhead, the execution time of parallel multiplication is reduced almost by half. The mex routine can be easily extended to the case where the number of CPU is more than 3.

In contrast to multiplication, matrix inverse does not seem to have inherent parallelism in its execution. However, with the partition scheme as shown in Figure 4, the procedure is divided into several multiplications and one inverse for the quarter matrix[10]. As the data elements are stored sequentially in the memory, the vertical partition can be performed without any data copy. The partitioned inverse routine itself is known to outperform the original one. In addition, inverse operation is able to benefit from the enhanced performance of threaded multiplication scheme described at the previous paragraph.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad A^{-1} = \left[\begin{array}{c|c} G & -G\,A_{12}\,A_{22}^{-1} \\ \hline -A_{22}^{-1}\,A_{21}\,G & A_{22}^{-1} + A_{22}^{-1}\,A_{21}\,G\,A_{12}\,A_{22}^{-1} \end{array}\right]$$

$$G = (A_{11} - A_{12}\,A_{22}^{-1}\,A_{21})^{-1}$$

Figure 4 Partitioned matrix inversion

3.2 fast Jacobian program

The most efficient way to compute Jacobian is to use the so called an adjoined approach[6], where the computation is divided into several multiplications as shown in Figure 5. The code is MATLAB implementation of the core part published for general use. Though this is known as the most efficient method that has ever published, yet the Jacobian occupies the largest portion of execution time.

```
1  J=zeros(size(U,2)*size(U0,2), size(Agrad,2));
2  for i=1:size(Agrad,2);
3      A=reshape(Agrad(:,i), NNode, NNode);
4      JJ=U0.'*1/rho(i)^2 *A* U;
5      JJ=JJ(:);
6      J(:,i)=JJ;
7  end
```

Figure 5 MATLAB code for Jacobian

In our model that exploits 32 electrodes, we use Jacobian matrix of 992 * 776. The x dimension of *J* is the product of the number of electrodes and that of injected current patterns, while the y dimension is the number of elements in the target mesh. Hence, the x dimension is equal to the number of given equation or relation. *U0* and *U* are sets of expected and measured voltage, respectively. *Agrad* is the system matrix describing the equation system with sparse matrix whose dimension is 2825761*3104. First of all, C implementation can expect an efficiency in treating the scalar elements such as *size(Agrad,2), NNode, 1/rho(i)^2,* compared with MATLAB, as MATLAB handles them as 2 dimensional matrices for consistency.

The Jacobian consists of consecutive iterations and each iteration is further divided into reshape (line 3), 3 multiplications(line 4), and copy operation (line 5). Reshape function changes the dimension of a matrix without changing the values of element. As for the C implementation of line 3, we can avoid a number of explicit library calls by maintaining a location pointer that advances for each calling instance. This is due to the fact that each column has the equal number of elements and stores them with column major order in *Agrad*. The copy operation at line 5 incurs an unavoidable but insignificant overhead.

As for line 4, it is desirable that multiplication of a scalar quantity such as 1/rho(i)^2 should be applied to the smallest size matrix directly to the data area. *A* is such matrix, and let's denote *1/rho(i)^2 * A* as *A'* from now on. In Addition, the matrix operation for sparse matrix is much smaller than that of full matrix. *A'* is an extremely sparse matrix. Note that both *U0* and *U* are full matrices that have hundreds of items in the respective rows. The MATLAB execution, we found that the multiplication of *U0.'* and *A'* automatically generates a full matrix, magnifying the computation time of the next matrix multiplications. Hence, if we enforce the temporary intermediate matrix to a sparse one, we can expect more efficiency. In addition, (*A'* * *U*) generates sparser matrix than (*U0'* * *A'*). To handle the sparse matrix operation, the DLL code declares the data structure for sparse matrix according to the common way, that is, representing each nonzero element with the tuple, (row, column, value)[11]. The sufficient amount of memory should be allocated to store tuples explicitly at the beginning of mex routine via the MATLAB C library function *mxMalloc()*.

Based on this data structure, we have implemented the following matrix multiplications:

(a) MATLAB sparse with MATLAB full matrix into our own sparse matrix structure. ($A' * U$, say T)

(b) MATLAB full with our own sparse matrix into the data block ordered by column. ($U0.' * T$)

The operation carefully considers the MATLAB data structure shown in Figure 3. The result of (b) can be easily returned to the caller after wrapping the data block with the MATLAB matrix structure.

3.3 Implementation of communication primitives for MATLAB

For more than one MATLAB applications to cooperate to achieve a common goal, they should be able to exchange a matrix with one another, whether they are located at the same node or not. As the built-in functions of operating system can also be called from the mex routine, IPC (Inter-Process Communication) mechanisms such as named pipe, window socket can provide a useful functionalities to exchange MATLAB matrices[12]. For a reliable and efficient message delivery between nodes connected via the communication network such LAN(Local Area Network), we have selected TCP(Transmission Control Protocol) of Window socket library, as it performs transport layer functions including automatic error control. It should be mentioned that this mex routine is compiled with wsock32.lib static library offered by Visual C++.

Since TCP is a connection-oriented protocol, a connection should be established between the participants prior to the actual data exchange. For each connection, one plays a role of server, waiting and then accepting a connection request, while the other that of client. Hence, a series of system calls for the connection management operation is provided in addition to the calls for data exchange. The mex program should bridge between the MATLAB user and the operating system by translating the different data format and calling argument. To begin with, it distinguishes the function requested from the caller by the first argument as shown in Table 1. The complex data structure on socket library is hidden from the MATLAB command interpreter, offering an easy interface to the MATLAB user.

Table 1 Map of argument to window socket functions

1-st arg.	server	client
1	socket	socket
2	accept	send
3	send	recv
4	recv	close
5	close	

It is extremely important to handle the parameter handed over to the mex routine as a MATLAB matrix, which is entirely different from the data type of C program. Especially, when an application sends a matrix, the mex routine should correctly map the data area of the given matrix to the buffer parameter of *send* function. Oppositely, the receiver should be able to construct a MATLAB matrix from the received data block. To this end, the receiver should know both the exact size and dimension of the matrix, so such information should be given via supplementary arguments from the caller. In addition, the mex routine should convert the socket descriptor, returned from the *socket* or *accept* call as integer type of C language, into the MATLAB matrix before return to the interpreter. Figure 6 shows the sample MATLAB code where client sends a matrix to the server after establishing a connection. In this example, client sends a matrix of size 776 * 992 named as R to the server, and the server stores it as J after receiving the data from the network. This function constructs a MATLAB cluster. As shown in this example, the

```
p1 = mcs(1);               p1 = mcc(1);
p2 = mcs(2, p1);           mcc(2, p1);
J=mcs(4,p2,776,992);
  ------------             ------------
mcs(5,p2);                 mcc(4,p1);
mcs(5,p1);

     (a) Server                (b) Client
```

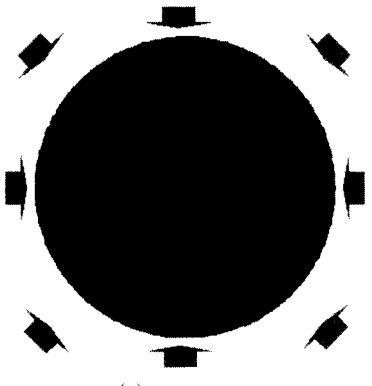
Figure 6 Example code for matrix exchange

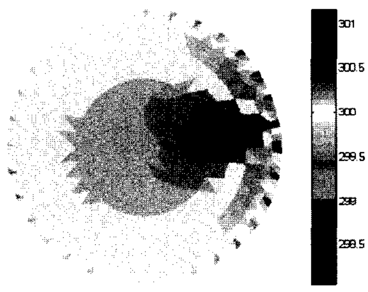
MATLAB user should do nothing but call *mcc* or *mcs* command on the command interpreter or in M-file script. That is, client calls *mcc*, actually invoking *mcc.dll*, while the server *mcs* invoking *mcs.dll*.

4. Distributed Grouping algorithm

To begin with, we define a frame as the set of measured voltages according to the injected current from one node. From the point of software algorithm, it doesn't matter whether the injection is performed at the electrodes sequentially or simultaneously. A frame needs 31*8 bytes. The is due to the fact that for one injection from an electrode, other 31 electrodes collect the data and store them as a floating point data type. Among the tomography algorithms, Kalman filter approaches, linear or extended, estimate internal image by sequentially inspecting frames one by one[13], while some other approaches such as Newton-Rapson take all frames into account simultaneously. Though the Kalman filter-based algorithm is faster than the others, its execution time is still very long, as it contains a number of time consuming operations such as inverse, multiplication of the 2 dimension matrix whose row and column are both proportional to the number of elements. For our general purpose Pentium III PC with 128 M memory and 600 MHz clock speed, it takes about 56 seconds to process one frame on average(excluding the first frame). The data dependency is so strong that it is very difficult to revise this program to a parallel or distributed one.

As natural, the efficient way to enhance the computation speed is to decrease the number of elements with or without the loss of correctness of the visualized image. When we can permit some loss of quality in the reconstructed image, an imprecise algorithm can be devised. Our strategy is to eliminate the element whose value is decided in the earlier iteration from the subsequent iterations. The elimination means that some elements are merged into one group and the group is treated as if one element. The grouping method has been proposed in several publications[14]. This strategy is based on the assumption that the target object


(a) true target



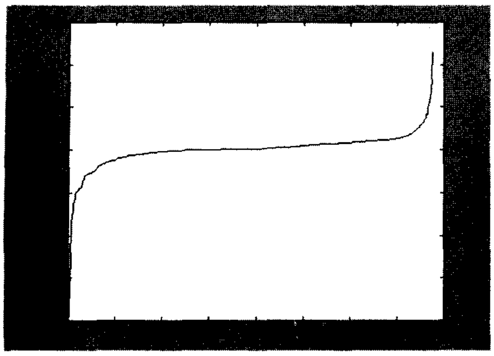
(b) the image after 1-st iteration

Figure 7 Real and calculated images

consists of, or filled with, one base material (known in advance) and some others. For example, fluidized bed system contains water and other substances.

Figure 7 shows an example where a material is located around the center of true target. After the first iteration, temporarily reconstructed image is also shown in Figure 7(b). The figure implies that there is some hindrance near the line extending the material and the electrode through which the current is injected. The elements belonging to this area need further iteration, but others may be considered as a base material from the early stage of iteration making the further computation unnecessary. That is, the elements that are below the certain bound are inferred as base material. Remaining elements may be other material, base material, or something else. Their resistivity values will be turned out with the progress of the iteration. But up to now, it cannot certain what they are.

Kalman filter begins its execution just with a


(a) after 1-st iteration

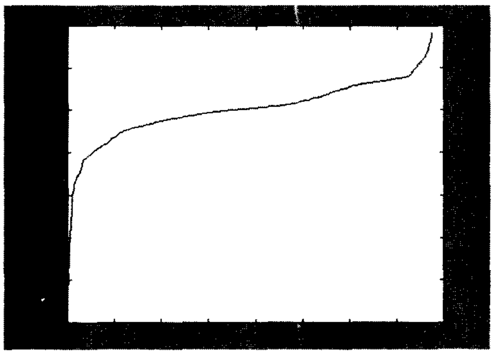

(b) after 3-rd iteration

Figure 8 Sorted value of each element

certain initialization, and each of iterations processes one frame. After the first iteration, it takes into account the estimated values produced up to the previous iteration. That is, the estimation after the k-th iteration is the accumulated result for the 1, 2, ⋯, and k-th iterations not just the result of k-th iteration. Hence, the first iteration can provide the most useful information on whether an element can be classified as the base material. In addition, it takes only 1/4 of other iteration to complete the first iteration as most matrices are initialized to sparse ones.

Figure 8 plots the resistivity distribution for each element, sorted by its value to demonstrate that some elements can be classified as base material. As shown in Figure 8, the image after the first iteration shows stepwise pattern that helps classify some element as base element, while the image after 3-rd iteration shows still more complex shape where the correct state of each element is not decided yet. As shown in Figure 8(a), there is a flat segment where graph shows almost horizontal line. We can find this pseudo line segment by inspecting the slope of consecutive points. We choose the average of start and end points of flat segment as the border value that classifies the elements. The determination of border point is another problem that is currently investigated.

The graph continues to be distorted for the time being within a certain number of iterations. This is due to the fact that the element fluctuates until it converges. The current injection from a different angle may draw a trace to the different direction. As natural, after the sufficient number of iterations, the image graph shows the correct internal structure of target object. As a result, if we discern the element whose value is below the bound after the first iteration and then mark as the base material, the processing time of next iterations can be reduced. Let such element be denoted as EI (Early Identified) element. The bound is a tuneable parameter. The lower the bound, the larger group can be obtained, resulting in more speed up but also the possibility of erroneous evaluation.

The first iteration can begin with any frame. So we make some nodes calculate the first iteration with different frames in parallel and then a master node combines the result. This enables us to maximize the number of EI element. Figure 9 shows the overall distributed computing model proposed in this paper, where 32 electrodes are attached. There are 4 computing nodes and each of them shares the information and data structure such as mesh, node, and element. After data acquisition, the master distributes 1, 8, 16, 24 frames respectively to slaves. The frame number is separated as far as possible to maximize the number of EI elements. Note that

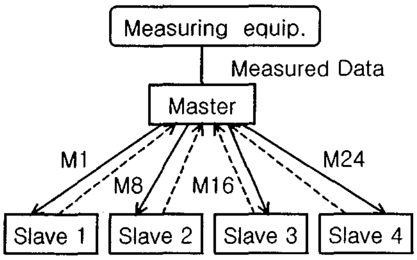


Figure 9 The distributed computing model

there are 4 electrode groups in Figure 7(a) named as **a, b, c**, and **d**. When there is just one computing node, it calculates with the data obtained from the current injection through the electrode group **a**. When there are node computing nodes, they groups the elements with the data set obtained from the injection through electrode groups **a** and **b**. When 4 node, they calculates with groups **a, b**, and **c**, while 8 nodes cooperates, with group **a, b, c**, and **d**. The number of computing nodes is identical to that of data sets.

Each node executes the first iteration with the allocated frame in parallel. After execution, the results are merged at the master. The traffic depends on the number of EI elements. The higher the traffic, the more reduction in the subsequent computation time is expected, though the increase of data transfer time is not significant.

5. Performance measurement and discussion

Table 2 shows the performance characteristics of master node. The specification of slave node is same except the number of CPU. Master node needs more computing power because serial iterations involve many time consuming operations on full size matrices, while the other node mainly manipulates sparse matrices in executing the first iteration of Kalman Filter. With this PC, it takes about 56 seconds to process one frame for original MATLAB code. In addition, the network interface of node as well as hub is 10 Mbps Ethernet.

The measurement of parallel multiplication shows about 48 % reduction of computation time, while that of matrix inversion shows about 64 % cut down by parallel and decomposed calculations. The calculation time of Jacobian has been cut down from about 20 seconds to 0.01 second. This remarkable enhancement results from the efficiency of C language in manipulating scalar type data as well as the efficiency of sparse matrix multiplication. Kalman Filter scheme, entirely programmed with MATLAB M-files, consists of consecutive iteration that is further divided into the following steps: FEM update, forward solution, spatial regularization, measurement update, and time update. The developed library shows identical reconstructed image with the original Kalman Filter M-file for the same input parameters. The maximum difference ratio is measured as $O(10^{-11})$. Finally, the computation time has been cut down from 56 to 29 seconds.

Table 2 Target machine description

CPU	Pentium III
# of CPU	2
OS	Windows NT
Memory	128MB

Return to the example target of Figure 7 to see the effect of cooperative grouping scheme. Figure 10 depicts how many elements can be merged into one group when 1, 2, 4, and 8 nodes cooperate, respectively. The darker area specifies the elements that can be classified as EI. As this example has a symmetric distribution, our cooperative scheme maximizes performance improvement by also symmetric frame allocation. Note that the size of element near the boundary is smaller than near the center. It is natural that the amount of grouped element depends on the internal structure of the target object, for example, how large portion is

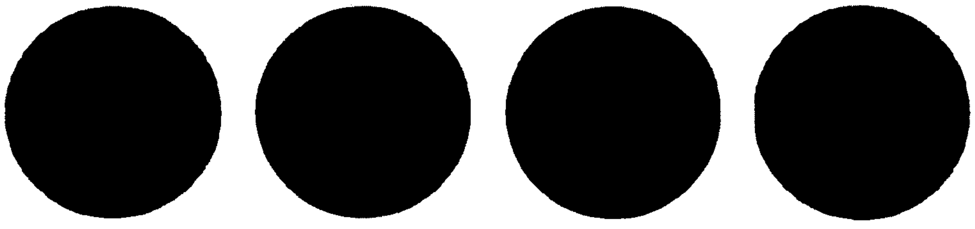

(a) 1 node (b) 2 nodes (c) 4 nodes (d) 8 nodes

Figure 10 Grouped elements

filled with base material, or where non-base-material is located, and so on. Besides, the performance is very sensitive to the number of elements, n, as most operation is $O(n^3)$. Table 3 shows the final number of elements after grouping for the respective experiments as well as the corresponding execution time. The execution time of no group is the result of just applying the parallel library and fast implementation of Jacobian.

Notice that there is a great speed improvement when the number of nodes changes from 1 to 2. This enhancement implies the reduction in the number of elements resulted from the large group size according to the following facts: First, more nodes can classify more elements having resistivity value of base material. Second, the two nodes processes with the data obtained by the injections from the opposite directions, so they can get independent EI's. Even if 4 or 8 nodes participate, the speed improvement is not so dramatic since many elements classified by respective nodes overlap with one another.

The cooperative method can dramatically reduce the frame processing time from 29.796 to 1.937 when there are 8 nodes, revealing the possibility of real-time image reconstruction via distributed cluster computing. The proposed algorithm includes data exchange overhead measured as small compared to the total procedure as well as the pre-grouping time amounted to 7.5 second. However, subsequent frame processing time, has been decreased significantly for the given example. As the number of loop is equal to that of electrode, whole speed up is very large. As natural, we can also achieve a reasonable reconstruction time with the upgrade on the platform components, for example, higher performance CPU, more memory, faster network interface, special hardware support like DSP, and efficient numerical library. That may enable even the real-time image reconstruction with our algorithm.

We measure the performance of proposed grouping scheme for the various targets as shown in Figure 11. Figure 11(a) shows the case where there is only one bubble near the electrode 1 (belonging to group a). Figure 11(b) shows that there are 2 bubbles near electrode 1 and 16, while Figure 11(c) shows 3 bubble case.

The number of groups after combining the EI elements and corresponding reconstruction time are shown in Table 4 for each target and for each number of computing nodes. For the target of Figure 11(a), there is little reduction(just 6) when there is just one node, as the bubble locates near the electrode. In contrast, another node manipulates with the injection from the electrode at the opposite side, the reduction is dramatic, reduction up to 464, as the current can travel far away to the bubble area. After that, the reduction is not so great even then number of computing node increases. For the target in Figure (b) and (c) where the internal state is somewhat complex, we cannot expect an

Table 3 Frame processing time

	No group	1 node	2 nodes	4 nodes	8 nodes
# of elmt.	776	507	267	235	196
time	29.796	15.344	3.375	2.484	1.937

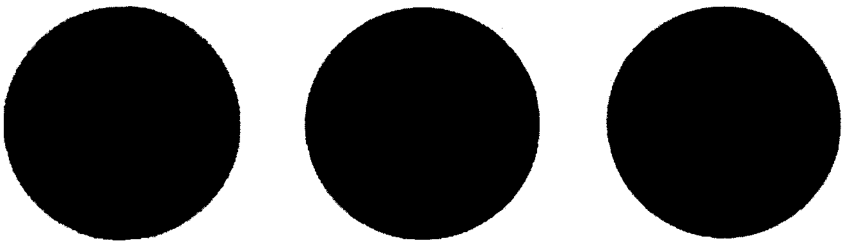

(a) left bubble (b) two bubbles (c) three bubbles

Figure 11 Various targets

Table 4 The number of group and corresponding execution times

	1 node	2 nodes	4 nodes	8 nodes
(a)	770 (29.125)	464 (8.115)	370 (4.906)	317 (3.429)
(b)	768 (29.122)	752 (28.301)	614 (16.301)	566 (13.086)
(c)	694 (22.697)	684 (21.950)	647 (18.732)	622 (16.879)

excellent grouping effect. However, as the matrix operations are typically O(n^3), even the small reduction in the number of elements can improve the speed of reconstruction time.

6. Conclusion

In this paper, we have described and proposed performance enhancement techniques for EIT, aiming at high-speed image reconstruction. The work includes an efficient parallel library for numerical computing tools and a distributed EIT algorithm where each node cooperates to reduce the number of elements. The extended libraries have not only halved the reconstruction time but also enabled distributed computing. The proposed cooperative computing algorithm has reduced the frame processing time dramatically from 29.796 to 1.937 when there are 8 nodes, revealing the possibility of real-time image reconstruction. Though the grouping effect depends on the internal structure of the given object, we can expect more performance improvement via larger group with more computing nodes. In addition, the border value, which decides whether an element can be grouped or not, is a tuneable parameter. As a future work, we will build a computing cluster with MPI(Message Passing Interface)[15] and a numerical library on top of LINUX operating system, and then develop a distributed algorithm that can enhance the reconstruction speed.

References

[1] M. Cheney, et. al., "Electrical impedance tomography," *SIAM Review*, No, 41, pp.85~101, 1999.

[2] M. Kim, S. Kim, K. Kim, J. Lee, Y. Lee, Reconstruction of particle concentration distribution in annular Coutte flow using electrical impedance tomography, *Journal of Industrial Engineering Chemistry*, Oct. 2001.

[3] R. Buyya, *High Performance Cluster Computing*, Vol.2, PTR PH, 1999.

[4] W. Feng, J. Liu, "An extended imprecise computation model for time-constrained speech processing and generation," *IEEE Workshop on Real-Time Applications*, pp.76~80, May 1993.

[5] M. Vauhkonen, et. al., "A MATLAB Package for the EIDORS project to reconstruct two-dimensional EIT images", *Physiology Measurement*, vol. 22, pp.107~111, 2001.

[6] R. Johnson, S. Jayaram, L. Sun, J. Zalewski "Distributed Processing Kalman Filter for Automated Vehicle Parameter Estimation: A Case Study," *IASTED Int'l Conference on Control and Applications*, Banff, Alberta, Canada, July 24~26, 2000.

[7] Woo EJ, Hua P, Webster JG, Tomkins WJ, Pallas-Aren, "A robust image reconstruction algorithm and its parallel implimentation in electrical impedance tomography," *IEEE Trans. on Medical Imaging*, 12(2), pp.137~146, 1993.

[8] http://www.mathworks.com

[9] S. Pawletta, and et. al., "A MATLAB toolbox for distributed and parallel processing," *Proc. MATLAB Conference*, 1995.

[10] K. Gallivan, et. al., *Parallel Algorithms for Matrix Computations*, SIAM, 1990.

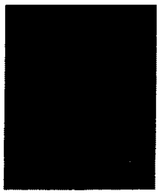
[11] E. Horowitz, S. Sahni, S. Anderson, *Fundamentals of Data Structures in C*, Computer Science Press, 1995.

[12] B. Quinn, D. Schute, *Windows Sockets Network Programming*, Addison-Wesley, 1995.

[13] M. Vauhkonen, et. al, "A Kalman filter approach to track fast impedance changes in electrical impedance tomography," *IEEE Trans. on Biomedical Engineering*, pp.486~493, 1998.

[14] K. H. Cho, S. Kim, Y. J. Lee., "Impedance imaging of two-phase flow field with mesh grouping algorithm," *Nuclear Eng. & Design*, pp.18~26, 2001.

[15] P. Pacheco, *Parallel Programming with MPI*, Morgan-Kaufmann Publishers, 1997.



이 정 훈

1988년 서울대학교 컴퓨터공학과 공학사 1990년 서울대학교 컴퓨터공학과 석사 1996년 서울대학교 컴퓨터공학과 박사 1996년~1996년 대우통신 광통신연구실 선임연구원. 1997년~현재 제주대학교 전산통계학과 교수. 관심분야는 실시간 통신, 분산 시스템, 멀티미디어 통신

박 경 린

1986년 중앙대학교 전자계산학과 공학사 1992년 텍사스주립대 컴퓨터공학과 석사 1997년 텍사스주립대 컴퓨터공학과 박사 1998년~현재 제주대학교 전산통계학과 교수. 관심분야는 분산/병렬 처리 시스템, 오류 허용 시스템, 컴퓨터 알고리즘, 성능 평가 등