CUCS-17-81

A Highly Parallel VLSI-Based Subsystem of the NON-VON Database Machine

(Extended Abstract)

David Elliot Shaw
Hussein Ibrahim
Department of Computer Science
Columbia University

Gio Widerhold
Jim Andrews
Department of Computer Science
Stanford University

A Highly Parallel VLSI-Based Subsystem of the NON-VON Database Machine

(Extended Abstract)

David Elliot Shaw
Hussein Ibrahim

Computer Science Department
Columbia University

Gio Wiederhold
Jim Andrews

Computer Science Department
Stanford University

July 1981

Summary

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 includes a brief overview of the NON-VON project and a more detailed discussion of the structure and function of the PPS unit and its constituent processing elements.

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 problem 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, et al., 1978; Shaw, 1980] 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 (Wiederhold, et al., [1981]; Shaw, [1980]) have begun to construct systems capable of communicating in an even more "human-like" manner, 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 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 most time-consuming 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 "difficult" operators of the *relational algebra* defined by Codd [1972]. Among these operations are the set theoretic operators *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 design alternative machine architectures supporting the highly efficient 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.

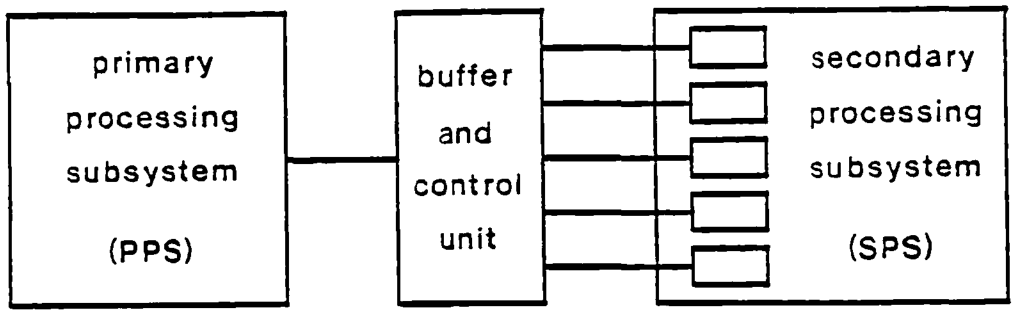
primary processing subsystem (PPS)
buffer and control unit
secondary processing subsystem (SPS)

Figure 2.1. Organization of the NON-VON Machine

2. Overview of the NON-VON Architecture

The theoretical basis for the NON-VON architecture was developed in the course of a doctoral research project at Stanford [Shaw, 1979], along with 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 an $O(\log n)$ improvement in efficiency over the usual evaluation methods employed on 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, described shortly) over the performance of previously implemented or proposed database machine architectures based on associative 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.

The proposed machine involves a *Secondary Processing Subsystem* (SPS) based on a bank of intelligent rotating storage devices and designed to provide very high access and processing bandwidth, along with a smaller, but faster *Primary Processing Subsystem* (PPS), again utilizing a high degree of parallelism, in which the operations in question may be very quickly evaluated. The top-level organization of the NON-VON machine is illustrated in figure 2.1.

Transfer between the two devices is based on a procedure called *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. Details of the SPS architecture, along with a description and analysis of the hash partitioning algorithm, have been presented elsewhere [Shaw, 1980a]; in this paper, we will focus on the structure and function of the PPS unit, which is being implemented using custom *n*MOS VLSI circuits.

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 (PE's)—on the order of several thousand, if a full-scale prototype were to be built using 1981 technology, and perhaps a hundred thousand 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 derived from the bus and from specified memory locations, and
4. I/O logic for local communication with the parent and (except for "leaf" PE's) two children, and for global data transfers, using the tree-structured inter-PE bus on a dedicated, "broadcast" basis, as explained in the next section.

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

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 according to a "hyper-H" embedding of the binary tree [Browning, 1978], as illustrated in Figure 3.2.

Because of its fixed I/O bandwidth requirement, 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

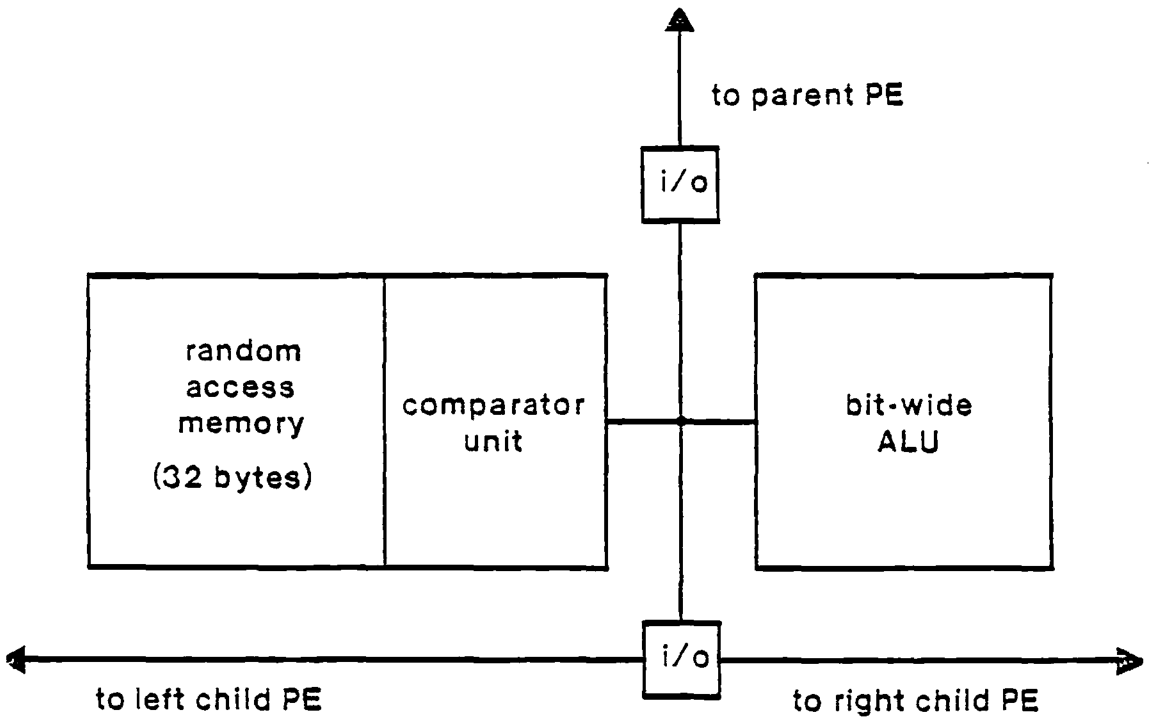

Figure 3.1. Components of a Single Processing Element

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 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 was designed to support a number of operations involving associative retrieval, logical manipulation of local (to the individual PE's) 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

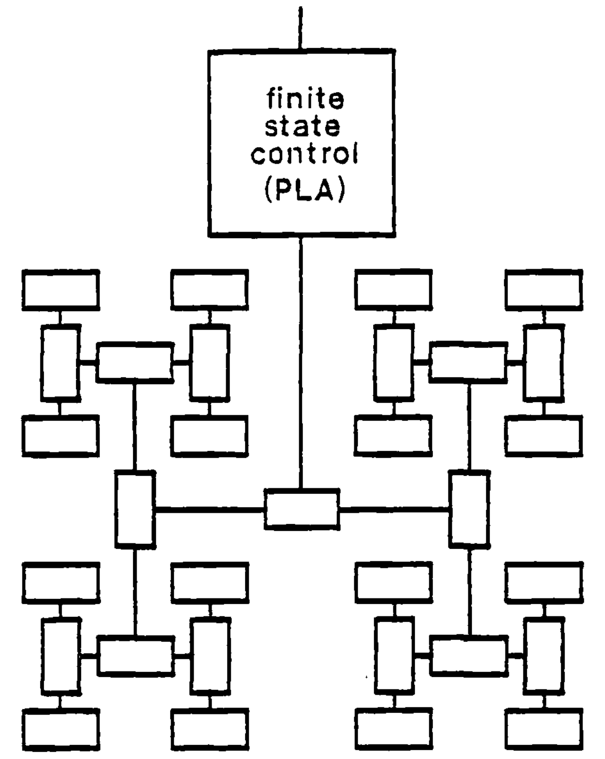

Figure 3.2. Structure of the PPS Bottom Chip

extended abstract, 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 below, 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*

In performing associative operations, the NON-VON PPS unit functions as a relatively fast, but inexpensive *content-addressable memory*, and may be 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 that 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.

Before examining the exact manner in which tuples are stored and selectively retrieved in the PPS, it will prove instructive to consider a simpler (non-VLSI-based) 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 communicatin 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.

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 at least two 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 Lee's 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 inverter 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 will be discussed in the final paper) 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 starage. Associated with each local memory is a simple, area-efficient, byte-wide comparator module capable of testing for either equality or one of the five other arithmetic relations ($\neq$, $<$, $>$, $\leq$, and $\geq$) 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 PE's. 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. (Details of this process will be included in the final paper.)

It is expcted that the time required for an associative marking 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 larger local 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 store which prevents our amortizing the PE logic over an even larger amount of RAM. 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 contemporary approaches to VLSI system architecture, 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.

In addition to the above discussion of associative matching, our final paper will briefly outline algorithms for

1. associative enumeration
2. multiple match resolution
3. sorting
4. certain arithmetic and statistical calculations

in order to convey a feeling for the kinds of operations for which the NON-VON architecture should prove well suited.

References

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.

Lee, C. Y., "Intercommunicating cells as a basis for a distributed logic computer", *Proceedings of the AFIPS 1962 Fall Joint Computer Conference*, Spartan Books Inc., Baltimore, Maryland, pp. 130-136, 1962.

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. (Reprinted in publications of ACM SIGARCH, SIGIR and SIGMOD.)

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

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