CUCS-22-81

The NON-VON Database Machine: An Overview

David Elliot Shaw
Salvatore J. Stolfo
Hussein Ibrahim
Bruce Hillyer

Department of Computer Science
Columbia University

Gio Wiederhold
J. A. Andrews

Department of Computer Science
Stanford University

September, 1981

Table of Contents

Abstract

The NON-VON machine (portions of which are presently under construction in the Department of Computer Science at Columbia, in cooperation with the Knowledge Base Management Systems Project at Stanford) was designed to apply computational parallelism on a rather massive scale to a large share of the information processing functions now performed by digital computers.

The NON-VON architecture comprises a tree-structured Primary Processing Subsystem (PPS), which we are implementing using custom nMOS VLSI chips, and a Secondary Processing Subsystem (SPS) incorporating modified, highly intelligent disk drives. NON-VON should permit particularly dramatic performance improvements in very large scale data manipulation tasks, including relational database operations and external sorting. This paper briefly describes the structure and function of the NON-VON machine and its constituent processing subsystems.

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

1 Introduction

The past decade has seen great progress in the development of powerful tools for the management of large and complex databases. Most evident, perhaps, is the tremendous influence of the relational model of data [Codd, 1971], whose great promise lies in its potential for insulating those who must administer and use the database from the peculiarities of its physical structure, focusing their attention instead on the underlying logical constructs defined by the problems at hand.

With this framework has come a number of high-level linguistic tools for the definition, organization, manipulation and retrieval of data. A common theme evident in the design of many such languages has been the incorporation of mechanisms supporting the nonprocedural specification of data manipulation and query tasks. In particular, a wide range of languages based on the relational and predicate calculi [Codd, 1972; Gallaire and Minker, 1978; Shaw, 1980a] have been introduced that allow the user to indicate what database operations are desired while minimizing the requirement for an explicit specification of how these operations are to be performed. Applying techniques borrowed from the field of Artificial Intelligence, some researchers (for example, Kaplan [1979]; Shaw [1980a]; Wiederhold, Kaplan and Sagalowicz [1981]) have begun to construct systems capable of communicating in an even more "human-like" manner, often making reference to domain-specific entities and relationships in the "real world".

In view of the rapid progress during the past decade in the development of powerful high-level tools for managing large, complex databases, the extent to which these tools have been applied in practice is quite disappointing. While a certain period of time must of course be allowed for the transfer of any technological advance from the research laboratory to its intended sites of application in the public or private sectors, there is reason to believe that the diffusion of contemporary database management technology has been severely retarded for more particular reasons. Specifically, the very limited actual penetration to date of relational database systems into the industrial, commercial and military data processing arenas seems to be attributable in large part to the rather serious time inefficiencies which characterize the performance of most currently operational systems.

Recently, a great deal of commonality has become apparent among the fundamental operations involved in a surprisingly large number of superficially disparate computational approaches to high level database management. Although these operations have been formulated in different ways by different researchers, their essential characteristics are captured by the primitive operators of the relational algebra defined by Codd [1972]. Among these operators are the set theoretic operations union, intersection, and set difference, the relational operators equi-join and projection, and several other operations derivable from these five.

Although the best sequential algorithms known for these operations are still quite inefficient on a von Neumann machine, particularly in the case of very

large databases, we believe it possible to implement alternative machine architectures supporting the highly efficient, but cost-effective, parallel execution of each of these relational algebraic operations, along with a number of other operations of practical importance, including large-scale external sorting. It is this belief which motivated the design of the NON-VON database machine.

This paper examines the organization and behavior of the NON-VON machine, illustrating the essential mechanisms involved in its operation in database management applications. Following a brief overview of the architecture of the full NON-VON machine in Section 2, we will focus on the organization and operation of the Primary Processing Subsystem (in Sections 3 and 4, respectively). By way of background, Section 5 reviews the architecture of a highly parallel architecture for retrieving variable-length strings which was proposed some time ago by another researcher. NON-VON's own PPS unit may be usefully thought of as incorporating elements of this earlier architecture, adapted for realization in VLSI, as described in Section 6.

Having already introduced the central mechanisms involved in associative matching, we describe in Section 7 NON-VON's hardware implementation of associative enumeration and multiple match resolution, which permit each element in a set of matching tuples to be output in turn without the need for intervening associative search steps. The manner in which these mechanisms are applied to the internal evaluation of the relational algebraic operators is described in Section 8. Sections 9 and 10 sketch the organization of the Secondary Processing Subsystem and the most important aspects of its use in the external evaluation of relational database operations.

2 Overview of the NON-VON Architecture

The theoretical basis for the NON-VON architecture was established in the course of a doctoral research project at Stanford [Shaw, 1979], and was accompanied by a mathematical analysis of the attainable time complexity of the equi-join and projection operators on such a machine. The architecture was shown to permit a rather surprising o(log n) increase in efficiency over the best evaluation methods known for a conventional computer system, without the use of redundant storage, and using currently available and potentially competitive technology. In many cases of practical import, the proposed architecture was also found to permit a significant improvement (by a factor roughly proportional to the capacity of the Primary Processing Subsystem) over the performance of previously implemented or proposed database machine architectures based on associative secondary storage devices.

Subsequently [Shaw, 1980a], algorithms for evaluating the selection, restriction, union, intersection and set difference operators (each with comparable or more favorable performance improvements) were also described, and the key procedure on which the architecture is based was contrasted with a related, but in this application, inferior method based on an associative

sorting technique described earlier in the literature. More recently, we have been studying several highly efficient, linear expected time algorithms for external sorting on the NON-VON machine.

NON-VON comprises a Secondary Processing Subsystem (SPS), based on a bank of "intelligent" rotating storage devices and designed to provide very high access and processing bandwidth, and a smaller, but faster Primary Processing Subsystem (PPS), again utilizing a high degree of parallelism, in which the relational algebraic operators may be very quickly evaluated. Transfer between the two devices is based on a process of hash partitioning, which is performed entirely in hardware by logic associated with the individual disk heads, and which divides the argument relations into key-disjoint buckets suitable for "internal" evaluation. The top-level organization of the NON-VON machine is illustrated in Figure 2.1.

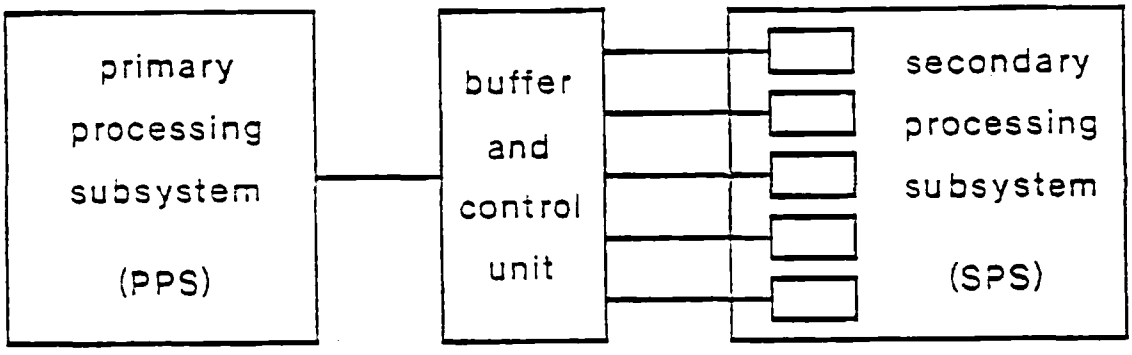

Figure 2.1 Organization of the NON-VON Machine

3 Organization of the Primary Processing Subsystem

The PPS unit functions as the site of what we call internal evaluation of the relational algebraic and other operations performed by NON-VON. Borrowing from the terminology of sorting, we use the term "internal" to distinguish that case in which the operand data is small enough (or can be broken into small enough pieces) to fit entirely within the primary storage device — in our case, the intelligent PPS unit; "external" evaluation refers to the case where the data exceeds the capacity of the PPS, and must be selectively

partitioned and transferred from SPS to PPS.

For purposes of this discussion, the PPS may be thought of as composed of a large number of very simple processing elements (on the order of several thousand, if a full-scale prototype were to be built using 1981 technology, and between a hundred thousand and a million during that period during which NON-VON-like machines would in fact be targeted for practical use), interconnected to form a complete binary tree. With the exception of minor differences in the "leaf nodes", each PE is laid out identically, and comprises:

1. a single common data bus,
2. a very simple (and area-efficient) one-bit-wide ALU for local flag manipulation,
3. an intelligent memory/comparator unit containing 32 bytes of local random-access storage and capable of arithmetic comparisons between values taken from the bus and from specified memory locations

The top-level structure of a single PE is illustrated in Figure 3.1.

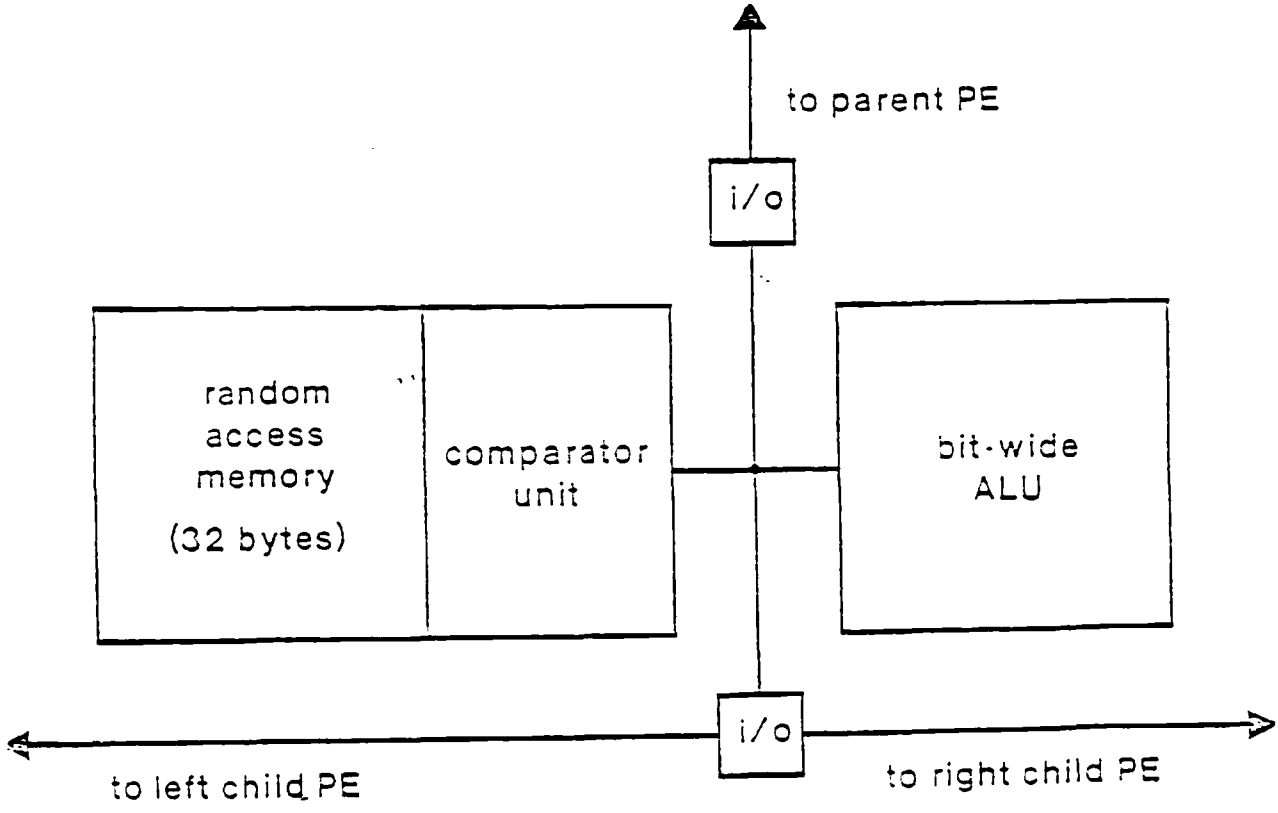

Figure 3.1 Components of a Single Processing Element

By contrast with a conventional microprocessor, no finite-state control logic is incorporated within the constituent PE's. Instead, a single programmable logic array (PLA) associated with each chip services all PE's on that chip, as

described below.

The PPS will be implemented largely using two custom-designed VLSI chips, which we call the PPS Bottom Chip and PPS Middle Chip. Bottom Chips will each contain a subtree of the full PPS tree, and will thus embody 2^k-1 constituent PE's for some k depending on device dimensions. Rough preliminary estimates based on 2.5 micron design rules suggest that a value of k = 3, corresponding to 7 PE's per Bottom Chip, might be feasible for our initial prototype. Within a single Bottom Chip, the PE's will be configured geometrically according to a "hyper-H" embedding of the binary tree [Browning, 1978], as illustrated in Figure 3.2.

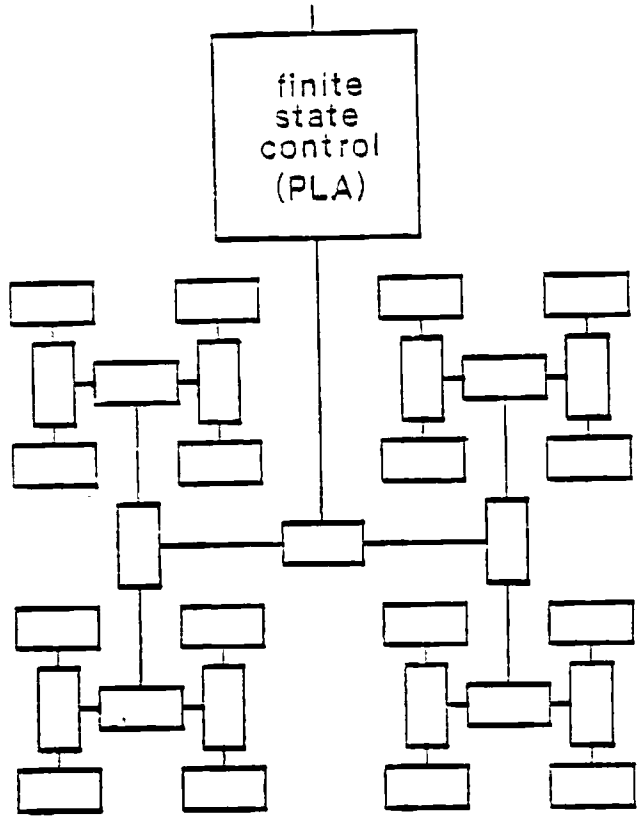

Figure 3.2 Structure of the PPS Bottom Chip

Because of its fixed I/O bandwidth requirements, independent of the size of the embedded subtree, the realizable capacity of the PPS Bottom Chip will increase quadratically with inverse changes in minimum feature width, thus permitting dramatic increases in the computational power of the NON-VON PPS unit as device dimensions are scaled downward with continuing advances in VLSI technology. (During the target time frame for a production version of a NON-VON-like machine, a k value of 7 or 8, corresponding to several hundred processing elements per PPS Bottom Chip, seems feasible.)

The PPS Middle Chip, on the other hand, will embed 2^m-1 "internal nodes" of the PPS tree (where m is a constant determined by pinout limitations, and independent of device dimensions), serving to combine 2^{m-1} subtrees, embedded either in separate Bottom Chips or (recursively) in lower-level subtrees rooted in other Middle Chips, into a single complete binary subtree. Because the number of processors per Middle Chip will be constrained by pinout limitations, and not by by minimum feature width, the capacity of the PPS Middle Chips will not benefit from the effects of scaling as will the Bottom Chips. This (provably unavoidable) I/O bandwidth limitation, however, will result in only a small, constant waste factor; the tree-structured intra- and inter-chip interconnection topology of the NON-VON Primary Processing Subsystem is in fact extremely well suited to the effects of future downward scaling.

4 Function of the Primary Processing Subsystem

The NON-VON PPS instruction set will support a number of operations involving associative retrieval, logical manipulation of local (to the individual PEs) flags, and a number of incidental functions (input and output, for example). While a discussion of all these functions is not within the scope of this paper, the two fundamental associative operations executed by the PPS hardware are of sufficient importance in the implementation of database management applications to merit special attention here.

In the context of the present discussion, these associative operations are probably best described in relational terms. Intuitively, a relation may be thought of as a table, with the rows referred to as tuples, and the columns called attributes. The relation shown in Figure 4.1, for example, expresses a part-whole relationship between airplanes and their (hypothetical) constituent parts.

PRODUCT	PART
DC-10	wheel
DC-10	engine-mount
DC-3	oxygen-mask
DC-10	oxygen-mask
DC-10	radio

Figure 4.1 Sample Relation

In performing associative operations, the NON-VON PPS unit functions as a

relatively fast, but inexpensive content-addressable memory, and may thought of as permitting the operation of relational selection [Codd, 1972] to be carried out in a short, fixed amount of time, independent of the size of the argument relation. Given a partial match criterion—that is, a set of attribute/value pairs which must be satisfied by any "matching" tuple—NON-VON is capable of either

1. associative marking: Simultaneously setting a flag bit in all PE's associated (in a manner to be explicated shortly) with a matching tuple, or

2. associative enumeration: Reading successive matching tuples out of the PPS unit (and into the control module) irredundantly, with each new tuple produced in a small, fixed amount of time.

In the sample relation of Figure 4.1, for example, the four tuples having "DC-10" as the value of their "PRODUCT" attribute could be simultaneously marked; alternatively, each of these tuples could be output in sequence, without the need for intervening search steps.

5 Lee's "Distributed Logic" Device

Before examining the exact manner in which tuples are stored and selectively retrieved in the PPS, it will prove instructive to consider the operation of a simpler associative device proposed by Lee [1962], whose operation is closely related to the associative mechanisms employed in NON-VON. Lee's "distributed logic" memory is based on a large, linear string of identical cells, each embodying a small amount (on the order of a byte) of storage, a few single-bit flags, and a modest amount of logic. Each cell is connected to (and can access the flags of) its immediate right and left neighbors; in addition, all cells are attached to a common broadcast bus.

Lee's device is capable of retrieving character strings on a content-addressable basis in time proportional to the length of the pattern string, but independent of the total storage capacity of the device. To illustrate his algorithm for associative parallel string matching, we assume each string to be stored in a contiguous sequence of cells, beginning with a distinguished delimiter symbol. To retrieve all strings beginning with, say, the sequence "SA", a command is first issued (over the common communication channel) instructing all cells containing a delimiter symbol to set their flag bits (independently, and in parallel). Next, a command is broadcast which instructs any cell whose leftmost neighbor has its flag bit set to turn off that neighbor's flag, but to set its own flag if its own storage byte contains an "S". During the third step, the match flag propagates another step to the right in all strings whose next character is an "A"; the matching strings are then easily identified.

6 Considerations for VLSI Implementation

It is not difficult to see how this associative string matching algorithm for a distributed logic memory can be extended to handle attributes and tuples, thus providing for a rapid parallel implementation of relational selection. There are several respects, however, in which Lee's approach is unsuitable for the implementation in VLSI of a practical Primary Processing Subsystem:

1. Direct broadcasting over a simple, single-level bus structure to a very large (and as device dimensions continue their downward trend, rapidly increasing) number of cells is, for reasons related to capacitive loading, impractically inefficient in a technology such as nMOS.

2. Although simple by comparison with a conventional microprocessor, the necessary matching, communication and control logic embodied within each cell would occupy considerably more area than the byte or so of local storage we have assumed in our description of the distributed logic memory device.

The NON-VON PPS design addresses the first concern by utilizing a single, hierarchically organized inter-PE data path to effect both the common broadcast and adjacent neighbor communication functions required for contiguous propagation-based parallel matching. The broadcast function is supported by interpolating simple level-restoring logic at each level in the tree, yielding a highly efficient structure for driving large capacitive loads. By choosing an appropriate order in which to sequentially enumerate the nodes of the PPS tree (two such schemes are now under consideration) and including the necessary I/O control logic within each PE, the same binary tree-structured data path may also be used for efficient communication between logically adjacent neighbors.

The second problem with a straightforward adoption of Lee's architecture for implementation in VLSI is solved in NON-VON by "amortizing" the cost (in chip area) of each PE's logic over a larger amount of local storage. Specifically, each PE embodied in the PPS Bottom and Middle Chips includes a full 32 bytes of random-access memory. Associated with each local store is a simple, area-efficient, byte-wide comparator module capable of testing for either equality or one of the five other arithemetic relations (less than, greater than or equal to, etc.) and of retaining the intermediate results necessary to sequentially perform a variable-length comparison between the stored and pattern values.

For simplicity, we may assume (at least in the context of this paper) that a given PE will store at most one tuple. The converse, however, is not the case: a single tuple, or even a single attribute value, could well exceed the capacity of one PE (there being no restriction on the length of either), and might in general be shared among several logically adjacent PEs. While the details are somewhat more complex, propagation among logically contiguous PE's

in such cases is effected in a manner quite similar to the analogous process in Lee's hypothetical distributed logic device.

It is expected that the time required for an associative matching operation will be quite close to that necessary to simply input the partial match specification through the broadcast tree, one byte at a time. After some consideration, it may be seen that no significant time cost is associated with our amortization of the comparator logic over a sequentially accessed memory when I/O and local processing times are well matched. It is only the problem of wasted capacity when the tuples are much smaller than the local storage capacity which prevents our amortizing the PE logic over an even larger amount of storage.

In short, the NON-VON PPS architecture offers the possibility of performing associative matching operations extremely rapidly — in fact, at a pace limited largely by the speed at which the partial match specification itself can be input. Through the exploitation of recent architectural advances applicable to VLSI systems, however, along with the careful balancing of storage capacity against distributed intelligence, we hope to bring the cost of PPS storage to within a small constant multiple of the price of an equivalent amount of ordinary random access memory implemented using comparable technology.

7 Associative Enumeration and Multiple Match Resolution

At the end of an associative marking operation, every PE that contains a matching tuple (or, in the case of large tuples, every PE in which a matching tuple starts) will have one of its internal one-bit flags set to 1. The corresponding register in all other PE's will be set to 0. In some cases, associative marking may be followed by another parallel operation involving this "mark register"; in other applications, however, it may be necessary to output all marked tuples (or the relevant portions thereof) through the root of the tree in an arbitrary sequential order, using the global communication bus.

This associative enumeration operation is supported by multiple match resolution hardware that uses the tree-structured communication path to very rapidly clear the mark register in all but the "first" marked PE. Because the order of enumeration is inconsequential for the database operations supported by NON-VON, any definition of "first" would have been acceptable for purposes of multiple match resolution. Our design preserves the mark flag within that matching PE which would be assigned the lowest node number in an in-order [Knuth, 1969] traversal of the tree. (An in-order enumeration of the nodes of a tree assigns a lower number to all nodes in the left subtree of a given node than is assigned to the node itself, and a higher number to all nodes in its right subtree.) In-order multiple match resolution has a particularly simple and elegant hardware realization within the NON-VON PPS.

8 Internal Evaluation of the Relational Algebraic Operators

As noted above, the PPS unit's associative marking and enumeration operations may themselves be regarded as implementations of the relational selection operator, which returns a relation consisting of all tuples satisfying a particular attribute-value specification. Selection, though, can be performed entirely within NCN-VCN's Secondary Processing Subsystem, obviating the need for transfer to, and processing within, the PPS. The importance of the associative marking and enumeration operations instead derives from its use as a building block in the implementation of the "difficult" operations (project, equi-join, and the set theoretic operations, for example) which, in contrast with relational selection, can not be evaluated by the SPS alone. While a detailed exposition of the algorithms for each of these "difficult" operations is beyond the scope of this paper, the essential behavior of the NCN-VCN PPS and SPS units may be illustrated by considering a single, particularly demanding operation which has a particularly simple realization within the NCN-VCN PPS: the equi-join.

The equi-join of a source relation R_1 with a target relation R_2 over the join attributes A_1 and A_2 produces as its result a relation containing one tuple derived from each possible pairing of one tuple r_1 from R_1 and one tuple r_2 from R_2 for which the join values $A_1(r_1)$ and $A_2(r_2)$ are equal. This pairing process is illustrated in Figure 8.1, which depicts a join of the PRODUCT—PART relation with a new relation, CUSTOMER—PRODUCT, over the PRODUCT attribute of each.

CUSTOMER	PRODUCT
American	DC-3
Western	DC-10
American	DC-10

PRODUCT	PART
DC-10	wheel
DC-10	engine-mount
DC-3	oxygen-mask
DC-10	oxygen-mask
DC-10	radio

Figure 8.1 Tuple Pairings in a Typical Join

In the equi-join operation, each result tuple is derived from its associated source/tuple pairing by concatenating the source and target tuples to form a new tuple having the attributes and values of both r_1 and r_2. The result of an equi-join is thus the union of a set of extended Cartesian products, one for each common join value. In the case of the more common variant known as the natural join, one of the two (necessarily identical) occurrences of the join attribute is removed in the result relation. The result of a natural join with the above parameters is illustrated in Figure 8.2.

CUSTOMER	PRODUCT	PART
American	DC-3	oxygen-mask
Western	DC-10	wheel
Western	DC-10	engine-mount
Western	DC-10	oxygen-mask
Western	DC-10	radio
American	DC-10	wheel
American	DC-10	engine-mount
American	DC-10	oxygen-mask
American	DC-10	radio

Figure 3.2 Result of a typical join

The join operation may in general be extremely expensive on a conventional von Neumann machine, since the tuples of the two relations must be compared for equality of the join attributes before the extended cartesian product of each group of matching tuples can be computed. In the absence of physical clustering with respect to the join attributes (whose identity may vary in different joins involving the same relations), joining is most commonly accomplished on a von Neumann machine by pre-sorting the two argument relations with respect to the join fields. The order of the tuples following the sort is actually gratuitous information from the viewpoint of the join operation. From a strictly formal perspective, the requirements of a join—that the tuples be paired in such a way that the values of the join attribute match—are significantly weaker than those of a sort. The distinction is moot in the case of a von Neumann machine, where no better general solution to this pairing problem than sorting is presently known. On the NON-VON machine, on the other hand, we are able to make use of the weaker constraints involved in the definition of the join operation to eliminate the need for pre-sorting.

The algorithm for the internal equi-join on the NON-VON machine is in fact extremely simple; it may be regarded as an associative version of the naive join algorithm in which each tuple in the source relation in turn is compared with all tuples of the target relation. On a von Neumann machine, this naive algorithm has quadratic (in the size of the argument relations) time complexity—specifically, the number of sequentially executed steps is equal to the product of the cardinalities of the source and target relations. Within the NON-VON PPS, however, only the source tuples are processed sequentially, since a given source tuple may be compared with all target tuples in parallel using the associative marking operation introduced earlier.

In applications where the result relation is to be retained in the PPS (as, for example, in the case where the join is to be followed immediately by another parallel operation on the result relation), the number of steps is thus equal to the cardinality of the source relation, and is independent of the cardinality of the target relation. Where it is necessary to sequentially output the result relation (either to the SPS, or to some system external to the NON-VON machine), a number of additional steps proportional to the size of

the result relation is required; the complexity of our algorithm, however, remains linear in the cardinalities of the source and result relations.

It should be noted that, in the worst case (in which the join attributes of both argument relations have only a single value, so that the join produces the full extended cartesian product of the two relations as its result), the result relation will itself contain a number of tuples equal to the product of the cardinalities of the two argument relations. As must be the case for any algorithm involving fixed-bandwidth sequential enumeration of the result relation, the NON-VON internal join thus has a worst case quadratic in the size of the argument relations considered alone.

Our analysis of time complexity in terms of the size of the argument and result relations is motivated by empirical observations involving the relationship between argument and result cardinalities in realistic relational database applications. In practice, the degenerate case of a single-valued join appears to appear so infrequently by comparison with joins in which the argument and result relations are of roughly comparable size that the argument/result model seems to better reflect performance differences of practical significance. Under the assumptions of this model, the NON-VON internal join algorithm offers a very significant advantage over the best algorithms known for a join on a von Neumann machine.

9 Organization of the Secondary Processing Subsystem

The SPS unit is a block-oriented storage and processing subsystem based on a (potentially quite large) bank of highly intelligent circulating mass storage devices. In our initial work, we have adopted an SPS design based on physically rotating devices, such as moving-head disk drives. Alternatively, the SPS could be constructed using non-inertial circulating storage devices based on magnetic bubble or CCD technologies, although such devices do not, at present, appear competitive on a cost-performance basis with rotating devices in this application. Either single- or multiple-head disk drives are suitable as a basis for the NON-VON PPS unit, but it is assumed (in the interest of economy) that the number of tracks considerably exceeds the number of read/write heads—that is, that "ordinary", as opposed to "head-per-track", drives are employed.

In the case of a PPS based on single-platter, single-head drives, all of the logic required for operation of the subsystem may be situated externally to the disk units, and the drives themselves need not be modified. One potentially practical realization of the NON-VON architecture—in fact, the one we would expect to adopt for purposes of prototyping the SPS design—might thus include a bank of unmodified single-platter, Winchester-type disk drives. Although the size of each drive might well be quite modest in such a design, the number of drives would ordinarily be quite large (perhaps several hundred) in most realistic applications. Because the mean time between single-drive failures could easily be unacceptably short within such a subsystem, we have

been forced to include drive-level fault-tolerance as a criterion for the adequacy of the SPS design. Although we have only recently begun to explore the options for incorporating such fault-tolerance within a single-platter SPS design, it appears that the structure of the PPS may permit the implementation of highly parallel error detection and correction mechanisms which might otherwise be impractically inefficient.

In the case of an SPS based on multiple-platter disk drives, on the other hand, special requirements are imposed on the physical design of the drives themselves. In contrast with a standard multiple-head disk drive, each head in such a drive would require its own sense amplification electronics, permitting all heads to simultaneously read their respective tracks. Although separate sense amplifiers are not commonly incorporated within standard multiple-head drives, it should be noted that drives capable of simultaneously reading from more than one head have in fact been marketed (in modest quantities) for several years at the time of this writing.

In either the single- or multiple-head designs, a small amount of hardware is associated with each disk head. The following capabilities are assumed for this "per-head" logic. First, it must be possible to examine an arbitrary (possibly compound) attribute in each tuple that passes under the associated head, comparing each such value against a single specified pattern value. As in the case of many earlier database machine designs (RAP [Ozkarahan, Schuster and Smith, 1974], CASSM [Su, Copeland and Lipovski, 1975] and DBC [Baum and Hsiao, 1976], for example), the SPS is able to collect for output all tuples found to match, or to bear some specified arithmetic relationship (less than, less than or equal to, etc.) to the given pattern value.

In addition, though, the NON-VON SPS is capable of sequentially computing a hash function for which the resulting hashed value falls within the range [0, 1]. Because the range of our hash function is fixed, the requirement for real-time hashing is well within the capabilities of the sort of simple and inexpensive hardware that would be required in a practical "per-head" logic unit. One implementation, for example, would combine the entire compound attribute into a single, fixed length "signature word" (of, say, 16 bits) by computing the exclusive or of each two-byte segment with the current accumulated signature word as it passes under the head. Tuples whose selected values hash to within specified subranges of the [0, 1] interval may be dynamically identified and transferred to the PPS unit.

As noted earlier, transfers between the SPS and PPS units, along with global control of the NON-VON machine, is the responsibility of a specialized Buffer and Control Unit. Although a number of interesting issues arise in the design of this subsystem, space does not permit a detailed discussion within the current paper.

10 External Evaluation of the Relational Algebraic Operators

Let us now consider the case where the arguments to the join exceed the capacity of the PPS device, which will be true in most applications of practical interest. Our strategy will be to partition the argument relations into mutually disjoint sets of buckets, the vast majority of which are small enough to fit entirely within the PPS. In general, one such bucket will then be transferred into the PPS during each successive revolution of the disk-based SPS, and an internal join performed on the subrelations in question. Partitioning is accomplished by examining in parallel the values of the join attributes of both relations as their constituent tuples pass under the many disk heads incorporated within the SPS.

The goal of the hash partitioning algorithm is to divide the argument tuples in such a way that each bucket ordinarily need be processed internally only once, leading to a linear-time algorithm for the join operation, even in the case of relations exceeding the capacity of the PPS. (Once again, it should be emphasized that we are using the equi-join operator only as a vehicle for illustrating the essential operation of the NON-VON engine. Projection, union, intersection, set difference, and a number of other "difficult" operators having similar properties are all implemented using the technique of hash partitioning, and are all executed in linear time.)

Unfortunately, not all partitionings support the achievement of this goal. If a given bucket contains a source tuple with a particular join value, for example, all target tuples having the same join value must be included within the same bucket if the full extended cartesian product due to this join value is to be formed during internal evaluation of the current bucket. If any join value is represented in more than one bucket, our strategy for linearity can not be employed. We refer to partitionings in which all tuples having a common join value are mapped to the same bucket as key-disjoint partitionings.

In the NON-VON SPS, common join values are assigned to the same bucket by hashing all join values into the interval [0, 1], and then assigning successive contiguous subranges of this interval to different buckets. If the combined size of the target and source relations were approximately four times the capacity of the PPS, for example, we might use the hash partitioning technique to divide the argument relations into five (allowing a small safety factor) key-disjoint buckets corresponding to the subranges [0, .2), [.2, .4), [.4, .6), [.6, .8), and [.8, 1].

Within the SPS, all relations are stored "across" the (typically large) set of individual intelligent disk drive units; more precisely, although single tuples are not split between drives, the tuples comprising a single relation will in general be distributed fairly evenly among all drives in the subsystem. During a single revolution of the bank of SPS drives, each head is capable of "looking at" all tuples passing beneath it. Unless the argument relations consume an unusually large share of the total amount of storage available within the system, it is thus possible to associatively examine, and perform simple processing on, the join values of all tuples in both argument

relations within a single disk revolution.

On the first revolution, all tuples whose join values hash into the subrange [0, .2) are dynamically identified, in parallel, by the "per-head" logic associated with all heads in the SPS unit, and are transferred "on-the-fly" into the PPS for internal evaluation. During the next disk revolution, all tuples whose join values hash into the second subrange are transferred into the PPS, and so on. While it is possible for the size of a single bucket to exceed the capacity of the PPS, the statistics of hashing give us a firm analytical basis for predicting the incidence of such "overflows". It has been shown [Shaw, 1979] that, independent of the distribution of join values, PPS overflows may be expected to occur so infrequently that, even if relatively inefficient, quadratic-time recovery procedures are adopted, these procedures should make only a very small, linear contribution to total running time.

11 References

Baum, Richard E. and Hsiao, David K., "Data Base Computers—a Step Towards Data Utilities", IEEE Transactions on Computers, v. C-25, December, 1976.

Browning, Sally, "Hierarchically Organized Machines", in Mead, Carver and Conway, Lynn, Introduction to VLSI Systems, Addison-Wesley, 1978.

Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", Proceedings of the 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, Association for Computing Machinery, 1971.

Codd, E. F., "Relational Completeness of Data Base Sublanguages", in Rustin, Randall (ed.), Courant Computer Science Symposium 6: Data Base Systems, Englewood Cliffs, New Jersey, Prentice-Hall, Inc., 1972.

Gallaire, Herve, Minker, Jack, and Nicolas, J. M., "An Overview and Introduction to Logic and Data Bases", in Gallaire, Herve and Minker, Jack, Logic and Data Bases, New York, Plenum Press, 1978.

Kaplan, S. Jerrold, Cooperative Responses from a Portable Natural Language Database Query System, Ph.D. Thesis, Department of Computer and Information Science, University of Pennsylvania, May, 1979.

Knuth, Donald E., The Art of Computer Programming, vol. 1: Fundamental Algorithms, Addison-Wesley, 1969.

Lee, C. Y. "Intercommunicating Cells as a Basis for a Distributed Logic Computer", Proceedings of the AFIPS 1962 Fall Joint Computer Conference, Baltimore, Spartan Books, pp. 130-136, 1962.

Ozkarahan, Esen A., Schuster, Stewart A., and Sevcik, K. C., "A Data Base Processor", Technical Report CSRG-43, Computer Systems Research Group, University of Toronto, September, 1974.

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, "A Relational Database Machine Architecture", Proceedings of the 1980 Workshop on Computer Architecture for Non-Numeric Processing, Asilomar, California, March, 1980. (Also reprinted in joint issue of ACM SIGARCH, SIGIR and SIGMOD publications.)

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

Su, Stanley Y. U., Copeland, George P., and Lipovski, G. J., "Retrieval Operations and Data Representations in a Content-Addressed Disc System", Proceedings of the International Conference on Very Large Data Bases, Framingham, Massachusetts, September, 1975.

Wiederhold, Gio, Kaplan, S. Jerrold, and Sagalowicz, Daniel, "The Knowledge Base Management Systems Project", ACM SIGMOD Record, 1981.