Practical Hardware for Linear Execution of Relational Database Operations

T.H. Merrett

School of Computer Science

McGill University

Key Words: Database machines, multidimensional paging, natural join, processor arrays, relational algebra, storage structures.

0 Introduction

It may be useful, in doing a natural join of two database relations (or some other operation such as intersection, union or projection) to calculate first a bit array showing which tuples from each relation participate in the result. Table 1 gives an example of a relation of 6 tuples joined with a relation of 5 tuples. Only the join attributes are shown. The values are in no particular order and duplicates appear because the join attribute is not necessarily the key of either relation: the values of other attributes are assumed to be such that each tuple is unique. The table entry is 1 wherever a match occurs and 0 elsewhere.

		relation R :	5	2	2	3	1	6
		7	0	0	0	0	0	0
		3	0	0	0	1	0	0
relation	S	2	0	1	1	0	0	0
		1	0	0	0	0	1	0
		2	0	1	1	0	0	0

Table 1. Attributes participating in join of relations R,S and their matches.

Such a bit array would be useful if it were sparse. The join could be computed by restricting the operands using the bit array and doing the appropriate combination of tuples. If, on the other hand, the result of the join is large, the computation required to access and combine the tuples would probably exceed that needed for a simple sort and merge. We will follow reference[1] in not considering this problem.

Although the techniques we discuss in the sequel can easily be used for operations other than join and for operations involving more than one attribute (see reference[1]), we will examine only single-attribute equi-joins.

1 VLSI Systolic Arrays for Equi-Join

Kung and Lohman[1] propose an array arrangement of very simple processors to perform joins and other operations of the relational algebra. Each processor is a bit-comparator and they suggest that about 1000 can fit on a VLSI chip. Chips can be combined to operate at the byte or even attribute level. We will suppose that an elementary processor can compare two values, one from each of two union-compatible attributes. The comparison array for single-attribute joins is shown in Fig. 1. Values from the join attribute of relation R are pumped through the array from left to right at intervals of two time units. Values of S are similarly pumped from right to left. When two values appear in the same processor, they are compared for equality and the result, true or false, is pumped out the bottom. This result gives the bit array of Table 1, as Fig. 2 shows.

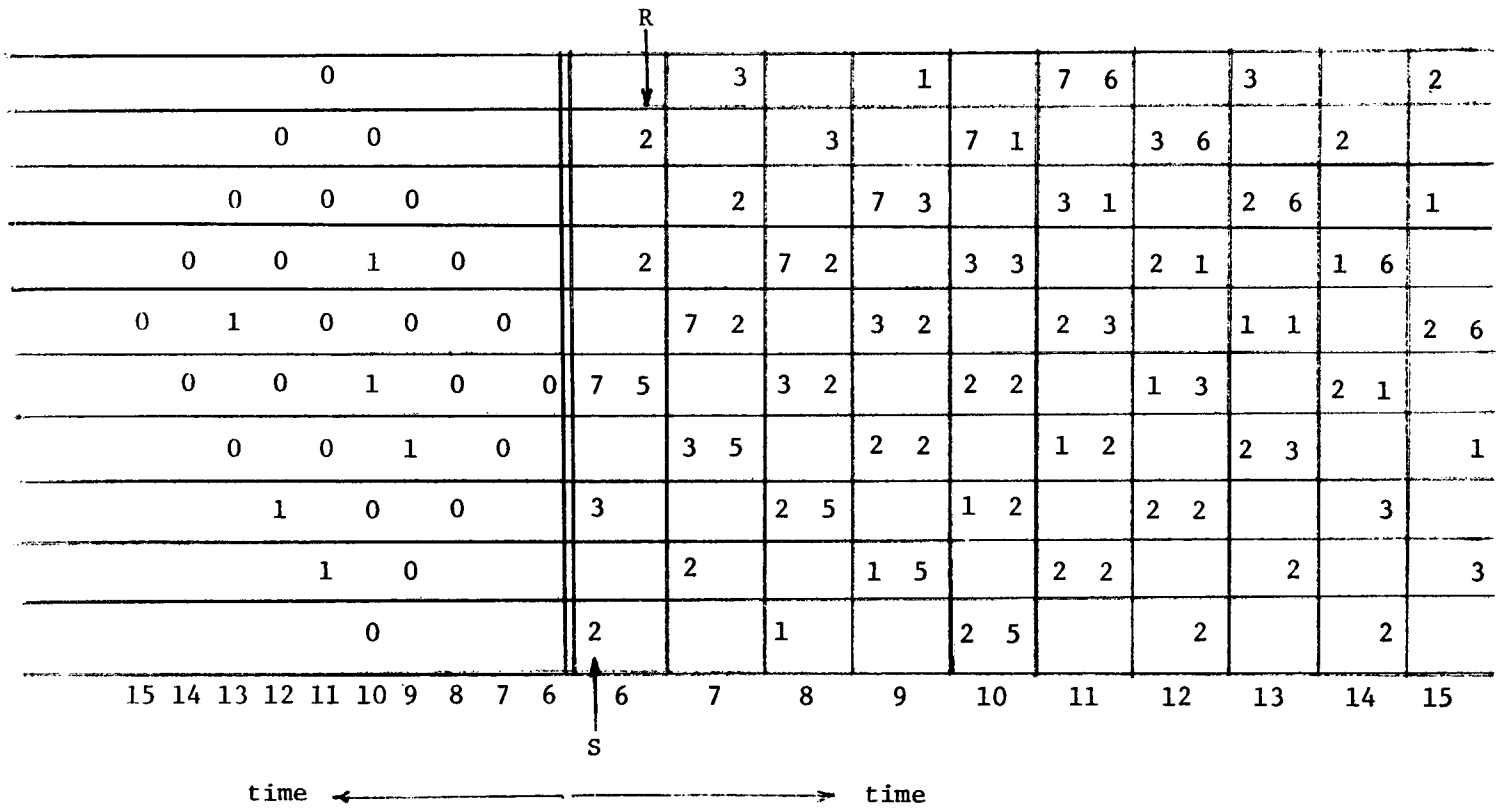

Fig. 2. Ten processors compute bit array for R*S in ten time units

Fig. 2 shows ten processors, each one represented by a row, producing the bit array of Fig. 1 in ten time units (five additional time units are needed to load the processors before the first comparison is made). It is apparent from the example that $|R| + |S| - 1$ processors and the same number of time units are needed in general. If fewer processors are available, the join can not be done in this linear time: one relation would have to be partitioned and the other passed repeatedly through the processor array.

Despite the assurances of reference[1], it is unreasonable to assume that chip integration will keep pace with relation sizes. To require circuitry with as many components as there are tuples in the pair of operand relations is effectively a violation of the plausible assumption that primary memory capacity will never be enough: secondary storage is always needed.

2 Multipaging

In reference[2] we proposed a storage organization for relations that has no index overhead but speeds access for queries and processing. This is an order-preserving, multidimensional hashing technique that trades overflows against waste of storage. The idea is simply to partition each attribute that is of interest and to make the page boundaries correspond to the partition. This gives a rectilinear segmentation of the data space which corresponds to the values of the relation attributes. The pages are then addressed just as we would address elements of a multidimensional array.

In one dimension, which is all we consider here, this amounts to a linear segmentation of the values of the join attribute of each relation. Table 2 shows the result of splitting R and S into segments each, between values 2 and 3 of the join attribute. Note that the split is between values 2 and 3 in both R and S : this is an important (but not indispensible) part of our approach.

R :	2 , 2 , 1	5 , 3 , 6
S :	2 , 1 , 2	7 , 3

Table 2. Partition of R and S between values 2 , 3 .

Note that within the partitions, the values still appear in any order. What is important is that no value exceeding 2 appears in the first partition and no value less than 3 in the second.

Now let us compute the bit array of the join, this time with only five processors (fig. 3).

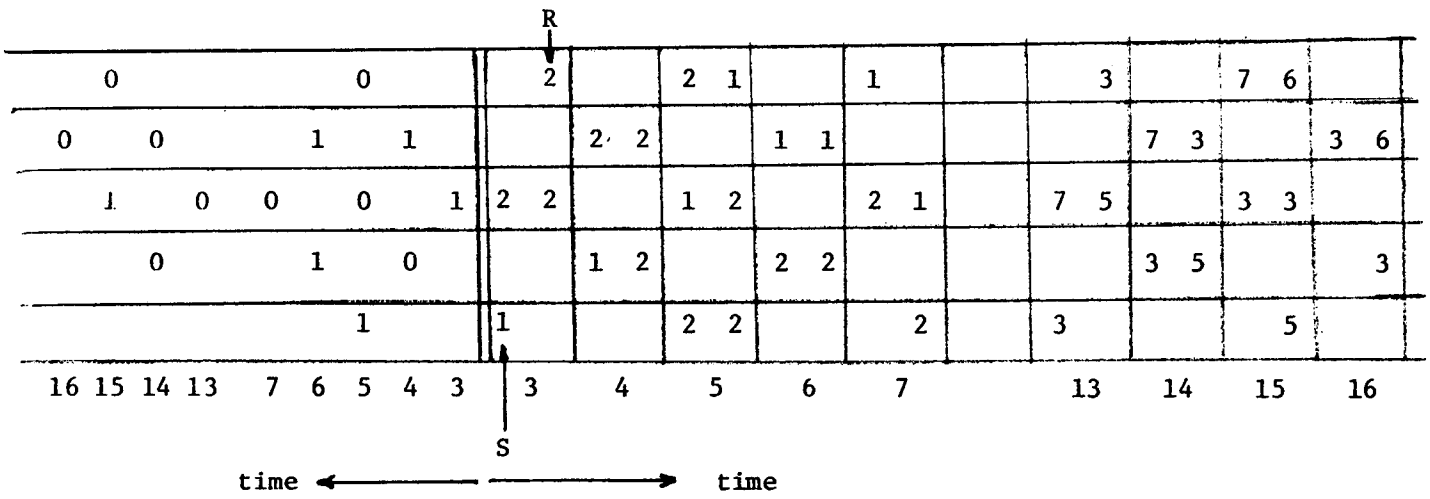

Fig. 3. Five processors compute bit array for R*S in nine time units.

The effect of the partitioning has been to *block-diagonalize* the bit array, with a consequent reduction in cross-section of the non-zero blocks of the bit array and therefore of the number of processors required. The example makes it clear that $\max_i (|R_i| + |S_i| - 1)$ processors are needed, where i runs over all b blocks on the diagonal. The time required, apart from loading and clearing the processor array for each block, is $\sum_i (|R_i| + |S_i| - 1) = |R| + |S| - b$.

When we are dealing with multi-attribute relations, the number of tuples in the partition is not the number of tuples per page in general, since there may be many pages corresponding to one segment of any attribute (see [2]). However, it is a fraction the number of tuples in the relation. Thus the number of processors in the systolic array, for one-pass joins, is much greater than a typical buffer size but still much smaller than the size proposed by Kung and Lohman[1].

3 Summary

We have applied the file-organizing technique of multipaging[2] to the proposal for a systolic array processor[1] for the relational algebra and have reduced the number of processors required from $|R| + |S| - 1$ to $\max_i (|R_i| + |S_i| - 1)$ where i runs over the multipage segments of the relation.

References

1. H.T. Kung, Philip L. Lohman, 1980: Systolic (VLSI) Arrays for Relational Database Operations, Proc. ACM-SIGMOD 1980, pp. 105-116. May 14-16, Santa Monica, Calif.

2. T.H. Merrett, 1978: Multidimensional Paging for Efficient Database Querying, ICMOD 78 Conference Proceedings, pp. 277-290. International Conference on Database Management Systems, June 29-30, FAST, Milano, Italy.

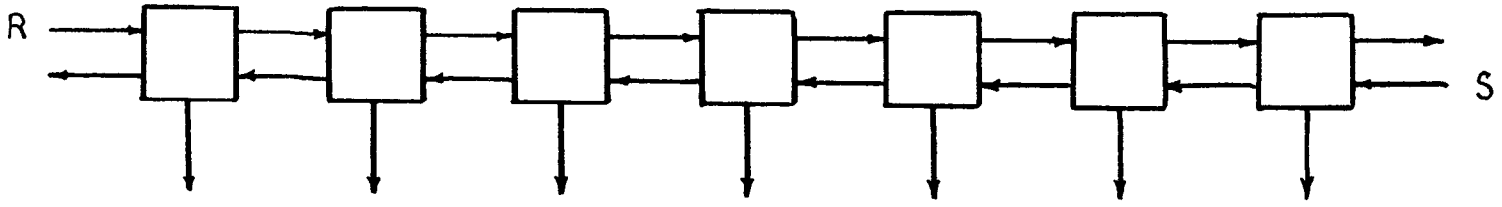

Fig. 1. Systolic array for single-attribute join of R and S .