An Efficient Parallel Join Algorithm Based on Hypercube-Partitioning

Hwan Ik Choi, Byoung Mo Im, Myoung Ho Kim, Yoon-Joon Lee

Dept. of Computer Science, Korea Advanced Institute of Science and Technology

Abstract

Many parallel join algorithms have been proposed so far but most of which are developed focused on minimizing the disk I/O and CPU costs. The communication cost, however, is also an important factor that can significantly affect the join processing performance in multiprocessor systems. In this paper we propose an efficient parallel join algorithm, called Cube-Robust, for hypercube multicomputers. The proposed algorithm is developed based on the observation that the size ratio of two relations to be joined is the dominant factor in the communication cost. We develop the analytic cost model for the proposed join algorithm. The performance comparisons show that the Cube-Robust join algorithm works better than others proposed earlier in a wide range of size ratios.

1. Introduction

The join operation is the most time consuming and frequently used one in relational database systems. Developments of efficient parallel join algorithms are necessary to fully utilize the processing power of distributed and parallel systems.

Many algorithms have been proposed for parallel processing of join operation[1, 4, 5, 9-12]. However, most of them focus only on reducing disk I/O operations. As VLSI technology advances, the main memory becomes cheaper and there exist general machines equipped with more than 1G byte memory. Many researches report the feasibility of the database management system for large main memory computers. In the case of multiprocessor systems with large main memory, the communication cost becomes another important factor in performance. There were some researches to reduce the communication cost. [8] tried to minimize the communication cost occurred during the result collection phase with the help of the specially designed network. In [5] the packet overhead was considered and the *relation-compaction-and-replication* phase was introduced.

Parallel Join algorithms can be classified into two main categories, *broadcast-based approach* and *bucket-based approach*, according to the distribution strategies of relations[6]. The broadcast-based approach requires that the fragment of a relation travels all the system nodes. Once the fragment of a relation arrives at a node, the node performs partial join operation with its own fragment of the other relation. After perfroming a partial join operation, a node sends the received fragment of relation to the next node. The broadast-based approach shows relavtively low communication cost. However, its performance is significantly dependent on the cost incurred by *false-join*. A false-join is the join operation that does not produce any result tuples. Increased number of false joins causes extra CPU cost. Figure 1 shows the logical diagram of the broadcast-based approach.

The bucket-based approach partitions two relations to be joined into disjoint buckets according to their attribute values. A bucket consists of tuples that have the same characteristics, and is assigned to a certain node. Each node contains distinct buckets. Usually, a bucket is formed based on hash values of join attributes. After buckets are distributed, all nodes perform join operations of two buckets in parallel. The bucket-based approach uses a "divide and conquer" strategy, because a large join operation is divided into smaller independent join operations. The bucket-based approach has a performance advantage due to the clustering effect: each cluster contains tuples that are likely to be joined. Clustering effect reduces the number of false-joins so that the bucket-based approach wastes less CPU time. However, the bucket-based approach must pay the communication cost for the reduced CPU cost because both relations have to be distributed. Figure 2 shows the logical diagram of the bucket-based approach

In this paper we propose an efficient parallel join algorithm, called *Cube-Robust*, for cube-connected multicomputers. Our main focus on developing Cube-Robust is to reduce the communication cost among processing elements in multiprocessor systems based on the size ratio of two relations. Here, the size ratio α of two relations is defined as follows: $\alpha = \frac{\textit{size of larger relation}}{\textit{size of smaller relation}}$.

0-8186-6400-2/94 $4.00

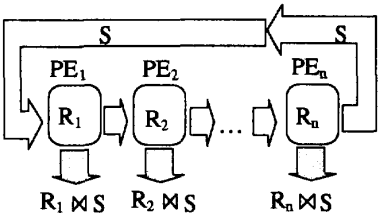


Figure 1. Broadcast-based approach

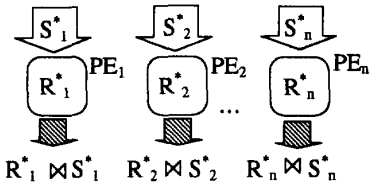


Figure 2. Bucket-based approach

Since our work is mainly concerned with efficient parallel join processing of the relations loaded into memory of some processing elements, all other works on the reduction of I/O costs can be equally applied to our scheme.

The remainder of this paper is organized as follows. In section 2 we proposes a new parallel join algorithm, Cube-Robust, on hypercube systems. We develop an analytic cost model for the proposed join algorithm in section 3 and performance comparisons are presented in section 4. Finally conclusions are presented in section 5.

2. Proposed parallel join algorithm

2.1. Parallel join algorithms for hypercubes

A. The Cube-Hybrid-Hash join algorithm

The Cube-Hybrid-Hash join algorithm(CHH)[9] is a straight adaptation of the single processor hybrid hash[4] join algorithm. CHH uses a bucket-based approach because CHH partitions each relation into disjoint buckets. As relations are partitioned into appropriate buckets, CHH can minimize the number of false-joins that in turn results in reduced CPU cost. However, the cost of distributing two relations may overcome the benefits of clustering effect as the size ratio of two relations increases.

B. The Cube-Nested-Loop join algorithm

The Cube-Nested-Loop join algorithm(CNL) adopts the broadcast-based method[11]. In CNL, a node reads the smaller relation and constructs the local hash table. After the construction of the local hash table, the node reads the larger relation S and performs local join operations if possible. All tuples of the larger relation are moved to the output buffers and are exchanged with its neighbors. For the received tuples, a node tries to perform the local join. According to the exchange steps, the larger relation is broadcasted among all nodes in the system.

CNL adopts a broadcast-based approach as the larger relation is broadcasted among nodes in the system. Since the fragments of the larger relation in each node have to be compared with all the tuples of the smaller relation as in the simple nested-loop join algorithm, it may suffer from the high processing cost. Although CNL is a broadcast-based algorithm, CNL does not take the advantage of the broadcast-based approach due to the broadcasting of the larger relation.

C. The Modified-Cube-Nested-Loop join algorithm

To analyze the characteristics of the broadcast-based approach more precisely we modified the CNL join algorithm to broadcast the smaller relation.

First a node reads the small relation and constructs the local hash table as in the CNL join algorithm. The tuples of the small relation are moved to output buffers and broadcasted among nodes in the system. The MCNL join algorithm reduces the communication cost by broadcasting the small relation in contrast to CNL. After broadcasting a node reads the larger relation and performs local join operations.

2.2. Basic concepts

The following observation is the basis on which our proposed algorithm is developed. The CPU processing cost of the broadcast-based approach is larger than the bucket-based approach in general. This is because all possible pairs of tuples in two relations have to be compared in the broadcast-based approach. On the other hand, the communication costs of these two approaches depend on several factors. First, suppose the sizes of two relations to be joined are almost the same(i.e., $\alpha \approx 1$). In a small dimension hypercube, the communication cost of the broadcast-based approach is less than that of the bucket-based approach. It, however, grows very fast and tends to exceed that of the bucket-based approach as the dimension of a hypercube increases. Second, if the size of one relation is much larger than that of the other relation(i.e., $\alpha \gg 1$), the communication cost of the broadcast-based approach can be significantly less than that of the bucket-based approach. Note that the broadcast-based approach broadcasts only one, i.e., the small relation while the bucket-based approach distributes both relations. In other words, the characteristics of communication costs in these two approaches have to be analyzed based on both the hypercube size and the size ratio of two relations.

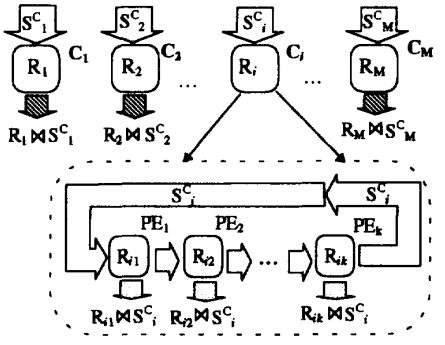


Figure 3. Basic idea of proposed algorithm

In an architecture based on hypercube interconnection, each node is connected(i.e., adjacent) to each of its $n \times log_2 N$ neighbors, where N is the total number of nodes. Since the hypercube possesses many desirable characteristics such as a low diameter, the high bisection width and symmetry, the hypercube structure has been much considered for many parallel processing applications[7].

To take advantages of two distribution strategies, the join algorithm must be intelligent enough to select a proper strategy based on the size ratio of two relations and the number of processors. In order to control the distribution strategy we have introduced the *hyperbucket* concept. A hyperbucket is a processor cluster and is a k-dimension subcube of n-dimension hypercube system ($0 \le k \le n$). An n-dimension hypercube has 2^{n-k} hyperbuckets. Thus, we can consider that an n-dimension hypercube is an $(n-k)$-dimension hypercube where each node is a hyperbucket. In the CR join algorithm, each relation to be joined is partitioned into 2^{n-k} fragments, and then the CR join algorithm distributes each fragment to hyperbuckets. Here, fragments of smaller relation are replicated to all the nodes within the same hyperbucket.

The communication cost in CR consists of inter- and intra-bucket communication costs. Inter-bucket communication occurs while fragments of relations R and S are distributed into hyperbuckets. Intra-bucket communication is necessary to replicate a received fragment of smaller relation to all the nodes within the same hyperbucket. The dimension of hyperbucket k is selected to optimize these two types of communication costs. CR selects the dimension of hyperbucket based on the size ratio α. Figure 3 shows the logical operation diagram of proposed processing scheme.

2.3. The Cube-Robust Join algorithm

The proposed CR join algorithm consists of four phases.

• Hyperbucket selection phase

In this phase, CR selects a dimension k of hyperbucket. The dimension k that minimizes the communication cost for both relations is

$$k = \max(\min(\left\lfloor \log_2 \frac{1+\alpha}{2 \times \ln 2} \right\rfloor, n), 0), \text{ where } \alpha \text{ is } \frac{|S|}{|R|}.$$

The equations are shown in [3].

• Bucket construction phase

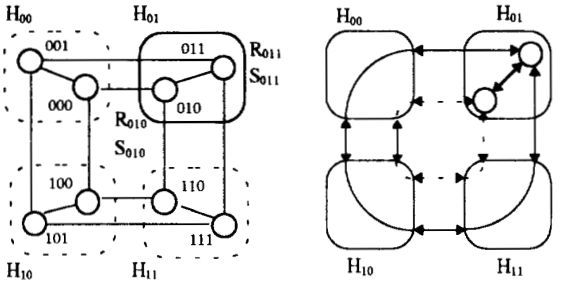
After the dimension k of hyperbucket is selected, an n-cube is considered as an $(n-k)$-cube with 2^{n-k} nodes(hyperbuckets). In the bucket construction phase two relations are partitioned into 2^{n-k} disjoint buckets and each bucket is distributed to a hyperbucket H_i ($0 \le i < 2^{n-k}$). Let $<i_{n-1}, i_{n-2}, \ldots, i_0>$ be the binary representation of i (a subscript 0 indicates the least significant bit). Hyperbucket H_h contains 2^k nodes whose binary representation of addresses is $< h_{n-k-1}, h_{n-k-2}, \ldots, h_0, * >$, where * denotes "don't care term". That is, nodes in a hyperbucket have the same $(n-k)$-bits in their address representation. For example, in Figure 4-(a) the dimension of hyperbucket is 1 and the 3-dimension hypercube is divided into 4 hyperbuckets. The hyperbucket $H_{<0,1>}$ consists of two nodes <0,1,0>, <0,1,1> whose the most significant two bits are <0,1>.

Each node allocates one output buffer for each link. The output buffer B_i ($0 \le i < n-1$) is associated with the neighbor whose address differs in the least significant i-th bit position with that of the local node. The B_i of a node contains tuples whose destined locations have the identical bits in the i-th to $(n-1)$-th positions to the address of its associated neighbor. That is, the B_i of node j contains the tuples for nodes whose address is represented as $< j_{n-1}, j_{n-2}, \ldots, j_{n-i-1}, * >$. Neighbor i is defined in the same manner.

Node i reads local fragments R_i, S_i and applies a hash function to get the destination of a tuple. Once the destination of a tuple is decided as H_h, this tuple is sent to the node i whose binary representation is $< h_{n-k-1}, h_{n-k-2}, \ldots, h_0, i_{k-1}, i_{k-2}, \ldots, i_0 >$. In Figure 4-(b) the tuples in the node <0,1,1> whose destination is $H_{<0,0>}$ are transmitted to the node <0,0,1> and tuples whose destination is $H_{<1,1>}$ are transmitted to the node <1,1,1>. Note that tuples in the node <*,*,1> never go to nodes <*,*,0>.

• Broadcast phase

In the broadcast phase, tuples of the smaller relation belong to k-dimension hyperbucket are replicated to all 2^k nodes in the hyperbucket. Each hyperbucket performs tuple replication operations in parallel. To fully replicate tuples, k steps of communications are necessary. For



(a) Hyperbuckets (b) Communication paths

Figure 4. 3-dimension hypercube with 4 hyperbuckets

$\|R\|$	size of the relation R(tuples)
T_R	size of tuple in R(bytes)
P	size of a page(bytes)
P_{occ}	average page occupancy factor
P_R	number of tuples per page for R, $\left\lfloor \frac{P \times P_{occ}}{T_R} \right\rfloor$
IO	time in reading/writing a page from/to the disk(*msec*)
n	dimension of hypercube
N	the number of nodes in hypercube, 2^n
k	dimension of hyperbucket
M	the number of nodes in hyperbucket, 2^k
A	size of a node address(bytes),
packet	maximum size of a packet(bytes)
comm	communication rate(bits/second)
cmp	time in comparing two values(μs)
hash	time in computing the hash value of a tuple(μs)
move	time in moving one tuple within main memory(μs)
P_{ovhd}	packet overhead for control information (*sec/packet*)
F	fudge factor

Table 1. Evaluation notations and values

example, in Figure 4-(b) the tuples of relation R in node <0,1,1> belonged to hyperbucket $H_{<0,1>}$ are transmitted to node <0,1,0> and simultaneously the tuples in node <0,1,0> are transmitted to node <0,1,1>.

By replicating the portion of smaller relation in a hyperbucket, each node in the hyperbucket contains the same fragment of the relation smaller relation R so that hyperbucket can be regarded as a single bucket but has 2^k processors.

• Local join phase

This phase performs local joins. We choose Hash Join as the local join method but other algorithms can also be used. According to the hash join method, we build a hash table in memory on each node for the tuples that have been broadcasted. Then for every tuples in the fragment of S that have been transmitted by the bucket construction phase, the node probes the local hash table to find matching tuples. Since all 2^k nodes of a hyperbucket H_i contains the same hash table, a tuple of S whose destination is H_i can be probed by any nodes in the H_i. The complete algorithm of CR is given in [4].

3. Cost Models

The hypercube machine under investigation is assumed to have bi-directional communication channels and each channel has its own communication buffer to provide concurrent data transfer.

In the proposed cost model, the size of relation S is always the larger than that of relation R. We assume both relations R and S are partitioned horizontally across the nodes and there are no data value skew. The number of comparisons required per tuple to probe the hash table is controlled by the fudge factor F that is larger than 1 due to potential collisions.

I/O and CPU overlap is not considered to make the cost model simple. The memory and output buffers are assumed to have enough space not to occur overflow. Final output costs are not included. Table 1 shows the notations we use to develop the cost models

3.1. Cost Model of Cube-Robust Join Algorithm

We analyze the cost of Cube-Robust join algorithm in the following.

• Disk I/O cost on a single node:

Since we assume enough space not to overflow, disk I/O cost of all algorithms to be compared with are same. Assuming a uniform tuple distribution, the access time is

$$C_{IO} = \left(\left\lceil \frac{\|R\|}{N \times P_R} \right\rceil + \left\lceil \frac{\|S\|}{N \times P_S} \right\rceil \right) \times IO .$$

• CPU cost for relation R:

The processing cost of relation R includes the cost C_{IR} to determine the destination of a tuple, the C_{DR} cost to construct bucket and the cost C_{BR} to replicate tuples in hyperbuckets.

To determine the destination of a tuple, a node examines tuples of local fragments. If the destination of a tuple is the local node or nodes in the hyperbucket to which the local node belongs, the tuple is moved to the local hash table. Otherwise, it is moved to an output buffer for further transmissions.

$$C_{IR} = \begin{cases} \frac{\|R\|}{N} \times \left[(hash + cmp) + \frac{N-M}{N} \times move + \frac{M}{N} \times move \right], & k \neq n \\ \frac{\|R\|}{N} \times (hash + 2 \times move), & otherwise \end{cases}$$

After locating tuples to the proper buffers, a node exchanges tuples in the output buffer B_i with its neighbor N_i $(n-k-1 \le i < n)$. At each step of exchange, the amount of incoming tuples is $\|R\|/(2\times N)$ due to the doubling effect [9]. The address of the arrived tuple is examined and moved to the output buffer or the hash table. Note that tuples arrived by exchanging B_i only enter the output buffer $(0 \le j < i)$. This is because the buffer B_i contains the tuples whose destined addresses are different in the least significant i-th position with that of the local node. Therefore, $M/2^i$ portion of incoming tuples are entered into the local hash table and at i-th step is

$$C_{DR_i} = \frac{\|R\|}{2\times N}\times\left[cmp + move + \frac{M}{2^i}\times hash\right].$$

Once the distribution of the tuples are completed, the local node replicates tuples among nodes in the hyperbucket and builds a hash table. Each node contains $\|R\|/N$ tuples in its local hash table and the local node receives tuples from (M-1) nodes. The cost for broadcast C_{BR} is

$$C_{BR} = (M-1)\times\frac{\|R\|}{N}\times(hash + 2\times move).$$

The total cost to process R is $C_{PR} = C_{IR} + \sum_{i=k}^{n-1} C_{DRi} + C_{BR}$.

• **CPU cost for relation *S*:**

The processing cost of relation S includes initial processing time, the time to construct bucket and the time to probe the local hash table for local joins.

The initial processing cost C_{IS} is the cost necessary to determine the destination of tuples in the relation S and is derived in the similar way to that of the relation R. Instead of moving tuples to the local hash table, tuples of the relation S are probed with the local hash table.

$$C_{IS} = \begin{cases} \frac{\|S\|}{N}\times\left[hash + cmp + \frac{N-M}{N}\times move + \frac{M}{N}\times cmp\times F\right], & k \ne n. \\ \frac{\|S\|}{N}\times(hash + cmp\times F), & \text{otherwise.} \end{cases}$$

At each step i, if a tuple is destined to the local node it is probed against the local hash. Otherwise, it is moved to an output buffer:

$$C_{DSi} = \frac{\|S\|}{2\times N}\times\left[(cmp + move)\times\left(1-\frac{M}{2^i}\right) + (hash + cmp\times F)\times\frac{M}{2^i}\right]$$

The total cost is $C_{PS} = C_{IS} + \sum_{i=k}^{n-1} C_{DSi}$.

• **Communication cost for relation *R*:**

The cost includes the time to transmit the buckets for relation R and the time to replicates buckets in the hyperbucket. The communication cost for bucket distribution is

$$C_{RD} = (n-k)\times\left[\frac{\|R\|\times(T_R + A)\times 8}{2\times N\times comm} + \left\lceil\frac{\|R\|\times(T_R + A)}{2\times N\times packet}\right\rceil\times P_{ovhd}\right].$$

After buckets are distributed, the replication phase is performed over k-steps. The number of tuples to be sent is doubled at each step. This cost is represented by

$$C_{RBi} = 2^i\times\frac{\|R\|}{N}\times T_R\times\frac{8}{comm} + \left\lceil\frac{2^i\times\frac{\|R\|}{N}\times T_R}{packet}\right\rceil\times P_{ovhd}.$$

The total cost associated with the communication is: $C_{commR} = C_{RD} + \sum_{i=0}^{k-1} C_{RBi}$.

• **Communication cost for relation *S*:**

The relation S only needs to be distributed over the (n-k)-dimension hypercube.

$$C_{commS} = (n-k)\times\left[\frac{\|S\|\times(T_S + A)\times 8}{2\times N\times comm} + \left\lceil\frac{\|S\|\times(T_S + A)}{2\times N\times packet}\right\rceil\times P_{ovhd}\right].$$

• **Total cost:**

The total cost is $C_{PR} + C_{PS} + C_{commR} + C_{commS}$.

3.2. Cost Model of Modified-Cube-Nested-Loop Join Algorithm

The cost of the Modified-Cube-Nested-Loop join algorithm is analyzed in the following.

• **Disk I/O cost on a single node:**

The cost incurred by disk I/O operations is same with that of the CR algorithm.

• **CPU cost for relation *R*:**

The cost includes the cost to hash every tuples of R and to store them in the local hash table. Each node should make the hash table for all tuples of relation R: $C_{IR} = \|R\|\times(hash + 2\times move)$.

• **CPU cost for relation *S* :**

The cost includes the cost to hash every tuples of S and to probe the local hash table for local join operations: $C_{IS} = \frac{\|S\|}{N}\times(hash + comp\times F)$.

• **Communication cost :**

The cost includes the transmission of tuples over n phases for the relation R. In the MCNL join algorithm, only the smaller relation is broadcasted over the nodes in the system. As in the CR join algorithm, the size of tuples transmitted is doubled during each phase of broadcast.

$$C_{comm} = \sum_{i=0}^{n-1} C_{Ci},$$

$$\text{where } C_{Ci} = 2^i \times \frac{\|R\| \times T_R \times 8}{N \times comm} + \left\lceil \frac{2^i \times \frac{\|R\| \times T_R}{N}}{PS} \right\rceil \times P_{ovhd}.$$

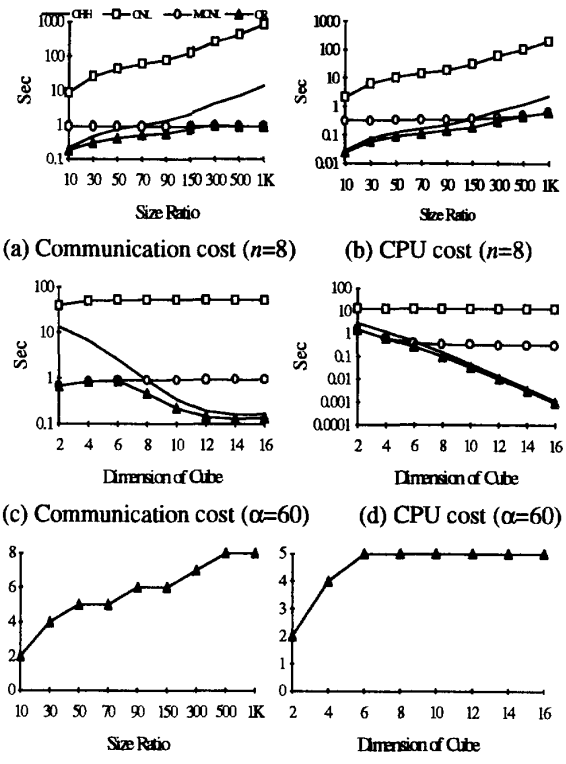
- **Total cost:**
  The total cost is $C_{IR} + C_{IS} + C_{comm}$.

4. Experimental Results

This section analyzes the performance of four parallel join algorithms Cube-Hybrid-Hash(CHH), Cube-Nested-Loop(CNL), Modified-Cube-Nested-Loop(MCNL) and Cube-Robust(CR) through the analytical cost models described in section 3. The cost models for CHH and CNL algorithms are identical with [9] and [11] respectively. In this experiment we measure the response time of four parallel join algorithms when the size ratio α is varied. We evaluate performance of four join algorithms mainly focused on the communication and CPU costs. The disk I/O cost is constant in all algorithms as we stated in section 3 and we do not take into account the performance effects of disk I/Os. We investigate two cases. The first experiment is to compare the communication and CPU costs of four algorithms. The second experiment is to analyze the overall performance of four algorithms. All performance evaluations are done by increasing the size of relation *S* while the size of relation *R* is fixed. The values of system parameters used in our analysis can be found in [4, 9, 11].

A. Communication Cost vs. CPU Cost

Figure 5-(a) shows the communication overheads of four join algorithms. The communication cost of MCNL depends on the size of the relation *R* because it is the only relation broadcasted. Due to the fixed size of relation *R*, MCNL shows the constant communication cost. CNL shows the worst case because the larger relation *S* is broadcasted. As α increases the communication cost of CHH inclines and becomes higher than that of MCNL when α is greater than 60. This means when the dimension of the hypercube is 8, the broadcast-based approach is superior to the bucket-based approach once α is greater than 60. CR which reduces the hypercube system into hyperbuckets based on α shows the best performance. When α is greater than 500, performance of CR is same as that of MCNL. The dimension of hyperbuckets is very dependent on α and is enlarged as α increases. The enlarged dimension of hyperbuckets makes the performance of CR to be converged to that of MCNL. Figure 5-(e) shows the dimension of hyperbuckets determined according to the value of α. If α grows over 500, the dimension of hyperbuckets is decided as that of the hypercube system and CR shows the same



(a) Communication cost (n=8) (b) CPU cost (n=8)

(c) Communication cost (α=60) (d) CPU cost (α=60)

(e) Bucket dimension (n=8) (f) Bucket dimension (α=60)

Figure 5. Communication and CPU cost comparisons

characteristics of MCNL. Figure 5-(b) shows the CPU cost characteristics of four join algorithms. The performance characteristics of CPU cost is similar to that of communication cost because the CPU cost is very dependent on the number of received tuples. CNL and MCNL which examines more tuples than CHH and CR show higher CPU cost.

Figure 5-(c) and (d) show the performance characteristics of join algorithms according to the dimension of hypercubes. As the dimension of hypercubes increases the communication cost of MCNL and CNL increases slightly. This is caused by the increased traveling path length as the dimension of hypercubes increases. The communication cost of CHH is decreased as opposed to the MCNL case. The number of tuples transmitted at each step in the bucket-based approach is constant ($\|R\| / (2 \times N)$) as we mentioned in section 3. The increased dimension of hypercubes reduces the number of tuples transmitted at each step exponentially and this cost benefit dominates the communication cost induced by the increased number of step. This effect is also shown in the case of CR. In the moderate range of hypercube dimensions, the behavior of CR is same as that of MCNL

because the dimension of hyperbuckets is determined as same as that of the hypercube system(Figure 5-(f)). When the dimension of hypercubes is higher than 5, the dimension of hyperbuckets stops increasing and becomes constant. By controlling the dimension of hyperbuckets properly, CR achieves the minimized communication cost. This dynamic behavior makes CR to resemble the performance characteristics of CHH and to show better performance than other algorithms.

The increased number of processors affects the CPU cost of MCNL and CNL a little. Although the increased number of nodes reduces the amount of tuples examined at each step of broadcasting, the increased number of nodes requires more broadcasting steps and as result, the CPU performances of MCNL and CNL are not enhanced. However, the CPU performances of CR and CHH are enhanced with the help of the increased number of processors. With the increased number of processors, a join operation is divided into more smaller jobs and a node spends less CPU cost. The reason that CR shows better performance than CHH is that CR spends less CPU cost to process the larger relation *S*. CR experiences less *move* operations than CHH due to the hyperbucket concept.

CHH which divides relations into more buckets must move more tuples to output buffers. However, CR which divides relations into less buckets moves less tuples to output buffers. Instead of moving tuples to output buffers, CR experiences more local join operations. Although CR experiences more false-join than CHH, CR shows better performance because the cost of a *move* operation is generally more expensive than that of a *cmp* operation.

B. Overall performance analysis

To analyze the effect of the size ratio α to the performance of join operations we have experimented with the 4- and 8-dimensional hypercube systems and the results are shown in *Figure 6*-(a) and (b). CR outperforms other join algorithms regardless of α. The performance characteristics of join algorithms are very similar to Figure 5 since the communication cost dominates the overall performances.

Figure 7 shows the contribution of the communication and CPU costs to overall performance of join algorithms. In CNL, the portion of the communication cost is nearly constant. As the dimension of the hypercube increases, the communication portion of CHH increases from 80% to 98%(Figure 7-(b)). Because CHH is based on "divide and

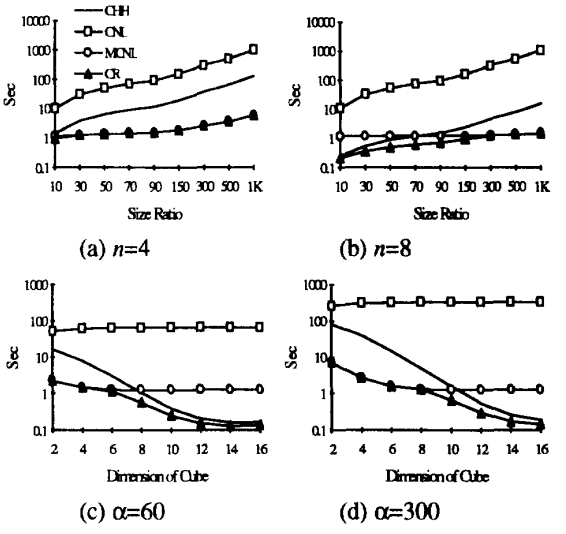
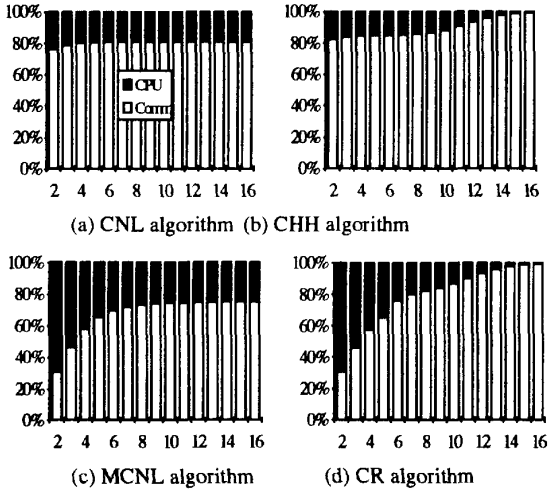


Figure 6. Overall performance comparisons

conquer", the increased dimension of the hypercube generates the larger number of buckets which require less CPU processing. As a result, the CPU cost portion decreases.

Figure 7-(c) shows the dominant factor of MCNL is the CPU cost in the moderate range of hypercube dimensions. In MCNL, the smaller relation *R* contributes to the communication cost and the larger relation *S* to the CPU cost because only the smaller relation *R* is broadcasted over the hypercube. In the moderate range of hypercube dimensions, the CPU cost needed for probing the local hash table dominates the communication cost for distributing the relation R. However as the dimension increases, the portion of the larger relation *S* kept in a node becomes smaller and this results in the low CPU cost. In contrast to the CPU cost, the communication cost increases slightly according to the increase of the hypercube dimension due to the increased traveling path length. Therefore the portion of the CPU cost in the overall performance decreases.

Figure 7-(d) shows the relative portion of the communication and CPU costs of the CR join algorithm. In the lower dimension of the hypercube systems, CR's performance characteristics are very similar to those of MCNL and become similar to those of CHH with higher dimensions.



(a) CNL algorithm (b) CHH algorithm

(c) MCNL algorithm (d) CR algorithm

Figure 7. Relative cost comparisons (α=60)

5. Conclusions

In this paper we have proposed an efficient parallel join algorithm, called Cube-Robust, for hypercube multicomputers. The proposed algorithm is robust due to its immunity from the size ratios of two relations to be joined. The Cube-Robust(CR) join algorithm proposed in this paper combines merits of the bucket-based and broadcast-based approaches by introducing hyperbucket concept. The dimension of hyperbucket is determined dynamically so as to minimize the communication cost. Hyperbucket of k-dimension transforms an n-dimension hypercube system into the $(n-k)$-dimension hypercube system. When the dimension of a hypercube system is large, the CR join algorithm takes advantage of clustering effect and achieves the low communication cost by applying the bucket-based approach to the reduced hypercube system. When the dimension of the hypercube system is small, CR uses large hyperbuckets to maximize benefits from the broadcast-based approach, i.e., reduced communication cost. We have shown the CR join algorithm outperforms methods proposed earlier through an analytic cost model and simulation experiments under various running environments. Since the communication cost in parallel join algorithms dominates the performance of join algorithms when the hypercube dimension is large or the size ratio of relation is high, optimization of the communication cost is necessary to achieve high performance of parallel join algorithms.

References

[1] Bitton, D. et al, Parallel Algorithms for the Execution of Relational Database Operations, ACM Trans. on Database Systems (Sept., 1983) 324-353.

[2] H.I., Choi, et al, An Efficient Parallel Join Algorithm Based on Hypercube-Parititoning, CS-TR-94-85, Dept. of Computer Science, KAIST.

[3] S. M. Chung, JaerheenYang, Distributive Join Algorithm for Cube-Connected Multiprocessors, Proc. Symp. on DASFAA (1993) 253-260.

[4] Dewitt, D., Katz, R., et al, Implementation techniques for Main Memory Database Systems, Proc. ACM SIGMOD (1984) 1-8.

[5] Frieder, O., Baru, K., Database Operations in a Cube-Connected Multicomputer System, IEEE Trans. on Computers Vol.38, No.6 (June, 1989) 920-927.

[6] Frieder, O., Multiprocessor Algorithms for Relational-Database Operators on Hypercube Systems, IEEE Computer Survey & Tutorial Series (Nov., 1990) 13-28.

[7] Harary, F. A Survey of the Theory of Hypercube Graphs, Comput. Math. Appli. Vol. 14, No. 4 (1988) 110-121.

[8] B. L. Menezes, K. Thadani, A. G. Dale, R. Jenevein, Design of a HyperKYKLOS-based Multiprocessor Architecture for High-Performance Join Operations, Proc. 5th International Workshop on Database Machines (Japan 1987) 88-101.

[9] E. R. Omiecinski, E.T. Lin, A Hash-based Join Algorithm for a Cube-Connected Parallel Computer, Information Processing Letters 30 (1989) 269-275.

[10] E. R. Omiecinski, E.T. Lin, Hash-based and Index-based Join Algorithms for Cube and Ring Connected Multicomputers, IEEE Trans. on Know. & Data Eng. Vol.1, No.3 (Sep., 1989) 329-343.

[11] E. R. Omiecinski, E.T. Lin, The Adaptive-Hash Join Algorithm for a Hypercube Multicomputer, IEEE Trans. on Parallel and Distributed Systems, Vol. 3, No. 3 (1992) 334-349.

[12] Schnieder, D., Dewitt, D., A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment, Proc. of SIGMOD Conf. (1989) 110-121.