95-84

RELATIONAL QUERY PROCESSING
ON THE NON-VON SUPERCOMPUTER¹

David Elliot Shaw

Department of Computer Science
Columbia University

Abstract

The central focus of this chapter is the highly efficient execution of relational database queries using a particular nonstandard machine called NON-VON. NON-VON is a massively parallel, non-von Neumann "supercomputer", portions of which are now being implemented in the Computer Science Department at Columbia University. The machine is intended to support the rapid execution of many large scale data manipulation tasks, including relational database operations and a number of other functions relevant to commercial data processing.

The NON-VON architecture includes a tree-structured primary processing subsystem, which we are implementing using custom nMOS VLSI circuits, along with a secondary processing subsystem based on a bank of intelligent disk drives. A high-bandwidth parallel interface provides for rapid data transfer between the two subsystems. This chapter briefly describes the organization

¹This research was supported in part by the Defense Advanced Research Projects Agency under contract N00039-80-G-0132.

of the NON-VON machine, and considers in somewhat greater detail the manner in which it is used for the high-speed processing of relational database queries.

1 Introduction

Simply stated, the goal of a computer systems engineer is to construct a system that

1. Does what the user wants done.
2. Does it as quickly as possible.

The concurrent realization of these two goals, however, is often impeded by a peculiar sort of "chicken-and-egg" problem. On the one hand, it is difficult to decide which operations should be made highly efficient without identifying those operations which are most frequently executed by a real body of users. Real users, on the other hand, avoid performing operations that require a great deal of time; if these features are essential to the system, they often avoid using the system entirely. The systems engineer is thus left with a distorted view of the intrinsic preferences of his or her users. Moreover, this distortion systematically favors the status quo, since the engineer is unable to identify currently unpopular features that would be used, if only they were more efficiently implemented.

It is our feeling that the present complexion of the market for database management software and hardware may reflect just such a circular distortion of user preferences. In particular, we believe that the many advantages of the relational model of data [CODD71], now widely accepted by most computer scientists, would by now have led to a nearly uniform adoption of relational database systems in industry were it not for serious problems of efficiency. Conversely, we believe that machines capable of supporting the highly efficient execution of relational database operations might well be

commercially available today if a sufficiently large market for such hardware had been identified a decade or so ago.

Within this framework, the efforts of researchers interested in the implementation of database machines [OZKA75, SU75, McGR76, HSIA77, DeWI78, SHAW79, BABB79, KIM80, SONG81] capable of supporting the relational model of data may be viewed as an "act of faith". Because we believe that relational database systems, and in particular, their most time-consuming operations, would be employed by a great many users if only they could be made much more efficient, we have adopted as one of our central goals the efficient, cost-effective support of the these operations on NON-VON.

In the first section of this chapter, we briefly sketch the architecture of the NON-VON machine. Section 2 describes some of the fundamental operations employed typical NON-VON algorithms, while Section 3 outlines the basis for NON-VON's use as a relational database machine. The fourth section details the manner in which small relations are manipulated; the processing of large relations is described in Section 5. We conclude with a summary of NON-VON's potential utility and a word of caution regarding the formulation of premature conclusions.

2 The NON-VON Architecture

This section outlines the bare essentials of the latest version of the NON-VON family of supercomputers, NON-VON 4. While it is hoped that this description will prove sufficient for an understanding of the algorithms and analysis presented later in the chapter, readers unfamiliar with the NON-VON architecture may wish to review the details of an earlier version of the machine, which have been presented elsewhere [Shaw, 1982].

The top-level organization of the NON-VON machine is illustrated in Figure 1.

Figure 1: Organization of the NON-VON Machine

NON-VON has two principal components, known as the primary processing subsystem and the secondary processing subsystem. In a typical configuration, the machine would be connected to a host machine, a general purpose computer serving as a front end device for interactions with the user.

The primary processing subsystem is organized as a binary tree consisting of a large number of small processing elements (SPE's). Using a (currently feasible) nMOS process with 2 micron feature size, 16 SPE's can be implemented on a single VLSI chip. Each SPE contains a simple eight-bit ALU, a 64-bit RAM, and communication connections to three neighboring SPE's, which are known as the parent, left child, and right child. In addition, each SPE is capable of communicating, within a single instruction cycle, with two additional SPE's, called the left neighbor and right neighbor. These neighbors are the predecessor and successor in an inorder traversal of the primary processing subsystem tree. Among the uses of NON-VON's physical (tree) and logical (linear) neighbors is the support of records whose length exceeds the capacity of a single local RAM.

The SPE's do not store programs locally. Rather, they receive instructions that are broadcast to them from higher up in the primary processing subsystem tree and function in a strictly synchronous manner. Within the top five to ten (depending on the configuration) levels of the primary processing subsystem, each SPE is connected to a large processing element (LPE). Each LPE is a general-purpose microcomputer with a larger RAM, and is thus capable of storing programs locally.

The LPE's may execute programs independently and asynchronously. In particular, LPE's at the roots of several subtrees of the primary processing subsystem (possibly at different levels) may broadcast separate instruction streams to the SPE's below them, giving NON-VON the capability for what is sometimes called multiple-SIMD execution. Each LPE includes a small amount of

specialized hardware to perform instruction broadcast and to generate control signals for the SPE's. The LPE from which a given SPE is currently receiving its instructions is sometimes called its control processor.

The LPE's are connected by a high-bandwidth interconnection network. The precise type of network has not yet been determined; among the candidates are several kinds of logarithmic-stage networks of the butterfly/omega/banyan family and certain configurations based on crossbar switches. While the detailed architecture of the LPE network does not comprise a central part of our research, the use of such a high-bandwidth network is essential to a number of high-performance NON-VON algorithms involving large collections of data, including some reported in this chapter.

The secondary processing subsystem incorporates a substantial number (perhaps between 16 and 256) of disk drives, each of moderate size. Each drive is connected via an intelligent head unit to an LPE in the primary processing subsystem, providing a very high bandwidth interconnection between these two subsystems. In addition to the reading and writing of data from disks, intelligent head units perform certain computationally simple operations "on the fly", passing results to the associated LPE's. By way of illustration, a partial match operation (equivalent to the relational algebraic operator select) may be executed, passing on to the primary processing subsystem only those records that satisfy certain attribute/value criteria.

We are now constructing a prototype primary processing subsystem containing a single LPE and 16,383 SPE's. This machine will be connected to a VAX 11/750, which will serve as host. After the completion of this single-LPE prototype, we hope to construct a machine having multiple LPE's and a larger primary processing subsystem.

satisfied immediately following the execution of a RESOLVE instruction.) REPORT causes the contents of a particular register in the single enabled SPE to be transferred to its control processor. The REPORT and RESOLVE operations are used in several of the database algorithms described below.

4 NON-VON as a Database Machine

The utility of the NON-VON supercomputer in database management applications stems from its highly efficient execution of the operators of a relational algebra [CODD71]. Specifically, NON-VON supports highly efficient parallel algorithms for the relational algebraic operators

- Selection
- Projection
- Join
- Union
- Intersection
- Set Difference

The machine also supports the highly efficient execution of summation, aggregation, and various statistical operations, all of which find use in numerous database applications. In this chapter, however, we will restrict our attention to the relational algebraic operators.

In the sections that follow, we will outline the algorithms NON-VON uses to evaluate each of the relational algebraic primitives, both in the case where the arguments can fit entirely within primary storage (the case we call internal evaluation), and where they reside on secondary storage (external evaluation). Space does not permit a detailed explication of all of the internal evaluation of all of the relational algebraic operators enumerated

3 Fundamental Operations

In order to understand the essential operations employed in NON-VON's algorithms for database operations, it would be useful to briefly review certain aspects of the NON-VON SPE and its instruction set. Of central importance is a mechanism that allows an LPE to selectively enable certain of its SPE's. Each SPE contains a one-bit flag called the enable bit. When the enable bit is set to 1, the SPE is said to be enabled; in this state, it responds to any instruction broadcast by its control processor. When the enable bit is 0, the SPE is disabled, and will ignore any such instructions.

The selective enabling of various SPE's is essential to a number of fundamental NON-VON operations. One of these operations involves the parallel comparison of strings stored in a number of SPE's against a string broadcast by their common control processor. All SPE's in which the match fails are disabled, while all matching SPE's remain enabled. Using a single machine language instruction that compares one byte, increments the SPE's memory address register, and disables the SPE in the event of a match failure, NON-VON is able to perform such comparisons at a rate of one byte per instruction cycle (about 400 nanoseconds). This operation is used in all of the relational operations described in the remainder of this chapter.

Another operation used in some of the algorithms described below is the rapid identification of a single SPE from among a set of SPE's in which a given match has succeeded. NON-VON's RESOLVE instruction turns off a particular one-bit flag in all SPE's except the one occurring first in an inorder traversal of the tree. Using this instruction, the members of a set of "marked" SPE's may be sequentially enumerated.

The REPORT instruction is meaningful only when exactly one SPE in the subtree controlled by a given LPE is enabled. (Note that this condition is always

above, and of the time complexity of each one. Our treatment will thus be abbreviated, and somewhat informal. Readers interested in further details may wish to examine material published elsewhere [Shaw, 1980; Hillyer, Shaw, and Nigam, 1983].

5 Internal Evaluation of the Relational Algebraic Operators

In the discussion that follows, we will assume that each SPE contains a single tuple of an argument relation. In fact, NON-VON supports both packed records, where several short tuples are stored in a single SPE, and spanned records, which are too large for a single SPE, and must be split among two or more [Shaw and Hillyer, 1982]. These techniques are, however, orthogonal to and beyond the scope of the current discussion.

Of the six relational algebraic operators listed above, the simplest to implement on NON-VON is relational selection. To select those tuples of a relation that satisfy some attribute/value criterion, NON-VON simply compares the required values of each attribute simultaneously against the appropriate field in each SPE in the primary processing subsystem, disabling all SPE's that do not match. One instruction cycle is required for each byte in the specified value string. When performing such an operation, the NON-VON primary processing subsystem functions as a simple content addressable memory. At the end of the select operation, only those tuples that satisfy the given attribute/value specification remain enabled. These tuples may be either enumerated sequentially using the RESOLVE and REPORT instructions or used as the arguments for other relational algebraic operations, depending on the problem at hand.

The execution of projection operations on NON-VON is more interesting. The removal of selected fields from each tuple is straightforward even on a von

Neumann machine. The difficult aspect of projection arises from the fact that the deletion of these attribute values may make two previously distinct tuples identical. Since relations are, by definition, sets, all duplicate tuples must be removed from the result relation in a true projection.

The project algorithm begins by issuing a RESOLVE against all tuples in the relation, thus marking an arbitrary tuple as the "current tuple". Using a sequence of REPORTS, the projected values from the current tuple are then sent to the control processor. The projected tuple is included in the result relation. In order to remove any duplicate tuples before they are enumerated, these values are also broadcast to all remaining tuples in the relation, and all matching tuples are marked as "excluded", as is the current tuple itself. These steps are then repeated for all tuples not yet excluded, until all tuples have been excluded.

It should be noted that this algorithm is actually sublinear (under the assumption that all input data is already present in the primary processing subsystem). This follows from the fact that the time required for projection is proportional not to the size of the input relation, but to the size of the result relation, since duplicate tuples are eliminated in parallel before they have a chance to initiate an execution of the program loop.

In practice, the join algorithm is typically the most expensive of the relational algebraic primitives. The NON-VON join algorithm, however, is both efficient and simple, corresponding closely to the "naive" sequential algorithm, but with its inner loop replaced by a single associative operation. The algorithm enumerates each tuple of the first relation in turn using the RESOLVE and REPORT instructions. Only the values of the join attributes of the current tuple are reported to the control processor. These values are then broadcast to all tuples in the second relation. All matching tuples concurrently mark themselves, and are read out in turn and concatenated with

the current tuple from the first relation to form a partial result. The process is then repeated for each tuple in the first relation.

Note that the total running time of this algorithm is linear in the size of the smaller argument relation (which may be chosen to be the "first relation") and the result relation. It should be noted, however, that the result relation may, in the worst case, be quadratic in the size of the argument relations; this limits the worst case running time of a join on any machine, whether sequential or parallel, that must enumerate its output sequentially. Fortunately, join operations in most applications of practical interest tend to produce result relations of the same order of magnitude as the argument relations. Providing this constraint is satisfied, the NON-VON join algorithm has time complexity linear in the size of the argument relations.

The NON-VON algorithms for the three set theoretic operations are somewhat simpler. In each case, each tuple from one of the argument relations is broadcast in turn, again using RESOLVE and REPORT instructions, for simultaneous comparison against all the others. The three algorithms differ only in the choice of tuples to be included in the output.

The set union algorithm enumerates the tuples in the first relation and compares each such tuple in parallel against all tuples in the second relation. All matching tuples are simultaneously marked as excluded, thus preventing the appearance of duplicate tuples in the result. After the last tuple in the first relation has been processed, the set of non-excluded tuples from the first and second tuples constitutes the result relation. The algorithm for intersection is identical to that for union, except that the result is the set of tuples that match, rather than all tuples except those that match. In the set difference algorithm, all tuples from the first relation that do not match against the second relation are included in the result relation. In all cases, the running time is proportional to the number

of tuples in the first relation. In the case of union and intersection, the first relation may be chosen to be the smaller of the two arguments.

It may prove instructive to compare the asymptotic running times of these NON-VON algorithms with the best known algorithms for a conventional computer system. With the exception of the selection operator, all of these operators are (in the absence of either special constraints or index mechanisms having very high storage and update costs) typically accomplished on a von Neumann machine by first sorting the argument relation(s). In the case of the three set theoretic operators, the entire tuple is used as a key. Projection is accomplished by pre-sorting on the projected attributes, while the join attributes serve as keys for sorting the arguments to the join operator. The sorting process moves identical key values to adjacent locations, where they may be easily processed in linear time. The sorting step itself, which requires $O(n \log n)$ time, thus dominates the complexity of each of the sequential algorithms.

On NON-VON, on the other hand, there is no need to pre-sort the relations, and each of the relational algebraic primitives requires only linear time (again with the exception of selection, which is faster). Intuitively, this follows from the fact that NON-VON is able to make equality (and other) comparisons against an arbitrary number of operands in constant time, independent of the size of the argument relations, thus obviating the need to sort.

Selection is a special case. On a von Neumann machine, the naive algorithm for selection requires linear time. Simple hashing can not reduce this time to constant unless 2^k hash tables are constructed and (at great expense) maintained, where k is the number of attributes. More sophisticated techniques have reduced the time required to between $O(1)$ and $O(n)$, depending on the number of specified attribute/value pairs, but at the expense of extra storage and extra processing at the time of tuple insertion. NON-

VON, on the other hand, is able to perform general relational selection in constant time, regardless of the number of attribute/value pairs, with no need for either auxiliary data structures or additional insertion time.

6 External Evaluation of the Relational Algebraic Operators

In most database applications, the argument relations are too large to fit entirely within the primary processing subsystem. In the case of relational selection, this presents no problem, since, as noted above, the intelligent head units associated with each disk head are themselves capable of performing relational selection dynamically on the tuples passing beneath them. The other relational operators, however, involve "global" comparisons that can not be performed by a single intelligent head unit having little storage and processing capacity. In particular, the disposition of a given tuple may be affected by a tuple in another part of the file, which may well pass under a different intelligent head unit at different point in time.

NON-VON attacks such problems by decomposing the argument relations into a number of partitions, each small enough to be processed within the primary processing subsystem, and constructed in such a way as to guarantee that no reference need be made to some other partition. By way of illustration, let us consider the case of external projection. Here, we wish to partition the (single) argument relation in a way that guarantees that any identical tuples that may be present after projecting out the specified attributes will wind up in the same partition. If this can be guaranteed, it will be possible to transfer each partition into the primary processing system in turn, so that all duplicate tuples in that partition can be detected and eliminated.

This process, which we have termed key-disjoint partitioning [Shaw, 1979] is accomplished by hashing the key (the projected tuple, the compound join

attribute, or, in the case of the set theoretic operators, the whole tuple) onto a real number in the range $[0, 1]$. The interval is divided into a number of partitions somewhat larger than the size of the argument relation(s) divided by the capacity of the primary processing system. All tuples (of both relations, in the case of the binary operators) falling within a given partition are processed simultaneously in the primary processing subsystem.

The manner in which this key-disjoint partitioning is accomplished is as follows. Recall that tuples of the argument relation(s) are distributed among the various disks in the secondary processing subsystem. The relation is scanned by all disk heads in parallel, and the intelligent head units hash the key of each tuple as it passes under the corresponding head. Each tuple is then sent through the LPE network to an LPE determined by the hashed key value, and is transferred from that LPE to the corresponding disk. At the end of this process (which requires time proportional to the size of the relation in bytes divided by the number of disk heads in the secondary processing subsystem), the file has been segmented into key-disjoint partitions.

Because a full-scale NON-VON configuration would incorporate a relatively large number of moderate-sized disk drives, most files would be divided into enough partitions that each one would be small enough to fit entirely within the PPS. In this case, each partition would be transferred (serially, in this case, since each partition would reside on a different disk drive) into the primary processing subsystem for internal evaluation.

In the case of extremely large files, however, a single partition might exceed the capacity of the primary processing subsystem. In this case, the partitioning procedure would be applied to each partition, thus dividing the size of each partition again by a factor roughly equal to the number of disk heads. The resulting sub-partitions would then be processed internally in the same manner. In its most general form, the algorithm provides for an

arbitrary number of levels of sub-partitioning; in practice, however, the large "fanout" at every stage would make the need for more than two or three levels quite rare. Formally, the algorithm has complexity $O(n \log n)$, assuming the argument relations can truly be of arbitrary size; the constant factors, however, are such that the effect of the $(\log n)$ term should in practice be of limited significance.

Consider the case in which the the argument relation(s) are no larger than the product of the capacity of the primary processing subsystem and the number of disk drives. The single hash partitioning step contributes little to the complexity of the algorithm. Specifically, this step requires time proportional to the size of the argument relation(s) divided by the number of disk drives (since both hashing and routing through the LPE network occur in parallel over all drives). The remaining steps dominate the running time of the algorithm; they require time proportional to the length of the argument relation(s), since each partition must be read into the primary processing subsystem in turn and subjected to internal evaluation. Since internal evaluation is quite rapid, the total running time of the algorithm should typically be a small constant multiple of the amount of time that would be required simply to read the argument relation(s) from disk in a conventional system.

When the argument relations are very large, it has been noted that the partitions must themselves be divided into sub-partitions. This process is somewhat more expensive than in the case we have just analyzed, since the original data is distributed evenly among the various drives, while the sub-partitioning process operates on data stored on a single drive. The effect of this difference, however, is only to make the running time of the partitioning step comparable to that of the remaining steps. Since no more than two, or for extremely large files, three recursive subpartitionings are likely to be

required in practice, the total running time of these algorithms should in practice be a fairly small multiple of the length of time that would be required to input the arguments.

7 Conclusion

Based on our work to date, we believe that the NON-VON supercomputer could provide significant performance improvements over conventional machines of comparable cost in the execution of the essential operations involved in relational database management. Central to NON-VON's predicted performance advantages in such applications is the use of associative processing techniques supported by NON-VON's VLSI-based primary processing subsystem and the utilization of "intelligent disks" to partition the input data using a technique based on hash coding. Further analytical studies, experimental development, and performance evaluation will be necessary, however, before the machine's utility in database management applications can be assessed with confidence.

References

DeWitt, David J., "Direct -- A Multiprocessor Organization for Supporting Relational Database Management Systems", in Proc. 5th Annual Symposium on Computer Architecture, 1978.

Hillyer, Bruce K., David Elliot Shaw, and Anil Nigam, "NON-VON's Performance On Certain Database Benchmarks", Technical Report, Columbia Computer Science Department, 1983.

Hsiao, David K., K. Kannan, and D. S. Kerr, "Structure Memory Designs for a Database Computer", in Proc. ACM Annual Conf., 1977.

Kim, Won, Query Optimization for Relational Database Systems, Ph.D. Thesis, Department of Computer Science, University of Illinois, August, 1980.

McGregor, D. R., R. G. Thomson, and W. N. Dawson, "High performance for database systems", in Systems for Large Databases, North-Holland, Amsterdam, 1976.

Ozkarahan, Esen A., S. A. Schuster, and K. C. Smith, "RAP: An Associative Processor for Database Management", in Proc. 1975 AFIPS National Computer Conf., vol. 44, AFIPS Press.

Shaw, David Elliot, "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives", Stanford Computer Science Department Report STAN-CS-79-778, October 1979.

Shaw, David Elliot, Knowledge-Based Retrieval on a Relational Database Machine, Ph.D. Thesis, Department of Computer Science, Stanford University, 1980.

Shaw, David Elliot, "The NON-VON Supercomputer", Technical Report, Columbia Computer Science Department, August, 1982.

Shaw, David Elliot and Bruce K. Hillyer, "Allocation and Manipulation of Records in the NON-VON Supercomputer", Technical Report, Columbia Computer Science Department, 1982.

Song, S. W., "On a high-performance VLSI solution to database problems", Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, August, 1981.

Su, S., and G. Lipovski, "CASSM: A Cellular System for Very Large Databases", in Proc. Conf. Very Large Databases, 1975.