データベース・システム　76－1
（1990.　3.　12）

モデルに基づく新しい並列データ・ベース・マシンの設計

アィサム.A.ハミド.　白鳥 則郎　野口 正一

東北大学電気通信研究所,

〒980 仙台市片平2-1-1

本論文は、メモリ共有型のマルチプロセッサ・データベースマシンにおけるジョイン操作を実現するための並列アルゴリズムの集合について述べる。これらのアルゴリズムの開発は、構成的アプローチに従っている。まずジョイン操作の処理にかかわる主な手続を従来の結果を整理し要約する。次にこれらの手続を実行する従来の手続の手法を効果的に統合することによりこれらを包含する一般的なアルゴリズムを構成する。更にこれらのアルゴリズムの性能について検討する。その結果、ハードウエア構成が与えられたとき、すべてに渡って最良の性能を示すジョインアルゴリズムはなく、ジョイン操作に関係したデータの性質に依存して最良の性能を与えるアルゴリズムが存在することを示す。

A New Realization for Parallel DataBase Machines With its Model

Issam A. Hamid　Norio Shiratori　Shoichi Noguchi

Research Institute of Electrical Communication, Tohoku University

2-1-1, Katahira, Aoba-ku, Sendai shi, 980, Japan.

In this paper we have given a design for shared memory multiprocessor parallel database machine using a very fast type of interconnection network. Also, a large set of parallel algorithms for implementing the join operation on such DB have been given. The approach is a structured one. The major steps of the join operation done by the machine is defined. Other alternative join algorithms are constructed by concatenating the different ways of performing those join operation. Performance model has been given to show that for a given hardware configuration there are different algorithms effect the best performance, which is depends on the characteristics of the data and its structure in the join algorithms.

1-Introduction

The general objective is, the design of a back-end Data base Machine (DBM) suitable for supporting multiple user, on line large relational data base systems.

The limitations of conventional database systems, the continuous advancements in memory-processor technology, and the continuous reduction in fabrication costs, inspired a new approach to database system implementation. This approach replaces the general-purpose von Numann processor with a dedicated machine, the data base (DBM), tailored for the data processing environment and in most cases utilizing parallel processing to support some or all the functions of the DBM's.

There is a novel scheme for classifying the set of DBM's proposed so far[Qa85]. This scheme was used to qualitatively evaluate and compare the respective of DBM's.

The most important characteristics of contemporary and anticipated very large scale relational database systems are, the vast amount of data in such systems, and the large number of users requiring simultaneous access to such data.

We have formulated the following *guidelines*, and motivations along which a DBM of a large capacity store with a big ability to handle the on-line multiple-user access with adequate response time can be designed .

1) The mass store is to consist of moving-head disks, because of its ability to provide a vast amount of on-line storage at a relatively low cost and moderate performance.

2) The indexing is of the kind of page-level indexing, which can greatly improves the execution time of the selection and the modification operations. But this may introduce overhead in executing these operations in the form of index access delay and maintenance, and increases the execution time of the other update operations.

We suggested to reduce this overhead, so that, the page must be selected to have a large size and be processed in parallel processor.

3) The DBM is to organize as MIMD type which can provide parallel access to the DB. Because of relatively low cost of the processor and memory devices, the use of parallel processing may enhance the effectiveness of a DBM. Although the SIMD organization of a DBM may reduce the execution time of a DM operation, it does not offer a real solution to the multiple-user problem. The MIMD has the ability not only to execute a DB operation in parallel, but also to execute more than one operation (for instance, from the same or different queries) in parallel. In this paper we introduce a new architecture for large database machine which has a strong capability of parallelism using a dedicated interconnection network with added mechanism of Fetch-and-add instruction, so that to over come the critical sections used in most conflict access control algorithms, and present rich set of parallel joint algorithm with high performance model.

4) The DBM is an off-disk type organization, which can reduce the initial large amount of data movement by taking advantage of the page level indexing and the local and sequential references in the database.

5) Over come some important problems that can simplify database software and increase its consistency operations; like,

a) concurrency control algorithms can be improved through the new added mechanism of Fetch-and-add instruction.

b) cross referencing is removed without a sort operation using hashing bit and pointer arrays.

2- The System Outline

The processor elements of MIMD should be designed to efficiently support not only the relational algebra operations such a selection and projection, but also the primitives that manipulate the page index as well.

1) **The main controller (MC);**

The MC interfaces the users to the database system, translate the user queries to the primitives of the DBM, controls the other component of the DBM. The implementation of the MC is strongly dependent on the way of partitioning of DBM's task, come from the host machine.

Based on this partition, the MC can be implemented using a minicomputer.

2) **The mass storage (MS);**

This subsystem consists of *two* level memory, and controlled by MSC (mass storage controller);

a) Mass Memory (MM)

Mass memory consist of a set of moving-head disks. The disk is modified to read and write from and to more than one track in parallel.

Tracks which can be read (written) in parallel from (to) one disk, from what is called a minimum access unit(MACU)[Qa85].

The tuples within this unit, are laid out on the tracks of the corresponding moving-head disk in a bit serial-word serial pattern.

Also, the MM stores another type of data, named index terms. These terms are clustered according to their attribute name, in order to improve their retrieval and storage costs. Hence, the index terms of same attribute resides in the same index page.

b) Parallel Memory Modules.(PMM)

This subsystem is organized as a set of large modules (blocks), each with size equal to an MACU. It is also, partitioned into a set of submodules, each store one track of a moving-head disk.

3) **The MIMD subsystem consists of MSIMD system.**

a) Every one SIMD has one CU. Several SIMD subsystems controlled(coordinated) by the main controller presented the total MIMD subsystem.

b) Each SIMD subsystem consists of a set of triplets of the form, <I/O controller processor, local memory>.

All PEs is controlled by the SIMD-CU, through the Broadcast Bus-(BB), which permits SIMD-CU to write the same data to all PEs, and read data, from every one of its PE's memory(Fig. 2).

4) Interconnection Network (IN) and parallel update

The IN has the ability to allow any number of SIMD's or moving-head disks to read(write) from (to) different memory modules simultaneously and enable two or more SIMD's to read from the same memory module in parallel. The input to IN is the connected to the input of the individual PEs. The output of IN have a number of buffers represent the memory needed for Fetch-and-Add execution. (Fig. 3)

The latter requirement is needed to provide the system with the capability of handling simultaneous processing of the database operations.

The IN has used Fetch-And-Add mechanism[St84] to serialize locking and reservation operation in database applications. In this sense we designed our IN to be a kind of Benes rearreangeble MIN because of its ability to realize arbitrary permutation function. We had succeeded to present very fast routing algorithm to realize certain very important permutation functions using such O(log N) time steps[Ha89]. Such bound excited us to use it for parallel updates in data base machine. The extra realization introduced here is to add a small buffer in the input and output of IN, to implement the fetch-And-add mechanism(Fig. 3).

Fetch-And-Add can serialize N simultaneous requesters in a single operation, while other mechanism like test-and-set, and Compare-and-swap free the N requesters to pass through the lock in a serial fashion.

The major reason behind the use of the Fetch-And-Add is to eliminate serialaccess to shared data, when two or more processors place requests to update the same memory cell, the requests are honored sequentially. To the extent, that processors are forced to execute Read/Modify/write serially.

Processing cluster is a set of triplets; processors(PE), local RAM, DMA controller. Local RAM has the capacity of several disk tracks. Number of PEs within one cluster is the number of sub-blocks with one block of the parallel buffer. The PEs within a cluster are controlled and managed by the cluster master PE.

The data with a cluster are moved between PEs via tuple bus, which is provided with both point-to-point and broadcast capabilities.

The parallel buffer is organized as a set of blocks, each block having the size of one MACU. A block is further partitioned into a set of sub-blocks of a mass memory disk.

3- Back-ground and Comparisons of other join algorithms

On the various multiprocessor DBM several parallel algorithms have been proposed to implement the join operation.

Most of these algorithms are parallel version of the nested-loops algorithm [De79] to perform join operation of the DBM. The algorithm assumes that both relations for instance, S and T are stored in page units on the system disk. The parallel nested-loops algorithm begins by having each of the PEs read a different page from relation S. Then, all the pages of relation T are broadcast, one by one to all PEs. Upon receiving a page of T each PE joins its page of S with the incoming page of T, using the nested-loops algorithm[De79]. These steps are repeated until all the S pages are joined.

The parallel nested loops algorithm has been extended[Va84] to allow a number of pages from relations to be read into a PE before the broadcast of pages of relation T would begin. Such an extension improves the speed of the parallel nested-loops algorithm by reducing the numbers of times the relation T has to be broadcast to the PEs during the execution of such algorithm.

If the number of pages storing the source relation is the same as that of the target relation, and equal to N, then it is easy to show that the parallel nested-loops a time complexity of $O(N^2)$[Va84].

Another proposal has used the uniprocessor two-way sort-merge algorithm [Bi83] as a basis for a parallel join algorithm. The parallel sort-merge-join[Bi83] algorithm has the time complexity O(N log N).

Another proposal for a parallel join algorithm has been given for the GRACE multiprocessor DBM[Ki84]. This algorithm joins S and T relations by first distributing their tuples between a number of buckets based on the tuples' join attribute values using a hashing function. The joining of tuples within a buckets is performed by PEs that use a sort-based algorithm. The PEs in GRACE[Ki84] are specially designed as hardware sorters with capability of sorting N tuples in linear time. The Grace' algorithm[Ki84] has the time complexity O(N), which is linear with respect to the number of pages that store the source or target relation.

Another Proposal for parallel join algorithm has been given based on Hypertree multiprocessor DBM[Go81]. This algorithm begins with each PE applying the same hashing function to the join attribute of its own tuples. The hash function uses the join attribute value of a tuple to compute a PE number. The tuple is then sent to the PE having such a number. During the second phase of the algorithm each PE join the tuples it receives as a result of the execution of the first phase.

Adopting a GRACE-like[Ki84] or [Good81] like algorithm alone does not guarantee that the join operation on a given machine will have the best performance. To remedy such a demerit we have given here three different algorithms. The first, one is linear and behaves similarly to the GRACE[Ki84] algorithm for joins when the probability of joining is high.

The second behaves as [Good81] but it is more general and good for joins of low probability.

The third algorithm is for joins that have moderate joining probability.

The use of simple optimizer to choose one of these algorithms for a given join operation is simple and guarantees that such an operation will in fact be carried out with high performance.

4- Parallel Join Algorithms

One of the most important operations of the relational systems and probably the most important limiting factor on their performance is the Θ-join of a source relation S and a target relation T on attributes A from S and B from T is the relation R, obtained by concatenating each tuple s∈S and each tuple t∈T whenever (s[A] Θt[B]) is true. Θ is one of the operators $(=,\neq,<,>,\leqq,\geqq)$, the Θ-join operation is generally needed for formulating queries which reference more than one relation. The most frequently used type of Θ-join operation is the equi-join, (=).

Database is divided into a set of large data units. Each called a page represents the smallest addressable unit of data. The tuples in the same page should have the same relation. Database directory contains the information needed to map a "data-Name" to the set of page addresses which store the named data. Data named at two levels; Relation level, tuple level. Tuple name: <relation name, attribute name, value>. Database directory consists of two indexes; Relation indexes, Page index. Relation index; Maps the"Relation name" to a set of page addresses. These pages contain all the tuples of the relation whose name is "relation name". Relation index contains entries for all the relations in the DB.

The best performing join algorithm is not unique but algorithms score the best performance depending on the characteristics of the input source and target relations.

In the following we give different algorithms categories depending on the relations tuples.

1) Global Broadcast:

Different MACU of the source relation and all of the MACU's of the target relation are assigned to every phase, (Number of phases = Number of MACU storing the source relation). Therefore, one cluster having a typical phase needs the read from the parallel buffer and process a source MACU and all of the target MACU's.

2) Global Hash:

Hashing function partitions the domain which represents the join attributes of both the source and target relations into dijoint subsets. Hence, the expected number of tuples from the source relation subset is the local memory units of the processors within a cluster. The set of source and target tuples which hash to one global bucket constitutes the tuples which are assigned to one phase for joining (i.e., number of phases = number of global buckets).

Selecting the tuples of phase i of a global hash algorithm requires a cluster to read from the parallel buffer into its PEs all of the source and target MACU's one MACU per time.

As one MACU is read into the PE's parallel memories, the PEs themselves in parallel compute a hashing function over the join atribute value of every tuple of the MACU and collect these tuples that hash to phase i (global bucket i).

The tuples read (collected) by a particular cluster during the execution of one phase of a basic algorithm can be distributed among the clusters' PEs for joining using one of two methods;

(a) *local Broadcast*;

The source tuples read (collected) by each PE during the initial execution of a phase are left in the PEs, (the target tuples of that phase temporarily buffered in the PEs, and the parallel buffer and the mass memory in the case of global broadcast algorithms) are broadcast, one tuple at a time over the tuple bus of the cluster. Then the PEs in parallel join the broadcast tuples within their share of the source tuples.

(b) *Local Hash*;

Source and target tuples of a phase are hashed based on their join attribute values to the PEs of the cluster that executes the phase. The hashing function statistically independent of the global one in case of global hash algorithms is computed by the PEs for every tuple of that phase.

Using the tuple bus the hashed tuples are moved to their destination PEs for joining.

There are three methods to a PE for joining a target tuple with the PEs' assignment from the source tuples of one phase.

(1) *The complete Comparison*:

The join attribute of the target tuple is compared to that of every source tuple in the PE. The target tuple is concatenated with a copy of every source tuple having the same join attribute value.

(2) *Sorting:*

The source tuples assigned to a PE are first sorted based on their join attribute value. Such joining is performed by the binary search method[] to locate those source tuples whose join attribute values are the same as that of the target one,

In a PE the sorting of the source tuples is implemented by actually sorting a table of pointers through which the source tuples are referenced.

(3) *The hash table*:

The hash tuples in a PE are first; hashed (based on their join attribute values) to a hash tuple using a suitable hashing function.

The join is performed then by hashing the target tuple (based on its join attribute value) using the same hashing function to the hash table.

A scan of the join attribute values of all the source tuples within a bucket of the hash table is needed to locate those source tuples whose join attribute values are the same as that of the target tuple.

An algorithm name can be thought as the ordered quadruple <X,Y,Z, basic>. X,Y,Z, are the results of the the three decisions taken by the normal algorithm and the word basic indicates that the algorithm belongs to the basic algorithms. Therefore, X$\in${*Global Broadcast, Global Hash*}, Y$\in${*Local Broadcast, Local Hash*}, Z$\in${*Complete, Sort, Hash*}. The word basic is related to the type of join operations on the source or target relations filtering. Therefore, we have about 12 types of join algorithms and we need to study there different performance on the data base machine.

From the point of view of a cluster the execution of one phase of a basic algorithm involves six subphases. During the 1st subphase, a set of source tuples (the tuples of a source MACU for the global broadcast algorithms or the tuples of the source relation which hash to a global bucket for the global hash algorithms) are read/collected by the PEs of the cluster.

In the 2nd subphase the selected tuples from the 1st subphase in PEs are hashed to the cluster's PEs. In the 3rd phase all PEs within the cluster sort or store in a hash table, their shares of the source tuples, or do nothing with them.

The PEs within a cluster execute the 1st 2nd and 3rd subphases in a pipeline fashion with the source tuple as the unit of the pipeline. So that when the tuple is read/collected by a PE in the 1st subphase the execution of the 2nd subphase is triggered which in turn triggers the execution of the 3rd one. when all tuples of the source relation read/ collected in the first subphase are processed through the second and 3rd ones, then the execution of the 4th subphase is triggered.

During the 4th subphase a set of target tuples (all of the tuples within the target relation for the global broadcast algorithm or the target tuples which hash to a global bucket for the global hash order) is read /collected by the clusters; PEs.

In the 5th subphase the target tuples of the fourth subphase are broadcast/hashed to the clusters' PE. In the 6th subphase every PE within the cluster joins its share from the source and target tuples.

The clusters PE's execute the 4th, 5th, and 6th subtuples in a pipeline fashion with the target tuples as the unit of the pipeline.

Hence we have three major algorithms for comparison. The **First** is global hash, local broadcast sorting, of basic joint functions(**Algo-1**) which is similar to GRACE[Ki84] algorithm of high join probability.. The **second** is *global broadcast local broadcast sorting*, of target relation joint functions(**Algo-2**) which behaves as [Go81] for low join probability. The **third** is global broadcast local broadcast sorting, of source tuples joint functions(**Algo-3**) for moderate join probability. In the next section we make a model for the comparison of these three algorithms.

5-Algorithms performance Model

How to study the performance of the proposed parallel algorithms in carrying out the join operation? One measure is total execution time of the algorithm(τ). τ is the total time required by the multiprocessor to execute the algorithm (assuming that there is no overlapping in processing). There are two exceptions to such overlapping.

(1) The activities of the different PEs within a cluster overlap each other.

(2) The activity of broadcasting tuples over the tuple bus of a processing cluster completely overlaps the activities performed by the cluster master PE, (updating and checking the Boolean arrays).

We have used analytic performance measure by probabilistic average value models, which are referred to as the join execution models.

The set of parameters which characterize the model can be as either input or output. The output category include τ performance measure. The input category has three groups of parameters, data, hardware algorithm and hardware.

The *data parameters* are;

$\beta(y)$ represents the cardinality of the relation; y, y$\in${S,T}, and has a value of 10^3-10^5 tuples.

$\gamma(y)$ tuple length of relation; y of value of 100 bytes.

γ_a joint attribute length of 10 bytes.

γ_c cardinality of the domain underlying the join attribute.

The *hardware algorithm parameters* are:

μ_h is the number of buckets in a hash table of a PE.

μ_B is the number of bits in a Boolean array.

The *hardware* parameters;

N_{PE}; is the number of PEs within one cluster, (or the number of tracks within one cylinder of the moving head disk) of value of 15.

Σ; is the capacity of an MACU of value of 0.71×10^6 bytes.

τ_c; is the time to compare the join attribute value of a source tuple to that of a target one of value of 0.02 ms.

τ_h; is the time to calculate a hashing function with the tuple's join attribute value as the input argument of value of 0.102 ms.

τ_s; is the time to swap two pointers within a PE's memory of value of 0.007 ms.

$\tau(s)$; is the time to move a tuple of relation y across the tuple bus of a cluster of value of 0.1 ms.

$\tau_{ave.}$; is the average access time of the system disk of value of 16 ms.

τ_D; is the time to move the disk head one track = 3ms.

τ_{td}; is the time to transfer an MACU between the system disk and the parallel buffer block = 16.7 ms.

τ_{pb}; is the time to transfer an MACU between the parallel buffer block and the PEs of a cluster = τ_{td}.

The join execution models has the following assumption:

(1) The multiprocessor hardware consists of only one processing cluster one parallel buffer block and one disk,

(2) The MACU is a cylinder of the moving head disk. The disk is modified for the parallel read out(write in) from (to) all of the tracks within one cylinder, simultaneously.

(3) A tuple of the source or target relation is equally likely to carry in its join attribute any value from the attribute underlying domain.

(4) All of the hashing functions used by join algorithm are ideal and statistically independent of one another.

A hashing function is ideal if it is equally likely to map a value from its domain to any value of its range.

Notation:

τ is the total time required by the multiprocessor to execute the join algorithm

τ_1 is the total output time to move the MACU's of the output relation from the cluster's PEs to the system disk during the execution of a join operation.

τ_2 Time spent by a PE in hashing tuples during phase x.

τ_3 is the total input time to move the tuples of the source and target relations from the system disk to the clusters' PEs during the execution of a join operation.

$\tau_3(x)$ is the input time of phase x.

K is the number of phases in the execution of the join algorithm

x is the phase Index , x = {1,2,....K}.

y is the Relation index where y∈{S (source relation, T(Target relation)}.

$\tau_4(x)$ is the time spent by a PE in comparing the tuples' join attribute value during phase x.

τ_5 is the total Processor Time = Time spent by a PE to execute a join algorithm.

$\tau_5(x)$ is the time for a processor to execute phase x.

$\tau_6(x)$ is the bus transmission time of phase x.

τ_6 is the total bus transmission time = Time to move tuples over the tuples bus of a cluster during the execution of a join operation.

$\tau_7(x)$ is the time spent by a PE in sorting source tuples during phase x.

ε_1 is the expected number of MACU's (cylinders) which store the output relation.

$\varepsilon_2(y)$ is the number of MACUs' (cylinder which store the tuples of the relation; y.

ε_3 is the fraction of the target relation tuples which survive the checking of the Boolean array BA-S during the execution of one phase of a global broadcast join algorithm.

ε_4 Number of distinguished values in the join attribute of the tuples of the relation y.

ε_5 Number of global buckets in a global hash join algorithm.

ε_6 Number of tuples in a source MACU.

ε_7 Fraction of the tuples from the relation T which hash to a global bucket and survive the checking of the Boolean array BA-S during the execution of one phase of a global hash Join algorithm.

ε_8 is the expected number of distinguished values in the join attribute of the tuples in a source MACU.

$\varepsilon_9(y)$ is the expected number of distinguished values in the join attribute of the tuples from relation (y) that belong to one global bucket.

R_1 is the ratio of the number of bits in the Boolean array(s) and the number of tuples in both the source and target relations.

R_2 is the ratio of the number of buckets in a processor hash tuple and the PE's expected number of source tuples per one phase of a JOIN algorithm.

Formulas for the join execution models:

The proofs of these two theorems have been refrained from here because of space.

Theorem-1

Given a collection of keys (possibly not distinct) with cardinality γ_k defined on a domain with cardinality γ_c, if a key is equally likely to have any value of the domain, then the expected number of distinguished values γ_{ck} in the collection of the keys is :

$\gamma_{ck} = \gamma_c(1 - Exp(-\gamma_k/\gamma_c))$ for $\gamma_k, \gamma_c \gg 1$. ■

Theorem-2

Given a collection of keys (distinct) with cardinality γ_k , if a key hashes to a Boolean array of size μ_B; with equal probability, then the expected number of bits set(α_{BS}) in the array is:

$\alpha_{BS} = \mu_B(1 - Exp(-\gamma_k/\mu_B))$ for $\gamma_k, \mu_B \gg 1$. ■

The τ spent in executing one of the target tuples partial filtering join algorithms can be execute as follows:

$\tau = \tau_1 + \sum [\tau_3(x) + \tau_5(x) + \tau_6(x)]$; because $\tau_3(x)$, $\tau_5(x)$, and $\tau_6(x)$ for all x∈{1,2,...,K} are on the average equal to $\tau_3(1)$, $\tau_5(1)$, and $\tau_6(1)$, respectively. Hence, $\tau = \tau_1 + K \cdot [\tau_3(1) + \tau_5(1) + \tau_6(1)]$.

The total output time τ_1 for each of the target tuples relation partial filtering join algorithms can

be computed by the following formula using the hardware input parameters:
$\tau_1 = \tau_8 (\tau_{pb} + 2\,\tau_{ave.} + \tau_{td})$.

The above formula states that for every MACU of the resulting output relation, it takes τ_{pb} to move the MACU to the parallel buffer block.

$\tau_{ave.}$ is to locate randomly an empty cylinder on the disk and τ_{td} to move the MACU to the empty cylinder of the disk.

In addition it takes another $\tau_{ave.}$ to move the disk arm back to resume moving the target relation to the cluster.

The formula that relates ε_1 to the models' input parameters is derived as follows; If ε_6 is the expected number of source tuples with the same value. Then the expected number of tuples in the output relation; $\varepsilon_{o/p}$ is (ε_t . $\varepsilon_6(v)$ .p) which is a tuple of the largest relation (ε_t) carries in its join attribute a value which exists in join attribute of the source tuples.
$\varepsilon_6(v)$ can be represented as the ratio; $\varepsilon_6/\varepsilon_8$, hence;
$\varepsilon_{o/p} = \varepsilon_t . (\varepsilon_6/\varepsilon_8) . (\varepsilon_8/\varepsilon_4)$. $\varepsilon_1 = (\varepsilon_t . \varepsilon_6)/(\varepsilon_4 . \Sigma)$

For a global broadcast, target tuples relation partial filtering algorithm K has the value of ε_{ms}, and ε_{mt}, which is the number of MACU's (cylinders) that store the tuples of the source and target relations, respectively. On the other hand for a global hash target relation algorithm, K has the value of ε_5 the number of global buckets associated with the algorithm.
ε_5 is chosen such that the set of source tuples which hash to one global buckets fits in the local memories of the PEs' within the cluster.

The number of source tuples per global bucket due to an ideal hash function that uniformly distributes a set of tuples with cardinality ε_6 over a set of ε_5 buckets is a normally distributed random variable with an expected value of ($\varepsilon_6/\varepsilon_5$)
Therefore;
$(\varepsilon_6/\varepsilon_5)\,\mathrm{LTS} = \Sigma$,
$K = \varepsilon_5 = \lceil \varepsilon_6 .\ \mathrm{LTS} / \Sigma \rceil$.

Two expressions exist for $\tau_3(1)$, one for the X = *global broadcast* and the other for X = *global hash -- target partial* algorithms.

In deriving these two expressions, it is assumed that the tuples of the source and target relation are stored on a adjacent cylinders of the disk.

During one phase of a *global broadcast* target partial filtering algorithm one source MACU and all the MACU's of the target relation are read by the processing cluster, therefore,
$\tau_3(1) = (\tau_{ave.} + \tau_{td} + \tau_{pb}) + \tau_{ave.} + \tau_{td} + \tau_{pb} + (\varepsilon_{mt} - 1)(\tau_D + \tau_{td} + \tau_{pb})$

The above formula states that moving the first target source (MACU) from the disk to the PEs needs on the average, TDAC to locate the cylinder on the disk that stores the MACU, τ_{td} to transfer the cylinder content to the parallel buffer block, and τ_{pb} to transfer the MACU to the PEs.

Also, for every remaining target MACU it takes τ_d to move the disk heads to the next cylinder, τ_{td} to transfer the cylinder content to the parallel buffer block and τ_{pb} to transfer the content of the parallel buffer block to the PEs.

ε_{mt}, which is the number of cylinders that stores the tuples of the target relations, and can be computed from the following formula;
$\varepsilon_{mt} = \lceil \phi_{tt} . \gamma_t / \Sigma \rceil$.
Where ϕ_{tt} is the cardinality of target relation, and NTS is the cardinality source relation. It is assumed that $\phi_{tt} = \phi_{ts}$. Also, γ_t and γ_s is the tuple length of the target and source relation respectively.
P_t and Ps; are the parameters of tuple bus of a cluster which are calculated assuming the effective bandwidth of 1 Mbytes.

During the execution of one phase of a *global hash* target partial filtering algorithm all of the MACU's which store the tuples of the source and target relations are read by the processing cluster.

This is needed to collect the source and target tuples which hash to one global bucket , Hence,
$\tau_3(1) = (\tau_{ave.} + \tau_{td} + \tau_{pb}) + (\varepsilon_{ms} - 1) \cdot (\tau_d + \tau_{td} + \tau_{pb}) + (\tau_{ave.} + \tau_{td} + \tau_{pb}) + (\tau_{td} - 1) \cdot (\tau_d + \tau_{td} + \tau_{pb})$.

The derivation of formulas for the quantities $\tau_6(1)$ and $\tau_5(1)$ of the target partial filtering algorithms are presented next:

During the execution of one phase of a *global broadcast* target partial filtering algorithm the tuples of a source MACU are joined with the tuples of the target relation. In doing so a *global broadcast, local broadcast* target partial filtering type's algorithm broadcasts to the PEs only those target tuples which survive the Boolean array checking. Hence;
$\tau_6(1) = \varepsilon_3 \cdot \phi_{tt} \cdot P_t$,
where, ε_3 is the fraction of the target relation tuples that survives the checking of the *Boolean array sorting* during the execution of one phase of a *global broadcast* target relation algorithm. A formula that relates ε_3 to the model's input parameters is;
$\varepsilon_3 = [1 - Exp(-\phi_{ts}/\varepsilon_{ms} \cdot (\gamma_c)) \cdot Exp(-\varepsilon_8/\mu_B)]$
[The details of deriving this formula have refrained from here because of space.]

To join one source MACU with all of the MACU's of the < *global broadcast, local hash,* target relation algorithm>, will first redistribute among the PEs within the cluster the tuples of a source MACU and those tuples of the target relation which survive the Boolean array checking. The tuple redistribution is performed using a hashing function and on the average only $(1 - (1/N_{PE}))$ of both tuples of the source MACU and the surviving tuples of the target relation need to be moved to the other PEs in the cluster.

Therefore,
$\tau_6 = [(\phi_{ts} \cdot P_s / \varepsilon_{ms}) + (\varepsilon_3 \cdot \phi_{tt} \cdot P_t)] \cdot (1 - (1/N_{PE}))$.
where [NTS/ε_{ms}] of the above equation is the average number of tuples in a source MACU.

During the execution of one phase of a <X= *global hash, target partial* algorithm> the tuples of one global bucket are joined.

In joining them a <*global hash, local broadcast, target partial* algorithm> broadcasts to the PEs of the cluster those target tuples which hash to the global bucket (phase) and survive the *Boolean* array checking.

Therefore, $\tau_6(1) = \varepsilon_7 \cdot \phi_{tt} \cdot P_t$,
where ε_7 is the fraction of the target relation tuples which hash to a global bucket and survive the checking of the *Boolean Array* during the execution of one phase of a *global hash* target partial algorithm.

A formula that relates ε_7 to the model's input parameters is;
$\varepsilon_7 = (1/\varepsilon_5) \cdot (1 - \mathrm{Exp}(-\varepsilon_6/\varepsilon_4) \cdot \mathrm{Exp}(-\varepsilon_8/\mu_B))$, where,
$\varepsilon_8 = (\varepsilon_4/\varepsilon_5) \cdot (1 - \mathrm{Exp}(-\varepsilon_6/\varepsilon_4))$.
The derivation is refrained from here because of space.

To join the tuples of a global bucket, a *global hash-local hash*, target relation algorithm, redistributes using a hashing function the source tuples of the global bucket and those target tuples of the same buckets that survive the checking of the *Boolean* array among the clusters PEs.,
Hence; $\tau_6(1) = [(\varepsilon_6/\varepsilon_5).TS + \varepsilon_7.\phi_{tt}.P_t](1 - (1/N_{PE}))$.

Next is how to derive a formula for $\tau_5(1)$, which can be expressed as follows:
$\tau_6(1) = \tau_2(1) + \tau_7(1) + \tau_4(1)$.

The input parameters for the above formula are derived next for each joint algorithm.
(a) Equation for $\tau_2(1)$;

When executing one phase of a <*global broadcast, local broadcast compare, sort or hash-target partial algorithm*>, the Boolean array hash function is used. This function is computed by every PE for its share from both the tuples of a source MACU and all of the tuples of the target relations.

The <*global broadcast, local broadcast, hash-target partial algorithm*> algorithm uses the hash table function. This function is computed in each phase by every PE for its share from the tuples of a source MACU and those tuples of the target relation which survive the Boolean array checking, therefore, for <*global broadcast, local broadcast compare or sort-target partial algorithm*> algorithm, we have;
$\tau_2(1) = [(\varepsilon_6/(\varepsilon_{ms}.N_{PE}) + (\phi_{tt}/N_{PE})]\tau_h = \tau\tau$
In the same way for <*global broadcast, local broadcast, hash-target partial algorithm*> algorithm we have;
$\tau_2(1) = \tau\tau + [(\varepsilon_6/(\varepsilon_{ms} \cdot N_{PE}) + \varepsilon_3 \cdot \phi_{tt}] \cdot \tau_h$.

In addition to the Boolean array hash function, a PE executing one phase of the joint algorithms, (<*global broadcast, local hash, (compare sort or hash-target partial* algorithms>), compute the local hash function for its share from both of the tuples of a source MACU and those tuples of the target relation, which survive the Boolean array checking.

Also, the purpose of executing one phase of the <*global broadcast, local hash, hash-target partial* algorithm> uses another hash function, (the hash table function) which is computed by every PE for its share from the tuples of a source MACU and those target tuples that survive the Boolean array checking and hash to the PE itself, therefore, for <*global broadcast, local hash, compare or sort-target partial* algorithms> algorithm we have;
$\tau_2(1) = [\varepsilon_6/(\varepsilon_{ms} \cdot N_{PE}) + (\phi_{tt}/N_{PE})] \cdot \tau_h + [\varepsilon_6/(\varepsilon_{ms} \cdot N_{PE}) + (\varepsilon_3 \cdot \phi_{tt})/N_{PE}]] \cdot \tau_h = \tau\tau\tau$;
and for <*global broadcast, local hash, hash-target partial* algorithm> algorithm we have;
$\tau_2(1) = \tau\tau\tau + [\phi_{ts}/(\varepsilon_{ms} \cdot N_{PE}) + (\varepsilon_3 \cdot \phi_{tt})/N_{PE}] \cdot \tau_h$.

When executing a global hash algorithm the global hash function is used to partition tuples between phases. This function is computed in each phase by every PE for its share from all of the tuples of the source and target relations.

Also, a PE executing one phase of the <*global hash, local broadcast, compare, sort, or hash-target partial algorithm*> type joint algorithm computes the *Boolean array* hashing function for its share from those *source* and *target* tuples which hash to one global bucket. Also, a PE executing one phase of the <*global hash, local broadcast, hash-target partial algorithm*> computes the hash table function for its share from the source tuples which hash to one global bucket and survive the Boolean array filtering.

Hence, for <*global hash, local broadcast, compare, or sort -target partial algorithm*> we have;
$\tau_2(1) = H \cdot [(\varepsilon_6/N_{PE}) + (\phi_{tt}/N_{PE})] \cdot \tau_h + [(\varepsilon_6/(\varepsilon_5 \cdot N_{PE})) + \phi_{tt}/(\varepsilon_5 \cdot N_{PE})] \cdot \tau_h = \tau\tau\tau\tau$.
and for <*global hash, local broadcast, hash-target partial algorithm*> we have;
$\tau_2(1) = \tau\tau\tau\tau + [\varepsilon_6/(\varepsilon_5 \cdot N_{PE}) + \varepsilon_7 \cdot \phi_{tt}] \cdot \tau_h$,
When $H = 0$, then $\varepsilon_5 = 1$, otherwise $H = 1$.

In addition to global hash function a PE executing the join algorithm <*global hash, local hash, compare, sort or hash -target partial algorithms*> uses another hash function, the local hash function which is computed in each phase by every PE for its share from those source and target tuples that hash to one global bucket. Also, the PE uses the Boolean array hash function which is computed in each phase by every PE for its share from those source and target tuples that hash to one global bucket.

Also, <*global hash, local hash, hash -target partial algorithms*> join algorithm uses another hash function, the hash table function, which is computed in each phase by every PE for the source tuples which hash to one global bucket and one PE and for those target tuples which hash to one global bucket survive the Boolean array sorting and hash one PE. Hence, for <*global hash, local hash, compare or sort -target partial algorithms*> join algorithms we have;

$\tau_2(1) = H\,[(\varepsilon_6/N_{PE}) + (\phi_{tt}/N_{PE})]\cdot\tau_h + 2\,[\varepsilon_6/(\varepsilon_5\cdot N_{PE}) + \phi_{tt}/(\varepsilon_5\cdot N_{PE})]\cdot\tau_h = \tau\tau\tau\tau\tau$

Also for *<global hash, local hash, hash -target partial algorithms>* join algorithm we have;

$\tau_2(1) = \tau\tau\tau\tau\tau + [\varepsilon_6/(\varepsilon_5\cdot N_{PE}) + (\varepsilon_7\cdot\phi_{tt}/N_{PE}]\cdot\tau_h$.

When a PE sorts a table of pointers on a set N tuples is on the average, N $\log_2$ N comparisons and pointer swaps; thus,

$\tau_7(1) = (\tau_c + \tau_s)\cdot N\cdot\log_2 N$. Where N is the average number of source tuples to be sorted by a PE during the execution of one phase of the target relation algorithms.

For *<global broadcast, local broadcast, sort -target partial algorithms>* we have; $M = \varepsilon_6 / (\varepsilon_{ms}\cdot N_{PE})$.

For *<global broadcast, local hash, sort -target partial algorithms>*

we have; $M = \varepsilon_6 / (NMS\cdot N_{PE})$.

For *<global hash, local (hash or broadcast), sort -target partial algorithms>* we have $M = \varepsilon_6 / (\varepsilon_5\cdot N_{PE})$.

In general $\tau_4 = \tau_c\cdot EN(1)$,

Where EN(1) is the expected number of join attribute value comparisons performed by a PE during the execution of one phase of one of the join algorithms.. If N_1 or N_2 are the average expected number of source and target tuples, respectively, assigned by one PE during the execution of one phase of a target partial join algorithm, then;

$EN(1) = N_1\cdot N_2$

If a PE uses a sorting method for joining a target tuple with N_2 source tuples uses the binary search method to locate the pointers to the linked list that stores those tuples whose join attribute value match those of the target tuples then the number of comparisons done by PE is $\log_2 N_2$, then;

$EN(1) = N_1\cdot\log_2 N_2$.

How to determine the best performance;

In this model the hardware input parameters are kept constant. The other parameters are varied. The τ and the number of hashing functions are used to evaluate the performance of the joint-algorithm.

In general the best performing algorithm among a set containing others over a range of a parameter value is the one with the smallest values to τ over that range. If several join algorithms have close τ values over that range, then the best performing algorithm is the one with the minimum number of hash functions.

$(N_{PE}, \Sigma, \tau_{ave.}, \tau_d, \tau_{td})$ hardware parameters that characterize the moving head disk. The disk is assumed to be modified for parallel reading (writing) from (to) a whole cylinder.

The time to read(write) one track of the disk is assumed to be the value of reading (writing) the whole cylinder (τ_{td}). It also assumed that τ_{pb} has the same value as that of τ_{td}.

The hardware parameters that characterize a PE is $(\tau_c, \tau_c, \tau_s, \tau_h)$, off-the-shelf PEs' parameters.

The parameters that characterize the tuple bus of a cluster (P_s, P_t) are calculated assuming that the bus has effective bandwidth of 1Mbyte.

We assume that ϕ_{ts} and ϕ_{tt} have the same values. γ_c, changes directly by changing the ratio (ε_6/N_{PE}).

R_2 is the ratio of the number of buckets in a PE's hash table, and the PE's expected number of source tuples per one phase of a Join join algorithm, and R_1 is the ratio of the number of bits in the Boolean array(s) and the number of tuples in both the source and target realtions, respectively. R_2 and R_1 are related to the amount of storage allocated to a hash table and to the Boolean array(s), respectively.

The best performing joint algorithm can be determined by two steps;

(1) The performance of the algorithm is analyzed for the variable input parameters, for which we can choose the best performing algorithm.

(2) The best performing algorithms are compared to one another to find out the best over-all performing one.

6- Conclusion

We have used the join models in this paper to access the effectiveness of tuning the multiprocessor structure which executes the join operation. Such an investigation has explained the effectiveness of using the Boolean array filtering techniques to speedup such an execution without substantially increasing the hardware cost.

文献(References)

[De85] Dewitt, D.J., et.al., "Multiprocessor hash based join algorithms," in Proc. 11th Int. Conf. VLDB, 1985, pp.151-164.

[Ha87] Hamid, I.A., et.al.,"A new fast control mechanism for rearrangeable interconnection network useful for supersystems," *Tran. IEICE*, vol. E70 No. 10, Oct. 1987, pp. 997-1008.

[Ha88] Hamid, I.A., et.al.,"A new controlling algorithm for Benes interconnection network without symmetry," *Trans., IEICE*, vol.E71, No.9, E71, Sep., 1988, pp.895-904.

[Ha89] Hamid, I.A., et.al.,"A new fastparallel computation model for setting Benes rearrangeable interconnection network," *Trans., IEICE*, vol.E72, No.4, April, 1989, pp.393-405.

[Go81] Goodman, J.R., "An investigation of multiprocessor structures and algorithms for database mangement," Reprot of Univ., of Calif., Berkely, May, 1981.

[Go83] Gottlieb, A., et. al., "The NYU Ultracomputer designing an MIMD shared memory parallel computer," IEEE, Feb., 1983, pp. 175-1189.

[Ki84] Kitsuregawa, K. et.al., "Arcitecture and performanc of relational algebra machine GRACE," in Proc. Int. Conf. Parallel Processing, 1984, pp.241-250.

[Qa85] Qada, G.Z., et.al., "A database machine for very large relational databases," IEEE Trans. Comput., vol.C34, pp.1015-1025, Nov. 1985.

[St84] Stone, H., "Database applications of the Fetch-and-add instruction," IEEE, Jan., 1984, pp. 604-612.

[Va84] Valduriez, P., et.al., "Join and semijoin algorithms for a multi-microprocessor database machine," ACM Trans. Dtatbase Syst., vol. 9., No. 1, pp.133-161, March, 1984.

Acknowledgment

The authors would like to thank NEC-Tohoku Software group, in the *large computer center* of *Tohoku University*, for their cooperation in this research. Also, much thanks with deep appreciation for *Miss* Ito Kyoko for her never-ending advices and the creative discussion led to this paper.

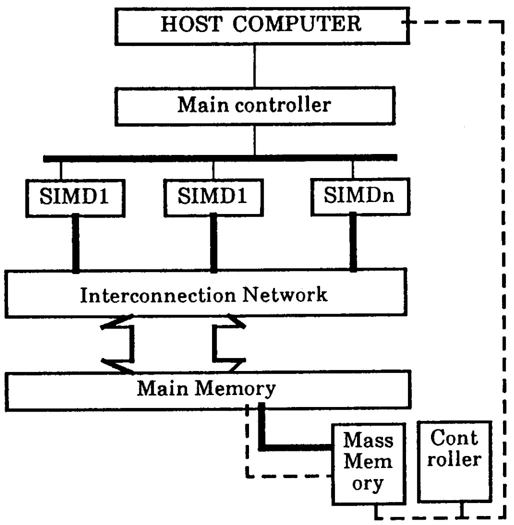


Fig. 1, System outline for parallel Database machine

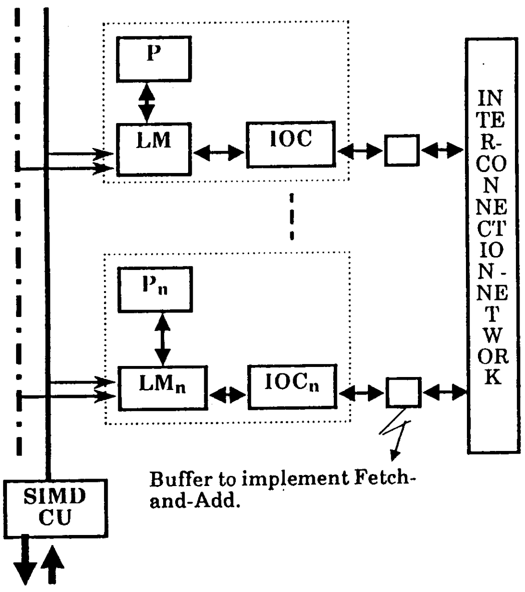


Fig. 2, One SIMD subsystem

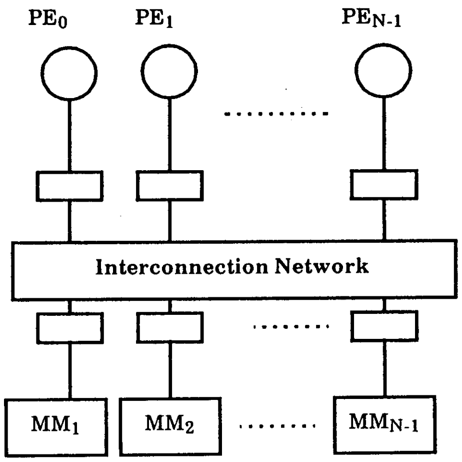
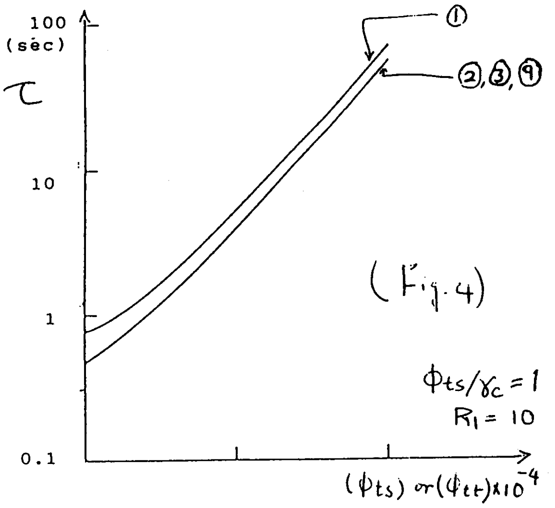


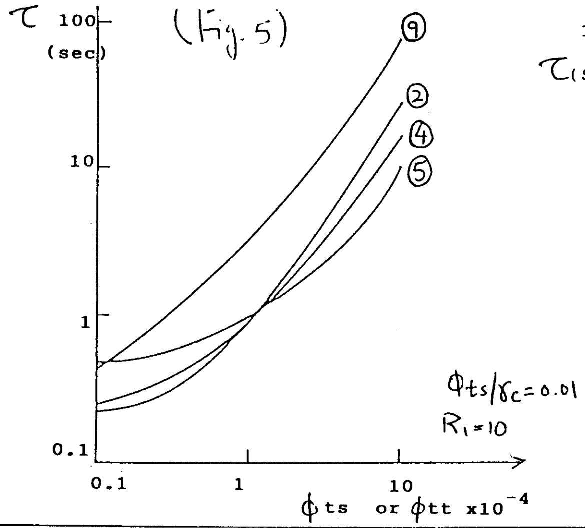
Fig. 3, Rearrengiable Interconnection network with buffers at the input and output of it.

Notation used in the Figs. 4 to 8.

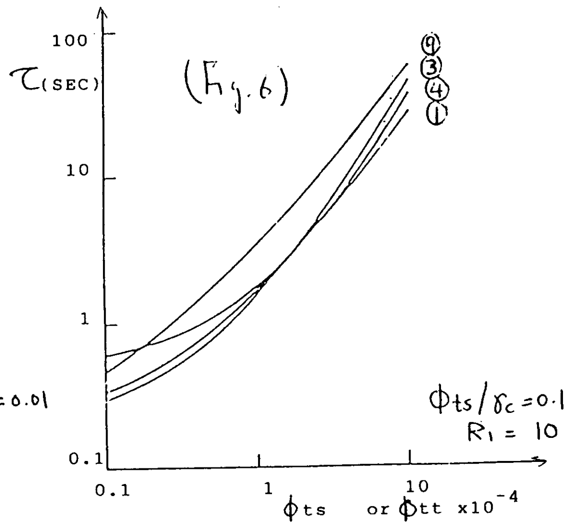
① = <	Global hash, local broadcast, sort, source tuple complete algorithm >
② = <	Global hash, local broadcast, sort, source tuple partial algorithm >
③ = <	Global hash, local broadcast, sort, target tuple partial algorithm >
④ = <	Global hash, local broadcast, sort, target tuple partial algorithm >
⑤ = <	Global broadcast, local broadcast, sort, source tuple, complete algorithm >
⑥ = <	Global broadcast, local hash, sort, basic algorithm >
⑦ = <	Global broadcast, local hash, sort, target partial algorithm >
⑧ = <	Global broadcast, local hash, sort, source tuple complete algorithm >
⑨ = <	Global hash, local broadcast, sort, basic algorithm >



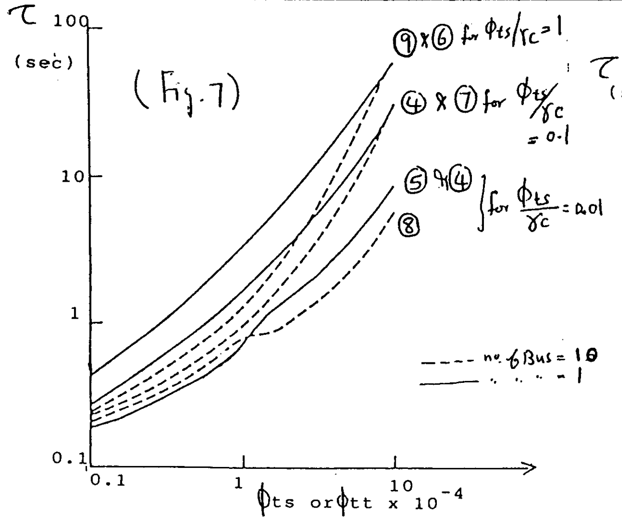
(Fig. 4)



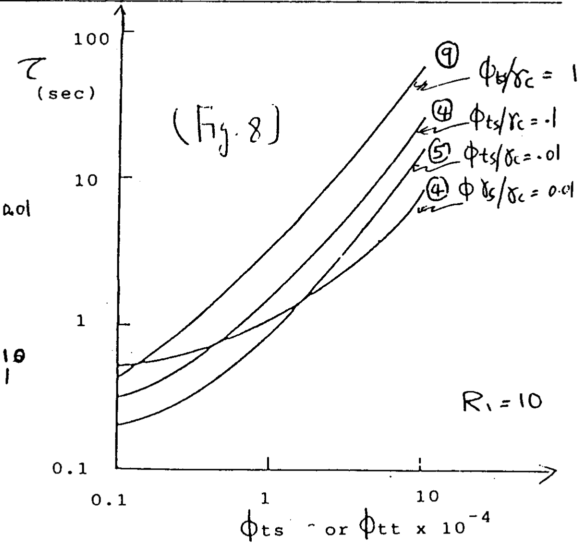
(Fig. 5)



(Fig. 6)



(Fig. 7)



(Fig. 8)