Synchronization, Communication and I/O Factors in Database Machine Performance

Andy Lowry
Columbia University Department of Computer Science
andy@cs.columbia.edu
April 13, 1988

Tech Report # CUCS-342-88

Abstract

Research in the area of Database Machines is surveyed with a focus on the effects of synchronization, communication and I/O burdens on the performance of proposed architectures. A number of machines described in the literature are used to illustrate the potential problems and the approaches that have been offered to address those problems.

This work was supported by the New York State Science and Technology Foundation's Center for Advanced Technolgy, under project number CAT(87)-5.

Table of Contents

1. Introduction

Over the past two decades, substantial research has been carried out, both in the United States and abroad, to develop specialized computer equipment to support database management systems (DBMS). The primary motivation behind most of the efforts has been the widely held contention that a conventional DBMS--a large software system running on a general-purpose uniprocessor along with other applications--is inherently limited in power and incapable of supporting very large database applications. Some of the factors contributing to the inadequacy of the conventional approach are:

1. *High cost of data access.* As it is impractical for the DBMS to scan every record in a large database, costly indexing strategies must be used to limit the scope of a search. See [Ullman 82] for a discussion of popular indexing methods.
2. *Limited processor speed.* Uniprocessor machines are not capable of delivering the power required by large database applications. The power must come from multiprocessor approaches.
3. *Resource competition.* The DBMS must compete for computational and I/O resources with other applications running on the machine.
4. *Operating system overhead.* Typical DBMS software is written as a collection of user-level applications that access machine resources via a general-purpose operating system.

The database machine (DBM) is an attempt to address these performance issues by removing all or parts of the DBMS function to one or more pieces of dedicated hardware. A classification scheme proposed in the preface to [Hsiao 83a] provides some insight into the broad spectrum of DBM approaches:

1. The *software backend* uses innovative software on an unmodified general-purpose computer dedicated to providing DBMS services, either directly to users or to other machines via a network or channel. The machine is provided with a highly stripped-down operating system (or none at all), allowing the database software to make unusually efficient use of the available hardware. In addition, the run-time machine environment is much more predictable than that of a conventional mainframe supporting multiple applications. This may lead to opportunities for great efficiency in software design.
2. The *intelligent controller* replaces the conventional disk controller and serves as a filter to reduce the volume of irrelevant data that must be processed by the host. In addition, the intelligent controller may effectively transform its associated disk into a content-addressable store, eliminating the overhead of disk address calculation (indexing) in the host.
3. The *hardware backend* includes hardware specifically designed or modified to support full DBMS service. The motivation behind these machines arises from the notion that specialized hardware should be capable of performing its tasks much more efficiently than a software implementation on a general-purpose machine.

Expected benefits of the DBM include:

1. *Improved Performance.* As mentioned above, this is the primary motivation behind most of the efforts. Improvements may be sought in terms of response time, throughput, or both. Secondary effects may result in improved performance for other applications on the host, due to the offloading of resource consuming database tasks.
2. *Increased Reliability.* In [Hsiao 79], reliability is actually listed first, ahead of performance, among the motivations for DBM's. Hsiao contends that the size and complexity of software DBMS's are such that there can be little confidence in the systems. He suggests that high-level hardware support would reduce the software complexity to a point where it could be readily understood and verified. However, [Date 83] points out that the resulting hardware complexity is at least as difficult to manage as the software complexity, and that

even a correct hardware implementation is subject to failure.[1]

3. *Growth Path.* Although in early DBM designs, growth was difficult at best and seldom available in small increments, growth potential has become a major concern in more recent designs. A growth path that offers incremental expansion to track increasing load and which does not require major disruption of service is ideal.

4. *Data Utility.* In [Baum 76], Baum and Hsiao argue the necessity of the DBM to act as data utility, playing a role in a multiple host network environment similar to the role played by public utilities in society: offering reliable shared access to resources by multiple clients, protection of the shared resources through security and integrity constraints, convenient access via high-level interfaces, coordination of concurrent access, etc. One of the original goals spurring research and development of DBMS was the desire of organizations to provide centralized access to their diverse information resources. The network data utility embodied in a DBM is one approach to meeting this need in the face of increasing decentralization of computing resources.[2]

The results of database machine research have been disappointing at best. After a prolonged period of great optimism and intense research, the years 1982 and 1983 brought the sudden sobering realization that the proposed machines had largely fallen short of the mark. The performance analysis by Hawthorn and DeWitt [Hawthorn 82], among the first detailed investigations into DBM performance in over a decade of research, showed that proposed DBM's offered little improvement over a conventional INGRES system with query precompilation.[3] In those queries where improvements were evident, the machines were based on storage and processing technologies that had not yet become cost effective. Boral and DeWitt argue in [Boral 83] that technological trends were actually causing those technologies to become *less* cost-effective or even outright infeasible, and therefore concluded that most of the proposed DBM's were of little practical value. Their further conclusion that I/O bandwidth limitation is by far the most pressing problem in DBM design has resulted in recent proposals that forego specialized processors and exotic storage technologies, relying instead on techniques to artificially boost the I/O bandwidth of conventional storage devices [Boyne 83, He 83, Hsiao 83b, DeWitt 86].

This survey explores problems encountered by DBM researchers in the areas of multiprocessor synchronization, interprocessor communication, and management of the I/O burden. Several proposals are used to illustrate potential performance barriers as well as approaches taken by researchers to address these problems.

This is not a survey of DBM architectures; many of the machines used for illustration are not fully explained, and many proposed machines are not mentioned at all. There are several good surveys available along these lines, including [Bray 79, Hsiao 80, Ozkarahan 86, Su 88]. Of these, [Su 88] is by far the most complete and up-to-date, and offers a uniform performance analysis for several classes of DBM. For complete and coherent descriptions of the machines used in this work the reader is referred to the bibliography.

---

[1]Dr. Hsiao himself appears to have been convinced on this point, as he is now actively involved in the design and development of a multiprocessor database machine that makes no use of specialized hardware [Demurjian 85].

[2]The other major approach to this issue is the distributed DBMS. See [Date 83, chapter 7] for an introduction and further references.

[3]The word *query* is used throughout in an unrestricted sense to mean requests made on the database, whether or not those requests require updating the database.

The following chapters are organized according to natural boundaries as defined by the use of a DBM configuration. First, chapter 2 explores the host-DBM interface. Next, chapter 3 discusses the DBM-database interface--the problem of getting the data into the DBM for processing. Finally, chapter 4 examines the synchronization and communication requirements between processors within the DBM.

Most of the DBM's discussed in this paper have been designed to directly support the relational data model defined in [Codd 70]. The CASSM machine [Copeland 73, Lipovski 78] provides direct support for the hierarchical data model, while MDBS [Hsiao 83b, He 83] and DBC [Banerjee 78a, Banerjee 79] are based on a data model in which records consist of arbitrary collections of attribute-value pairs. In all cases, the machine is capable of supporting the relational data model by means of translation, if not directly. For descriptions of the relational and hierarchical data models, the reader may consult virtually any of the introductory database texts currently available, such as [Date 86, Ullman 82, Korth 86].

2. Host-DBM Communication

The performance benefit of offloading some or all of the database processing activity from a host to special-purpose dedicated hardware seems to have been accepted almost as gospel by early DBM researchers. Attention was directed almost entirely at opportunities to reduce the workload on the host, without recognizing factors that would create new processing burdens and additional workload for the host. Primarily, this burden comes from the communication required between the host and the DBM. Additional difficulties arise when synchronization is required in order for communication to proceed.

Date [Date 86,p. 348] emphasizes that it is not even sufficient for the net workload on the host to be reduced; the reduction must be substantial, as indicated by the *offloading theorem*:

> For offloading to be cost-effective, the amount of work offloaded should be an order of magnitude greater than the amount of work involved in doing the offloading.

It is not entirely obvious that communication costs incurred in a DBM configuration could be close to the cost of I/O for comparable (or even higher) data traffic in a host DBMS. In fact, a common emphasis in many DBM designs is that there will be a drastic reduction in the amount of irrelevant information that must be transmitted to and processed by the host. In [Hsiao 80], Hsiao offers the following characterization of the problem of irrelevant information:

> For very large data bases, the *90-10 rule* says that, in order to find the relevant data, nine times as much additional irrelevant data will have to be brought into the main memory for processing; that is, only 10% of all the data processed will be relevant to the request or query.

If the 90-10 rule is on target, it is difficult to imagine how a realistic attempt to reduce irrelevant data volume could be ineffective. Two observations help to explain this. First, although data volume is substantially reduced, delivery time may still be quite high, as the data will not be immediately available for delivery. Second, the delivery costs may increase substantially.

2.1 Backend Configurations

Drawin and Schweppe [Drawin 83] studied the effect of host-DBM communication for backend database machines and concluded that communication overhead was by far the dominant factor in determining overall response time. Communication bandwidth was found to be a surprisingly insignificant factor above an easily achievable minimum level (approximately 24K bits per second).

The effect of high data delivery time is apparent in this study, which does not assume a substantial decrease in query execution time on the DBM compared to the host. Thus, although the DBM provides a reduced workload for the host, response time cannot be significantly lessened.

2.1.1 High Cost of Interprocessor Communication

Communication cost in a host-backend configuration is substantially higher than the cost of disk I/O, for a number of reasons. Most important is the necessity of using high-overhead protocols to synchronize communication between two independent processors. Message packet formats typically include a substantial amount of overhead data to support protocols, and this overhead factor increases as message size decreases. Message acknowledgements are required in many cases, and except in cases where a "piggy-backing" technique is feasible, these acknowledgements present an additional, pure overhead

load.[4] Message length is often limited by the protocol or even by physical characteristics of the communications channel.[5] Operating system services to support message generation and receipt are costly, as complex protocols must be followed, and often several levels of buffering are encountered before the message data finds its way into the application's address space.[6]

In contrast, disk controllers are designed for synchronous operation with the host processor, thus costly protocols requiring acknowledgement are avoided. In addition, direct memory access (DMA) devices deposit data directly into the computer's main memory with minimal effect on processor activity. Large blocks of data may be transferred in this manner without requiring processor attention. There are no layers of overhead data surrounding the retrieved data, so each data byte must be copied at most once into the application's address space. Moreover, a reasonal DBMS would be expected to deal directly in memory pages, so that once a page has been filled with data from the disk, it need only be assigned to the application's address space, with no copying required.

In a single host configuration, it is perhaps conceivable that the communication cost could be reduced by providing an interface that makes use of some of the efficiencies described for the disk controller. In a network configuration where the backend serves many hosts, however, the communication costs seem unavoidable.

2.1.2 Response Set Cardinality

One of the parameters studied in [Drawin 83] was the choice between a tuple-at-a-time interface and a set-oriented interface. It is pointed out that even in systems based on a set-oriented data model such as the relational model, internal software layers are often built on tuple-at-a-time services, so that such an interface may be the most convenient choice for a DBM designer.

Intuitively, a DBM providing tuple-at-a-time access should perform much worse than one operating with a set-oriented interface. The reason is that (ignoring acknowledgements), two messages must be transmitted for each tuple retrieved. In the ideal situation of set-oriented access, only two messages are required for the entire query: one to transmit the query to the backend, and one to send results back to the host.

In practical terms, the difference may not as pronounced as expected. In fact, Drawin and Schweppe found only marginal differences between the two choices. In part, this is accounted for by the fact that the response set size was held constant at four in their simulation. Aside from this, the authors point out that even if the host and backend exchange tuple sets, the application program still may require tuple-at-a-time service. In this case, there is a software burden caused by buffer management necessary to provide the tuple-at-a-time application interface. In addition, message packet length restrictions discussed above imply that a single message be be insufficient to transmit the entire response set. If only a few tuples can

---

[4]Piggy-backing is only effective if an outgoing message is (almost) immediately available following the receipt of an incoming message. This would not be typical in a backend DBM situation, as outgoing messages would usually be generated by the application only after it has received and processed the incoming message. Even if processing is minimal, the operating system overhead involved in delivering the message to the application consumes more time than could typically be allowed.

[5]For example, in a token ring network, the ring is physically lengthened by switching a local buffer into the circulation path in order to insert a message. The size of the buffer therefore limits the message size.

[6]The reader who is unfamiliar with data communication protocols will find a clear introduction in [Tanenbaum 81].

fit in a message packet, the reduction in response messages is minimal. In any case, there will be a substantial reduction in messages from host to backend, since the host no longer needs to send a request message for each tuple. Furthermore, if response messages arrive in quick succession, a moving-window protocol can be used to reduce the number of acknowledgements needed from host to backend.

2.1.3 High-Level Query Language

A consideration related to the response set cardinality issue is the form in which queries are presented to the DBM by the host. In [Epstein 81], Epstein and Madrid discuss the host-DBM communication issue with respect to the IDM-500, a backend database machine commercially available from Britton-Lee. They stress the importance of a high-level query language that allows an entire request be to specified in one communication, as well as set-oriented result set retrieval. A single query may require a number of operations, including selections, joins, projections, etc. An interface that required the host to specify each low-level operation could require several message exchanges even before the first results could be reported. The high-level query language implies a capability on the part of the DBM to parse and compile, and preferably optimize, a schedule of primitive operations (a "query tree") to implement the query.

Another decision made by Britton-Lee was the use of a binary encoding for queries. Although this allows for more concise query messages, it is not clear that the extra work required on the part of the host is less than the resulting savings in communication costs, especially in cases where the unencoded query could fit in a single message packet.

Most DBM proposals have emphasized the importance of a high-level query language, although the term "high-level" is certainly not used consistently. In most cases, a logical data manipulation language such as SQL is the intended meaning. In the case of relational databases, data manipulation languages are generally declarative in nature, rather than procedural. However, several DBM's require that queries be specified procedurally. In the early cellular logic devices such as CASSM [Copeland 73, Lipovski 78], for example, queries must be compiled by the host into a "high-level" assembly language program, with instructions for marking records satisfying a query condition, activating subroutines actually stored in the database, computing aggregate functions on marked records (sum, average, maximum, etc.) and other such high-level operations. The term "high-level" here refers to the available instruction set, rather than the level of language abstraction. The RAP machine [Ozkarahan 75, Schuster 79] took a similar approach.

DIRECT [DeWitt 79, Boral 82] also requires query compilation on the part of the host. In this case, a lower-level instruction set was used to yield programs for multiple query processors cooperating in the query execution. The compilation process made use of parameterized templates for relational operations, which would be augmented with custom code to perform query-specific comparisons, field extraction, etc. The decision to place the compilation burden on the host was listed as a mistake in [Boral 82], due to the increased host-backend communication burden. A better alternative, according to the authors, would have been compilation by the control processor, which is already present and is responsible for overall coordination of the query processors.

2.1.4 Other Considerations

Two other facilities offered in the IDM-500 are precompiled stored commands and a flexible buffered output interface.

The stored command facility is supported by allowing the user to define parameterized command sequences and then later invoke the sequences by suppling only the command name and parameter values. A command cache speeds access to the stored commands, which also reside permanently on the database mass storage. In addition to allowing for very concise invocation of command sequences, the command execution time can be reduced because prevalidation of the commands in the sequence is performed at definition time. Normal checks for proper attribute names, security violations and the like are then dispensed with at execution time.

As was mentioned above, CASSM also offered the possibility of storing assembly language subroutines in the database for later execution. In fact, the developers of CASSM have proposed an entire programming technique called "associative programming" based on the possibility of automatically triggering these stored subroutines upon certain attempts to read or modify data items [Su 77]. This facility has been investigated as a basis for sophisticated security and integrity constraint enforcement [Hong 81, Hong 82].

The flexible output interface suggested in [Epstein 81] is designed to place minimal burdens on the host in delivering query results to applications. An internal buffer in the IDM-500 collects result records and signals the host when data is available. The host then initiates the transfer when it is ready. This makes it possible for the host to delay transfer until the requesting application is in a runnable state. The operating system can swap in data areas for the application process and then transfer result data directly to those areas, thereby bypassing operating system buffers that might be necessary were the host required to accept the data when the target data areas are swapped out.

2.2 Content Addressable Storage Devices

The idea of mingling processing logic with storage gave rise to the notion of *associative*, or *content-addressable* storage devices. Such devices allow stored data items to be identified by means of their contents, rather than by physical location on the device [Su 88,p. 9]. In the extreme case, a processor is associated with each bit of storage, and all processors can operate in parallel to perform search operations. Such a device is termed a *fully associative* memory. At the other extreme is the conventional von Neumann arrangement, wherein a single processor is responsible for access to all memory elements. Between these two extremes are devices of varying degrees of associativity, termed *pseudo-associative* memories.

Three classes of content-addressable storage device have been proposed for use in database machines:

1. *Highly Associative Memories.* These devices are typified by relatively small amounts of semiconductor storage (tens or hundreds of bytes) per processor.
2. *Processor Per Track (PPT) Devices.* These devices, first proposed by Slotnick [Slotnick 70], are built from head-per-track (fixed head) disks by associating a processor with each disk head. This approach is sometimes referred to as *cellular logic*, where the name derives from the view of a PPT device as comprising several moderately sized memory *cells*, each supported by independent processing *logic*.

3. *Processor Per Surface (PPS) Devices.* These devices do not rely on the costly fixed head disks required for a PPT device; moving head disks are used, again with processing logic added to each read/write head. Thus each processor is responsible for an entire disk surface.[7]

2.2.1 Some Highly Associative Memory Devices

An early example of a highly associative memory device is STARAN, developed by Goodyear Aerospace for the United States Air Force and used in real-time applications such as air traffic control [Rudolph 72, Moulder 73, Berra 79]. STARAN consists of a 256x256-bit memory array, in which each 256-bit data word (row) is operated upon by a processor in a bit-serial fashion.

The Massively Parallel Processor (MPP) is another machine developed at Goodyear Aerospace, following initial design by NASA scientists, and intended for applications involving two-dimensional arrays of data requiring primarily local neighbor interactions, such as weather prediction, image processing, and certain numerical applications [Batcher 80, Batcher 85]. MPP consists of a 128x128 array of processing elements, each configured with a 1K-byte local RAM and a 32-bit shift register. Davis describes algorithms that carry out relational database operations using the MPP as a highly associative memory, with each of the 16K processing elements able to perform matching operations in parallel on locally stored tuples (SIMD operation[8]).

A third example of a highly associative memory is the primary processing subsystem (PPS) of the NON-VON supercomputer designed and prototyped at Columbia University [Shaw 79, Shaw 81, Shaw 8x]. The NON-VON PPS is a massively parallel (as many as a million elements) collection of small processing elements (SPE's) arranged in a binary tree interconnection with a moderately powerful control processor (CP) at the top. Each SPE is equipped with a small (on the order of 32 bytes) local RAM. Associative processing is achieved by means of three communication primitives supported by the binary tree: *broadcast*, by which the CP is able to send data to all SPE's simultaneously; *resolve*, which selects a single SPE from those participating in the operation; and *report*, by which a single SPE (perhaps selected by a prior resolve operation) is able to send data to the CP. Associative arithmetic operations (e.g. sum, count, max) can also be performed quickly using tree-neighbor communication (send to/receive from parent/left child/right child).

2.2.2 Some PPT Devices

The Context Addressable Segment Sequential Store (CASSM), a research database machine designed and prototyped at the University of Florida, was the first of the cellular logic database machines [Copeland 73, Lipovski 78]. CASSM was designed to directly support the hierarchical data model, with each tree of the record hierarchy laid out in a linear pre-order fashion along consecutive cell memories. The relational data model is represented by means of a two-level hierarchy, with a level-one record representing a relation, and its subordinate level-2 records representing the relation tuples. Trees, and

---

[7]Some writers refer to these devices as *processor per head*, or PPH devices. We agree with [Date 86] that PPS is a more informative and accurate designation.

[8]SIMD (Single instruction stream/multiple data stream) operation is one of four categories in a classification of multiprocessor computer architectures proposed by Flynn [Flynn 72]. The other classifications are SISD (single instruction stream/single data stream), MISD (multiple instruction stream/single data stream), and MIMD (multiple instruction stream/multiple data stream).

even individual records are allowed to span cell boundaries with complete transparency, and this accounts for a great deal of complexity in the cell logic.

The Relational Associative Processor (RAP), designed and prototyped at the University of Toronto [Ozkarahan 75, Schuster 79], is another cellular logic device designed to support the relational data model directly. The data format of RAP is more compact than that of CASSM; however since each memory cell is restricted to storing tuples from a single relation, a certain amount of wasted storage is almost unavoidable. The RAP cell processors include multiple comparators, allowing complex selection criteria to be applied on a single cell rotation where CASSM would require several rotations. In the RAP.2 design [Schuster 79], a data staging mechanism was provided to allow RAP to process databases that are too large to reside on a single fixed-head disk. The data staging works by assigning two tracks (cells) to a single processor, and loading data into one cell while its twin is being processed.

A third cellular logic device is the Rotating Associative Relational Store (RARES) designed at the University of Utah and supporting query-only (i.e. no update) support for relational databases [Lin 76, Lin 77]. The most significant departure from the earlier cellular logic designs is the orthogonal data organization. RARES is not truly a PPT device, since a cell comprises a *band* of several adjacent disk tracks, all managed by a single cell processor. The bytes of a tuple are laid out in a radial fashion along the tracks in a band, with large tuples occupying several adjacent radii. Wasted space arises from the requirement that all tuples in a relation follow the same storage pattern, so that if tuple length is not a multiple of band width, bytes in the last occupied radius for a tuple will be unused. To alleviate this problem, a relation is not required to make use of all the tracks in a band, and bands are allowed to overlap. However, if a tuple occupies additional radii as a result of a layout that uses less than the available band width, the advantages of the orthogonal layout scheme (to be discussed) will be correspondingly reduced.

2.2.3 Some PPS Devices

The first important PPS device was the Content Addressable File Store, developed and marketed by International Computers Limited of England [Coulouris 71]. Although CAFS is technically not a PPS device, it acts as if it were, since the CAFS processing logic operates on a datastream that is multiplexed from all the heads of a moving head disk. Such a disk, sometimes called a *parallel read-out* or *cylinder read-out* disk, is able to send an entire cylinder's worth of data to CAFS in a single revolution. Thus, CAFS processes one cylinder of data per revolution, in accordance with the PPS model. The processing logic of CAFS is an example of class of devices known as *on-the-fly filters*. The primary mission of a filter is to remove useless data from a data stream in an attempt to reduce the effects of the 90-10 rule discussed in the introductotry remarks for chapter 2. The CAFS filter utilizes multiple simple comparators operating in parallel (MISD operation) to apply complex selection criteria to each tuple in the tuple stream. A projection filter is also used to remove unneeded fields from selected tuples. If all the comparators are not needed for a given query, additional concurrent queries can be accomodated. In this case the host will need to distribute the resulting tuples to the various queries, as CAFS reports what amounts to the set union of the individual query results. In [Babb 79], additional filter hardware is described to support join and duplicate elimination operations.

SURE (Such-Rechner) is another on-the-fly filter developed at the Technical University of Braunschweig in West Germany [Leilich 78, Schweppe 83]. As in CAFS, a parallel read-out disk is used

to provide a multiplexed data stream for SURE. The fourteen selection units in SURE operate in MISD fashion on the data stream, as do the comparators in CAFS. However, the SURE selection units are significantly more powerful since they can be programmed to perform several tests per tuple and logically combine the individual test results to form a single match bit for the tuple. Due to the lack of buffering in SURE, the selection unit programs are constrained to access the tuple data in the order it is received, thus making it necessary to assign multiple selection units to the query if multiple tests are required on a single attribute (such as would be the case, for example, in range test). As in CAFS multiple concurrent queries can be executed by partitioning the selection units among the queries. For each tuple passing through SURE, the match bits from the individual selection units are concatenated, along with a unique ID for the tuple, to form a match word to be passed on to the host. The match word generator is capable of logically combining the individual match bits, as well as performing threshold tests (e.g. *at least two of the following conditions are met:* ...) to determine whether or not to send the match word to the host.

The NON-VON machine, already discussed in the context of highly associative memories, also makes use of on-the-fly filters as the basis of its *secondary processing subsystem* (SPS). The SPS comprises several small single-surface disk units, each equipped with an on-the-fly filter. The disks are connected to the PPS by associating one disk with each SPE at a certain level of the tree. Parallel loading of of the PPS is accomplished by loading each subtree of the PPS rooted at the SPS level with data from its attached disk. The filters associated with the SPS disks are used to perform the bulk of selection and projection processing on NON-VON (although the PPS can also perform these functions efficiently), and also to support external join algorithms.[9] The NON-VON SPS operates as a true PPS device, although *logical* cylinders consisting of tracks from several disks, rather than *physical* cylinders, are processed in parallel.

A unique on-the-fly filter fed by parallel read-out disks is proposed by Lehman for execution of simple relational transactions in a high-volume environment (e.g. automatic bank tellers, airline reservations, credit card authorization) [Lehman 84]. Lehman's proposed filter consists of a programmable systolic array arranged in a two-dimensional grid. Each row of the array is programmed to execute a single transaction, and many (i.e. thousands of) compatible transactions can be programmed into separate rows the array. The relations required by the transaction are fed down through the columns of the array, with each attribute occupying a column. The i^{th} attribute in a tuple precedes the $(i+1)^{th}$ attribute by one systolic time unit. Each systolic element can be programmed to perform a single operation (comparison or arithmetic) on a data item received from its upper neighbor, and can pass a result value to its lower neighbor, as well as a single control bit to its right and lower neighbors. The control bit is used to suppress further processing on a tuple that does not meet selection criteria. Unlike CAFS and SURE, Lehman's systolic array is capable of evaluating arbitrarily complex query criteria, including multiple access to a single data item, by compiling a single transaction into multiple consecutive rows. There is a penalty, however, since some systolic elements must typically remain unused when multi-row transactions are programmed.

---

[9] *External* algorithms must be used to carry out operations in cases where the operand data exceeds the capacity of whatever processing hardware is being used (the NON-VON PPS, in this case). Conversely, *internal* algorithms are used when the operand data can be accomodated in its entirety by the processing hardware. Generally, external algorithms involve iterations of the corresponding internal algorithms operating on the operand data in batches.

2.2.4 Analysis of Host Communications

Of the content addressable storage devices discussed, STARAN is perhaps unique in that its high cost and severely limited capacity make its use totally unjustified unless data can be passed through at an extremely high rate. Otherwise the processing becomes I/O bound, as the time required for associative search becomes insignificant in comparison to time required to load and output data [Berra 74, Berra 79]. The required data rates can pose technological and cost problems on both sides of the device.

The PPT devices suffer from a synchronization problem caused by the possibility that several cell processors may wish to output to the host simultaneously. In all three of the devices discussed above (CASSM, RAP, and RARES), this problem is handled by an output arbiter which allows only one of the contending cell processors to proceed with its output operation. Other processors must mark their records and wait for the next cell rotation to try again. Thus, output contention results in additional cycles required to complete the output operation, even though there may be quiet periods later in the current cycle.

The RARES orthogonal data layout was motivated by a desire to reduce the likelihood of output contention. Since any cell performing output occupies the output channel for the duration of its scan of the record being output, the decrease in record scan time resulting from the RARES organization causes a decrease in output contention. As data from several tracks are multiplexed onto the output channel, the achievable data rate is also increased.

The goals of high-level functionality discussed in section 2.1.3 are met to varying degrees by the machines discussed in this section. STARAN seems most suited to applications in which it functions solely as a filter, since its severely limited capacity requires higher level operations such as join and duplicate elimination to be executed by means of several passes through the device. Such a primarily external execution is unlikely to compete with execution in the host. MPP and NON-VON, with their larger capacities, are more likely to be able to process these operations internally. In addition, with the help of the SPS filters, efficient hash-based external algorithms can be used to great advantage on NON-VON.

CASSM and RAP both provide support for implicit join processing[10] using different techniques. The CASSM cell processors are each equipped with a one bit wide local RAM that can simulate backward intracell marking operations by retaining mark information to be applied in the next rotation. The local RAMs can also be used for intercell marking by treating the entire collection of local RAMs as a single global RAM (GRAM) with a bit corresponding to each record on the CASSM (for pointer operations) or each possible field value (for intercell value matching). Since access conflicts to the GRAM may occur, instructions requiring intercell marking may take several instruction cycles (disk rotations).

The RAP approach to implicit join processing uses tuples from the source relation to construct selections for the target relation. The combined results of the target relation selections form the implicit join result. Due to the multicomparator structure of the RAP cell processor, several source tuples can be used for this operation on each iteration, with the target query consisting of a disjunction of queries formed from the contributing source tuples.

---

[10] An *implicit join* of a source relation and a target relation produces those tuples of the target relation that are joinable with source tuples. It is equivalent to a join followed by a projection onto the attributes of the target relation.

RARES is unique among the cellular logic devices in its support for sorting operations. If the tuple data are not already in the desired sort order, RARES can perform an initial pass in which a histogram is computed giving the number of times the sort field value falls within each of a number of small intervals. During each subsequent revolution, some collection of consecutive intervals is used to select tuples to be sent to the host. The number of intervals used on each cell rotation is restricted to prevent outputing more tuples than the host can sort during the following rotation. Collection of cell output and internal sorting are pipelined in the host. The complete sorted relation is obtained by concatenating the results of the internal sorts. An identical algorithm is proposed for sorting on NON-VON, with interval selection constrained so as not to exceed the PPS capacity. Internal sorting within the NON-VON PPS is a linear time operation.

Among the PPS devices, CAFS and SURE primarily serve as agents for data reduction by performing selection processing. Due to lack of buffering, the SURE processor suffers from a severe problem in that tuple ID's, not tuple data, are reported to the host in response to a query. This means that additional disk accesses are required in order to retrieve the tuple data. The approach used in CAFS provides a much more useful interface, with tuples routed simultaneously to the selection unit (comprising the comparators and combination logic) and the retrieval unit. The latter component buffers the tuple data, removes unneeded attributes as specified in the query, and suppresses the tuple from the data stream sent to the host if the selection unit indicates failure.

Actually, the CAFS design may result in effects similar to the SURE tuple retrieval problem. This is because the CAFS database is maintained in two separate pieces, named the content addressable store (CAS) and the direct access store (DAS). The CAS, which provides the CAFS filter data stream, contains only abbreviated summaries of the full records stored on DAS, with a DAS pointer attached to each summary. In the case that tuple data not present in the summary record is required, the retrieval unit must obtain the information from DAS before passing the tuple on to the host.[11]

In [Babb 79] Babb describes an enhancement to the CAFS selection unit to facilitate implicit join and duplicate elimination, thereby raising the level of services offered by CAFS. The filter is equipped with a number of bit vectors and mechanisms for addressing individual bits in the vectors via tuple data. For an implicit join, bits are turned on as qualifying source tuples pass through the filter. As the target relation is subsequently passed through the filter, tuples whose join data address zero bits in the bit vector are suppressed, even if they pass selection criteria. The host need only retrieve qualified source tuples if an explicit join is required.

Two bit-vector addressing mechanisms suggested for CAFS are (1) precomputed *coupling indices* with different values of the relevant tuple data assigned different indices, and (2) on-the-fly hashing of relevant tuple data. The coupling index approach, aside from being impractical for applications where arbitrary field combinations might be relevant in different situations, also reduces effective output data rate since the coupling indices, extending the size of each tuple on the CAS, thereby reduce the rate of tuples

[11] It is not clear from the cited sources how the on-the-fly filtering of CAS data can proceed when DAS accesses are required. It would appear that a the projection unit must be equipped with a buffer for tuples pending DAS access, or there must be a mechanism to suspend and later resume filtering of CAS data. A combined approach could reduce the likelihood that suspension of filtering would be required.

received by the filter from the CAS.[12]

The LEECH processor [McGregor 76] employs a bit-vector filter similar to that adopted for CAFS, but explicit join support is enhanced by utilizing two bit vectors. In LEECH, the source and target relations are passed through the filter and processed exactly as in CAFS, except that qualifying source tuples are not sent to the host. In addition, qualifying target tuples are used to address a second bit vector and turn on corresponding bits. The source relation is then passed a second time through the filter, and tuples whose join data address zero bits in the second bit vector are suppressed. There appears to be nothing that would prevent a similar algorithm from being implemented on CAFS, given that multiple bit vectors are already present for other reasons.

Finally, host communication is not an issue for the remaining filters, as neither filter passes tuples on to a host. The NON-VON SPS communicates only with the NON-VON PPS, and Lehman's systolic array is used for relation update rather than retrieval.[13]

---

[12]On the other hand, duplicate elimination based on hashed addressing necessarily risks suppressing nonduplicate tuples due to hashing collisions. Multiple bit vectors addressed by independent hash functions can reduce, but not eliminate the likelihood of this problem.

[13]It would actually be a simple matter to add tuple output capability to the array. An additional column could be programmed to generate a flag bit for tuples to be output, and an output collection unit could then distribute flagged tuples as they exit the bottom of the array.

3. Database I/O

The high cost of data access was identified in the introduction as one of the major reasons for the failure of the conventional DBMS approach to adequately support large database applications. This cost arises from a combination of the 90-10 rule, inappropriate mass storage addressing, and inefficiencies imposed by the storage hierarchy which is almost universally present in conventional computer systems.

The 90-10 rule makes it impractical to scan entire relations to satisfy a query, as only a very small fraction of the tuples in each referenced relation will end up contributing to the result set. Therefore a content addressing capability is needed to retrieve only relevant tuples. Unfortunately, conventional storage devices do not offer such a capability, so a conventional DBMS must explicitly provide content addressing via artificial, and usually approximate, means. This invariably involves some form of indexing, whereby precomputed data structures are consulted to convert query specifications into a list of potentially relevant record addresses. This list, which, it is hoped, includes only a small fraction of irrelevant records, can then be used to retrieve and test each of the indicated records, discarding those that do not qualify.

Due to the extreme cost/speed tradeoffs characterizing available storage technologies, virtually all current conventional computer systems make use of a storage hierarchy, with a small amount of fast, expensive main memory directly available to the processor and a comparatively massive quantity of inexpensive rotating magnetic storage accessible only by means of copying needed data into and out of main memory. The storage hierarchy compounds the data access problem faced by a DBMS in the following two ways: First, data access is slow, requiring significant delays (due to seek and latency times) before data transfer can begin, and then providing relatively low data rates. Second, and closely tied to the first effect, block transfers involving significant quantities of data are enforced for all disk accesses, in an attempt to amortize the seek and latency costs over a number of accesses. This amortization pays off for most applications, where locality of reference ensures that a high percentage of the contents of a block will be useful. For database applications, however, locality of reference is often quite low, so the 90-10 rule finds another application in that a large percentage of the data actually transferred from disk to main memory will be worthless to the query at hand.[14] Focusing on individual tuples, which are typically stored as whole units on the disk, the 90-10 rule strikes again since often, many attributes are immediately projected out of retrieved tuples.

The combination of indexing and the storage hierarchy incurs an additional cost since the index structure itself is usually quite large (often exceeding the size of the database proper [Berra 87]), so that it too must be maintained on mass storage. Index access thus reduces available I/O bandwidth for database transfers.

The following sections explore approaches taken by DBM researchers to alleviate the database I/O bottleneck.

---

[14]Clustering techniques, which attempt to place related records in physical proximity, can be used to improve locality of reference, but only for queries that fit the clustering strategy adopted; no tuple-oriented clustering strategy can increase locality of reference for all queries.

3.1 Content Addressable Memory

Content addressable memory devices were introduced in section 2.2, where their output behavior was studied. In this section, further motivation and analysis is presented, focusing on access by these devices to their underlying storage.

The primary motivation for the use of content addressable memories in support of database operations is the elimination of costly indexing requirements. As pointed out above, indexing is essentially a software implementation of content addressability. Providing content addressability directly in the storage device therefore obviates the need for such measures. Substantial economies in both time and space may be realized as a result.

Another benefit of content addressable memories is the elimination of the storage hierarchy within the DBM. In NON-VON, for example, relational operations are carried out directly in the PPS (assuming operand relations fit). Similarly, the cellular logic devices (CASSM, RAP, and RARES) attempt to provide a full complement of high-level database instructions that operate directly on the memory cells. Post processing by the host may still incur storage hierarchy costs, such as when the reported result set is so large that the host must swap portions in and out of main memory during processing. In devices that offer less functionality, such as CAFS (without the bit-vector enhancement) and SURE, the processing burden on the host is correspondingly increased, thereby increasing the likelihood of host storage hierarchy costs.

Potential output delivery delay has been discussed previously with respect to content addressable storage devices. A related issue is their extremely low processor utilization during typical operation. The 90-10 rule states that each processor will be passively passing by irrelevant information during the majority of its operation cycle.[15] This underutilization is aggravated when contention arises in PPT devices, either for the output channel or (in the case of CASSM) for access to remote portions of the GRAM. This is because the contention forces the current instruction to extend into the subsequent instruction cycles, where progressively lighter unfinished workloads will remain.

In contrast to the overall underutilization of processing power, peak utilization may be quite high, and this can pose problems, especially for on-the-fly filters. A tradeoff arises between increasing I/O bandwidth (using techniques such as multiplexing of several data streams, as in the parallel read-out disks) and keeping peak processing loads within the limits of the filter. CAFS makes use of multiple simple comparators acting in parallel to enable complex filtering at high data rates. SURE adds an element of complexity by allowing the selection units to perform tests on several tuple fields in succession, but peak computational burden is not increased as a result, as the data operated on by the individual tests cannot overlap. Thus additional tests are performed during what would be idle periods for the CAFS comparators. The NON-VON SPS, being a true PPS device, does not suffer increased peak load at the individual processors as a result of additional I/O parallelism. Lehman's systolic array is capable of performing arbitrarily complex selection tests at peak data rates as a result of the systolic

[15]It is noted that Lehman's claims of extremely high processor utilization in his systolic array seem outrageously misguided. In a typical case, each systolic element will be actively involved in useful work during only one time step, when its target data arrives. This is one time step out of perhaps millions required to stream a relation through the array. During all other time steps, the element is simply passing data. Even if the element must act on the data in order to pass it, calling such action useful work is, in this author's opinion, highly misleading.

progression of data and the ability to use several rows to program a single selection.[16]

3.2 Paritítioned Content Addressable Memory

The concept of a partitioned content addressable memory (PCAM) was introduced by the designers of DBC, a database machine designed and prototyped at Ohio State University [Banerjee 78a, Banerjee 78b, Banerjee 79]. The PCAM is a cellular logic device where the memory cells are partitioned, and cell processors act within one partition at a time. In DBC, PCAM was introduced as a way to increase the capacity of DBM's based on content addressable storage devices. DBC's main memory is a PCAM implemented as a PPS device with each *logical* surface actually comprising corresponding surfaces from a large bank of moving-head disks. In a single cell rotation time, any cylinder from any disk in the disk bank can be scanned associatively by the cell processors.

Since the cell memories are distributed over a large partition, DBC uses a complex associative indexing scheme to identify cells (cylinders) that might be relevant to a given query. With several queries active concurrently, the disk controllers can be directed to seek in a somewhat optimized pattern to fulfill the queued cylinder requests.

The PCAM appears to be a compromise between the desire for content addressable storage and technological factors that make PPT devices impractical (high cost and limited capacity). The result can be seen as a merging of limited hardware support for content addressing with additional content addressing support through indexing.

PCAM was used for database storage in another database machine also named DBC, this one developed by Sperry Univac and modeled to some extent after the Ohio State DBC machine [Freeman 79, Champine 79].

3.3 Anticipatory Paging

One of the interesting benefits of the relational data model in database management is the fact that operations may be performed on the tuples in an argument relation in any order whatsoever, without affecting the result (although some efficient algorithms require sorted relations). This fact is used to advantage in multiprocessor DBM's that utilize anticipatory paging to keep processors supplied with pages from operand relations.

The first DBM to use this approach was the DIRECT machine designed and prototyped at the University of Wisconsin at Madison [DeWitt 79, Boral 82]. DIRECT consists of a collection of query processors and a collection of memory page frames, with a crossbar switch that allows arbitrary assignment of page frames to processors (including multiple readers of a single page frame). The query processors operate in MIMD fashion, with the processor pool partitioned at any given time among several concurrent queries. A backend controller (BEC) is responsible for allocating processors to queries and for making needed pages available in the page frames. The rearrangement-invariance of relations allows for a logical page request protocol, wherein a query processor that has finished processing its current page of data requests the "next" unprocessed page from the relation. The BEC is allowed to hand back any

[16] But note that there is no possibilty of intra-tuple field comparisons.

page frame containing an unprocessed page, as there is no real notion of "nextness." This organization provides opportunities for anticipatory paging, wherein the BEC can load pages into page frames in anticipation of page requests that it knows will be forthcoming.

In order to reduce the likelihood that intermediate result pages need to be swapped out of their page frames, a dataflow approach to query execution was proposed, in which query processors are assigned to an operation as soon as pages of its operand relations begin to appear. Note that the logical page addressing mechanism permits extreme flexibility in the allocation of processors to operations, as it is possible to assign additional processors to operations in progress at any time, and likewise a processor can be removed from an operation as soon as it has finished processing its current page.

GRACE, a relational database machine researched and prototyped at the University of Tokyo [Fushimi 86], also makes use of an anticipatory paging strategy. In GRACE, intermediate query results are accumulated in several hash buckets in a large staging memory. When any bucket becomes too large it is swapped out to a secondary memory. Later, when the bucket is accessed for a subsequent operation, the data is swapped back in anticipation of its use, without regard to the tuple rearrangement that effectively takes place.

Despite the attractiveness of anticipatory paging strategies, the designers of DIRECT found paging to be a disastrous bottleneck in overall machine performance [Boral 82]. An unanticipated effect of MIMD operation and the paging strategy was that page utilization was often low for operation results. Low utilization would occur when several query processors cooperating on a query, writing results to separate page frames, each produced only a small number of result tuples. The resulting decrease in page availability had an adverse affect on paging activity, as would be expected.

3.4 Disk Cache

A large semiconductor disk cache, or staging memory, can potentially reduce storage hierarchy effects for database operations, particularly in the case of temporary relations created as intermediate query results. If the intermediate results can be maintained in the disk cache without ever being written to the disk, the potential savings in disk traffic can be substantial. Additional savings can be realized by keeping large portions of the indexing structures resident in the disk cache.

The Delta relational database machine was designed and developed at the Institute for New Generation Computer Technology (ICOT) as part of the Japanese fifth generation computer effort [Murakami 84, Shibayama 84, Sakai 84, Murakami 85]. Delta consists of two major processing subsystems: the RSP (Relational DBM Supervisor and Processing subsystem) and the HM (Hierarchical Memory subsystem). The HM consists of a large general-purpose mainframe computer equipped with a very large, fast, nonvolatile semiconductor disk cache and a bank of high-capacity moving head disks. The task of the HM is to create, manage and destroy relations and their data on behalf of the RSP, which contains specialized hardware for fast execution of relational operations. Multiple high-speed channels connect the HM and RSP and are tuned to match the processing speed of the RSP. The disk cache is used to stage relations from the moving head disks, as well as for temporary storage of intermediate results. The latter include both complete relations resulting from intermediate query operations, as well as batches involved in external operations when operand relations exceed the capacity of the RSP hardware.

SPIRIT-III is another Japanese research machine being developed at Hiroshima University in cooperation with Mitsubishi Electric Corporation [Kamibayashi 82]. A large semiconductor staging memory is interposed between mass storage and processor workspace to establish a three-level memory hierarchy. Unique in the SPIRIT-III design is the addition of three types of on-the-fly filters to the data channels connecting adjacent memory stages. Selection and projection filters are similar to those found in CAFS, while hash-based grouping filters automatically distribute tuples into physically disjoint buckets in the destination memory stage to facilitate parallel join and other operations.

RDBM is the successor to the SURE effort at Technical University of Braunschweig [Schweppe 83]. Inflexibility of the SURE filter was the basis for a decision to decouple filtering from the raw data stream by means of an intermediate buffer. RDBM includes several specialized hardware components that share a large RAM to store intermediate relations.

3.5 Eliminating the Storage Hierarchy

As mentioned in section 3.1, one approach to reducing the effects of the storage hierarchy is to do away with it altogether. This is achieved with the content addressable memories by eliminating the main memory component of the hierarchy. The opposite approach is to design a machine with nothing but fast, random access memory. The Massive Memory Machine (MMM), proposed by researchers at Princeton, is one such machine [Garcia-Molina 84]. This can be viewed as a step beyond the disk cache.

The MMM proposal suggests that the machine be equipped with on the order of tens of gigabytes of RAM. The main problem anticipated with such a machine is the degradation in memory access that would be caused by long bus lines in a memory conventional architecture. A solution is proposed that utilizes on the order of a hundred identical processors, each managing a portion of the RAM. The configuration is novel in that it is a rare example of a proposal for a multiprocessor machine operating in SISD mode for reasons other than fault tolerance.

Each individual processor in MMM communicates with its local chunk of RAM via a moderately short local memory bus. Remote data access is achieved by means of a global memory bus, also of moderate length, connecting all the processors through controllers called ESP's. The name is derived from the fact that processors need not make memory requests for remote access; the required data simply appears on the bus at the right time, as if through some sort of precognition on the part of the ESP controller. This works because each memory read is local to some processor, and that processor sends the data item along the global bus as soon as it is read, knowing that all the other processors will be waiting for it.

Relational processing on the MMM would make use of indices and use standard conventional techniques, except that due to the efficiency of fine-grained access to any point in the RAM, indices can be made much more accurate than is currently the case.

3.6 Multi-Backend Machines

One recent response to the problem of insufficient I/O bandwidth has been the introduction of a class of database machines termed *multi-backend database systems* (MBDS) by Demurjian and Hsiao [Demurjian 85]. Such a machine consists of a collection of identical backend machines (of the software backend variety in all the current offerings), each independently managing its own set of mass storage devices. Interprocessor communication is required to perform some operations, but attainable effective

I/O bandwidth is much higher than with a shared disk system accessed via a disk cache or other means. The reason is that shared resources need not support the data traffic imposed by mass storage accesses, which are handled locally. The MBDS approach thus appears to offer oppotunities for highly parallel database machines with good growth patterns, assuming interprocessor communication in support of database operations does not become a bottleneck.

One example in the MBDS class of database machines is the DBC/1012™ machine manufactured and marketed by Teradata Corporation [Shemer 84, Neches 85, Neches 86, Su 88]. The Teradata machine utilizes from six to 1024 backend processors connected in a unique configuration called the *Ynet*.

Another MBDS is the machine under development at the Naval Postgraduate School in Monterey[17], which I will refer to as NPS-MBDS[18] [He 83, Hsiao 83b, Demurjian 85]. The machine consists of a number of software backends that communicate over a high-speed bus network (currently ETHERNET) and are controlled by a single control processor.

A final entry in the MBDS category is the GAMMA machine, designed and prototyped at University of Wisconsin at Madison by some of the researchers who developed the DIRECT machine [DeWitt 86]. GAMMA consists of a number of backend machines, perhaps including some diskless backends, connected via a very high speed network providing point-to-point communication. A single control processor is responsible for submitting queries for execution and reporting results back to the originating host.

3.7 Data Clustering

Data clustering is a technique that attempts to increase the proportion of useable information retrieved in each disk access. Clustering strives to locate records that are likely to participate in the same queries physically close to each other. Note that clustering is not useful in machines that do not maintain indices, since in such machines all tuples in a relation must be examined regardless of clustering.

Perhaps the earliest database machine to provide clustering mechanisms beyond the relation level was DBC [Banerjee 78b]. For a relational database, DBC clusters each record first by its relation name, and within that top-level cluster, by a cluster number determined by hashing a combination of fields specified by the database administrator. That is, all tuples comprising a relation reside on as few memory cells (disk cylinders) as possible, and records hashing to the same cluster number within a relation are stored together. Cluster assignment for new tuples is performed by the DBC control processor.

Attribute-based clustering is used in Delta. The advantages of attribute-based clustering are two-fold. First, the intra-tuple consequences of the 90-10 rule, due to the necessity of retrieving unwanted tuple fields, are eliminated. Second, all searches benefit from attribute-based clustering, rather than only those that refer to attributes used in the tuple-based clustering of a particular relation. The disadvantage of attribute-based clustering is that tuples are fragmented in the database, and the required attributes must

[17]The project began at Ohio State University with some of the same researchers who were responsible for DBC, and later moved to the NPS in Monterey with the principle researcher (Hsiao).

[18]The source materials refer to the machine as simply MBDS, which is potentially confusing since the entire class of machine is called MBDS. To add to the confusion, some references call the machine MDBS.

be located and concatenated in order to report results. In addition, attribute-based clustering can require more storage due to linking information (such as tuple ID in the Delta scheme) that must be stored along with each attribute value to allow for tuple reconstruction.

Delta maintains a three-level index structure for relation data. The attribute name is used at the top level of the index to locate a collection of value range tables. Each entry of these tables points to a TID (tuple ID) range table, and each entry of a TID range table points to a collection of value-TID pairs. Content searches for attribute values enter with the attribute name and then use the query constraints to select potential value range table entries to be followed. All pointers in the resulting TID range table are then followed to locate potential matches. Location of an attribute value for a given tuple requires searching of all the value range table entries found via the attribute name. The resulting TID range table entries can then be selected for further searching on the basis of the given tuple's TID. The HM is responsible for breaking up inserted tuples and for reconstruction of tuples as needed for reporting or for processing by the RSP.

A flexible tuple clustering scheme is provided in NPS-MBDM, wherein the DBA can specify individual clustering patterns for any or all of the attributes in a relation. These patterns define several partitions on the relation, one for each clustering attribute. A *supercluster* is then defined for each possible intersection of a set of clusters in which one cluster is chosen from each partition. The superclusters form the basis for clustering the relation tuples within each backend machine.

GAMMA supports a two-level tuple clustering scheme in which each tuple is clustered first according to the relation to which it belongs, and then optionally by means of a secondary clustering attribute specified by the database administrator.

3.8 Horizontal Partitioning

Horizontal positioning is a load balancing technique for MBDS machines. Relation tuples are distributed among the backend machines in an attempt to achieve a roughly even distribution, so that the workload for parallel operations on the relation will be uniformly distributed among the machines, resulting in a higher degree of parallelism.

The Teradata DBC/1012 using a randomizing hash function to distribute tuples evenly among the backend processors.

In NPS-MBDS, horizontal partitioning of each supercluster is performed in a blocked round-robin manner. That is, the control processor maintains a table of how many blocks are allocated to each cluster by each backend machine. When a new tuple is added to a cluster, the control processor checks to see if there is room for the tuple in any of the cluster blocks. If so, the tuple is placed in that block. Otherwise, a new block is allocated for the cluster on one of the backends currently storing the least number of blocks for the cluster, and the new tuple is placed in the new block.

In GAMMA, horizontal partitioning and clustering are independent of one another. The database administrator may specify any of the following partitioning schemes for a relation:

1. *Round Robin.* This is like the NPS-MBDS policy, except that the policy is implemented at the tuple level rather than the block level.
2. *Hashed.* The database administrator selects one of the relation attibutes as the *hash*

4. Interprocessor Synchronization and Communication

Having examined in previous chapters the problems associated with getting data into and out of the DBM, we explore in this chapter the difficulties associated with employing multiple processors in a single DBM. Synchronization issues arise for the following two reasons:

1. *Algorithmic Requirements.* If several processors cooperate to process a single query, synchronization may be needed in order ensure that all necessary operations are performed, without replication, and in the correct order.

2. *Concurrency Control.* In cases where the DBM is capable of processing several queries concurrently, either by distributing the queries among different subsets of processors or by interleaving execution of portions of the queries, care must be taken that operations are performed in a manner that results in consistency, both among the responses generated for the several queries, and in terms of the final database state.[19]

In fact, among the DBM's that allow concurrent execution of multiple queries, the task of concurrency control is handled by a single control processor, which is also responsible for scheduling query execution on the DBM. Thus there is nothing fundamentally different from concurrency control on the DBM's that have been proposed and concurrency control on a conventional DBMS. The designers of the MMM suggest that concurrent query execution, which is motivated by a desire for increased throughput, may not be needed given the throughput boost offered by their machine [Garcia-Molina 84]. Although other proposals are not this explicit, it may be that concurrent execution is absent from some other DBM designs partly because their designers ascribed to this view.

There is one DBM which is an exception to the preceding claims, in that a novel approach to concurrency control is presented. In Lehman's systolic array, one of the primary goals was to produce a machine capable of massively concurrent execution. It turns out that due to the systolic effect and the organization within the array of the individual operations of the transactions, all transactions in a batch are guaranteed to execute in a serializable fashion.[20] Thus concurrency control in the array is important only in the pipelined scheduling of transaction batches.

Synchronization requirements represent a burden and a potential performance barrier in a multiprocessor architecture. The synchronization mechanism may itself impose an overhead processing burden on the system, as when a handshaking protocol is needed to ensure that two processors have both arrived at a synchronization point. In addition, processor utilization is reduced if two processors do not arrive at a synchronization point simultaneously, as the earlier processor must then wait for the other to catch up.

4.1 Synchronization For Free

One approach to reduce synchronization overhead costs is to organize the operation of the system around some periodic event that all processors can observe. If this event is a necessary part of the operation of the system for other reasons, then some synchronization is achieved at effectively no additional cost. The SIMD machines (including associative memories and cellular logic devices) generally

---

[19]The reader wishing an introduction to theoretical and practical issues in concurrency control is referred to [Bernstein 87].

[20]A *serializable* execution of a set of transactions is a possibly interleaved execution of the operations required by those transactions that is guaranteed to be equivalent to some non-interleaved (i.e. *serial*) execution of the same transactions. The need for serializability of transaction executions is one of the fundamental tenets behind most work in concurrency control.

fall into this class, where the synchronizing event may be a pulse of a shared processor clock or the rotation of a disk. The effect is also seen in the MISD machines (the data filters CAFS and SURE), and even in some MIMD machines (e.g. Lehman's systolic array), where synchronization is provided by the progression of a single shared data stream (MISD) or by the inherent synchronization of the multiple data streams (MIMD).

It should be noted that while this approach may effectively eliminate synchronization overhead, synchronization requirements may still cause major penalties. For example, it was noted in section 3.1 that the SIMD machines and the filters typically exhibit extremely low processor utilization. From a synchronization point of view, this can be seen as idle time caused by the need to reach synchronization with some other component. This component may be another processor within the DBM, or it may be an external component. An example of the former case is the NON-VON PPS, where certain processors disable themselves after deciding that the current stream of instructions does not apply to their local data. Those processors must then wait for an algorithmic synchronization point with the still active processors.[21] Alternatively, the other component may be the external agent controlling the synchronizing events, such as the data source for MISD machines.

An interesting synchronization effect is noted in the case of the SPS-PPS interface in NON-VON. Due to the fact that the PPS operates in a SIMD fashion, with all SPE's operating in lock-step unison, the individual SPS units are forced to act in synchronization whenever they must interact with their associated SPE's. This is a difficult problem, since synchronous operation of the disk units themselves is technologically impossible.[22] The solution proposed in [Hillyer 86] is to equip each SPS unit with a moderate sized RAM buffer (a track-sized buffer is recommended) to allow synchronized access to track data despite unsynchronizable disk activity.

4.2 Dataflow Style Operation

The dataflow concept organizes computation around synchronization points established by the natural progression of data through the computation. Thus, with regard to a particular data item communicated between two processors, one of the processors is a *producer* of the data item while the other is a *consumer* (there may be multiple consumers, and even redundant producers for a given data item). A synchronization point is naturally established and enforced whenever a consumer requires a data item, since if that item has not yet been produced, the consumer must wait for the producer before continuing. This requirement may be enforced to varying degrees. For example, if a consumer makes reference to a piece of data solely for the purpose of passing it along to another processing stage, without basing any decisions on its value or reporting the value to an external agent, the consumer can be allowed to proceed before a definite value has been established. In some cases, this can actually eliminate the need to produce a value, as a later processing stage may decide to discard the data item altogether.[23]

---

[21] In extreme cases, the control processor may continue to broadcast instructions even though all processors have become disabled, simply because the cost of constantly checking for this situation (a resolve operation is needed) is too high.

[22] The use of alternative bulk storage technology for the SPS memories, such as bubble or CCD memory, would allow tight synchronization, but this would add considerably to the design complexity of the SPS. If inertial bulk storage such as magnetic disk is used, as is recommended in the NON-VON reports, synchronization is impossible due to random variations in platter rotation speeds and startup times.

[23] Processing strategies attempting to capitalize on such phenomena are sometimes termed *lazy evaluation* strategies.

One example of the use of dataflow techniques was noted in the discussion of paging strategies for DIRECT (section 3.3). In this case, a query processor is assigned to an operation as soon as its operand relations begin to emerge from other query processors working on operations that produce those relations. This is in contrast to the simpler strategy of waiting for each intermediate relation to be produced in its entirety before allowing any operations that make use of the relation to begin. In section 3.3 this strategy was described in relation to its potential for decreasing the total paging activity by reducing the chance that a page of an intermediate result would need to be swapped out prior to its use in a later operation. From a synchronization point of view, another benefit is that the resulting finer grained synchronization can result in better processor utilization and reduced overall response time. The flexibility in query processor assignment allowed by the logical (next-page) addressing scheme even allows the control processor to partially overcome one of the dangers of dataflow scheduling, namely that consumer processors may encounter long delays waiting for data from producers. For example, suppose a query involves a join of two relations resulting from selection operations. If the selectivity factor (percentage of tuples meeting selection requirements) is low, production of data pages resulting from the selections may be few and far between. A query processor assigned to the join operation may therefore experience significant idle periods. In DIRECT, due to the logical addressing scheme, the control processor is free to release a processor from such an operation and even reassign it to the operation for which it was previously waiting. Thus consumer becomes producer, and processor utilization is increased.

In RDBM, the shared RAM is managed by a dedicated processor, which allocates segments of RAM for relations upon request. The normal mode of access to relation tuples is via a logical addressing scheme similar to that provided by DIRECT, although in this case the granularity is at the tuple, rather than the page level.

In GRACE, resources for a query are allocated by the control processor prior to initiation of query execution. Processors allocated to particular operations in the query retrieve their data from preallocated areas in the shared staging memory, and synchronization is based on access to these areas. In order to increase the opportunity for parallelism in certain operations, producers may partition their output relation to a collection of separate memory areas based on a randomizing hashing function. In order to reduce contention between cooperating producer processors requiring access to the same memory modules, each producer places its output in a single memory module, partitioned within the module according to the hashing function. Thus a single hash bucket is horizontally distributed among the various memory modules. A consumer processor is then assigned to each bucket, and the assignment of memory module to consumer processor is cycled so that every processor eventually gets access to every memory module. In each step, each processor accesses the tuples within its assigned hash area in its currently assigned memory module. This mode of operation is another benefit made possible by the invariance of relational operations under rearrangement of relations. Note that the cycling of memory module assignment to consumer processors may exhibit some synchronization delays if the hash bucket portions are not uniformly sized during a given step of the cycle.

GAMMA utilizes a dataflow approach in which producer-consumer data exchange is accomplished by means of message passing among processes running on the various nodes. At query startup, a scheduler process establishes a collection of operator processes at the nodes to perform the operations required by the query. Each operator process begins either by retrieving data from its local disk (selector processes) or by waiting for operand data from other processes. In addition, each process is equipped with a "split table" which it uses to determine where each of its result tuples should be sent. Local

communication software recognizes when a process sends a message to another process on the same node and short-circuits the normal communications software in that case. The distribution of all operand relations among the processors (except in cases where knowledge of horizontal partitioning indicates that only one processor can possibly possess relevent data) allows for a high degree of parallelism, but runs the risk of generating a great deal of message traffic. Message volume is reduced by means of a number of algorithmic techniques, primarily applied to join processing. In this case, consistent hash partitioning is utilized on the two operand relations to ensure that, following distribution of the relations, purely local processing is adequate to complete the join. To reduce traffic required for distribution of the source relation, the target is first distributed, and each node hashes received tuples and constructs a bit vector showing which buckets contain relevant information. The target relation bit vectors are collected by the scheduler process and then distributed to the operator processes assigned to the source relation. The bit vectors are added to the split tables for those producer processes, and messages carrying tuples which cannot contribute to the join result can thereby be suppressed.

A VLSI based DBM described by Kiyoki, Kato and Masuda [Kiyoki 86] combines dataflow synchronization with functional programming concepts. A fundamental feature of the operation of this machine is that each processor carries out its operations with no knowledge of the history of its operands. This allows a flexible approach to resource allocation, making available techniques analogous to the common subexpression elimination found in optimizing compilers. In the case of operand relations that can be determined to be identical, the scheduler can dynamically balance storage and computational costs in determining whether to retain a relation for reuse or recompute the relation when it is needed again.

4.3 Component Speed Matching

In designing the DBC machine, much attention was paid to the performance match between communicating processors. Thus, for example, a great deal of analysis was done to determine the proper performance goals for the structure memory in order to keep the main database memory operating at peak speeds without incurring excessive cost due to "overkill." This design orientation, while offering certain cost advantages, also contributed to what resulted in an extremely complex design, and makes performance growth difficult in DBC. The design philosophy behind DBC focused on the use of highly optimized special function hardware components to achieve performance believed impossible with general purpose hardware.

An example of the lengths to which this approach took the DBC design is in the design of hardware to access and maintain directory information. The primary component serving this function is the structure memory (SM), which includes several processors, the structure memory processors (SMP's), each accessing a local CCD memory. Data in a query is used by each SMP to access relevant directory information in its memory via hashing. Hash buckets are uniformly distributed among the structure memory units so as to increase the potential for parallelism among the structure memory processors. Each SMP maintains a directory in RAM to keep track of the distribution of each hash bucket among the memory modules constituting its local memory unit. Each selection criterion in a conjunctive query yields a set of main memory unit addresses (cylinder numbers) which are fed to a structure memory information processor (SMIP). The SMIP itself contains several processors, each with a local RAM. The SMIP is responsible for forming the intersection of the directory results reported by the SM for the individual selection criteria. This is done by hashing each reported main memory unit address to select a processor

in the SMIP that will store the address. The processor hashes the address to a bucket in its local RAM and there maintains a count of how many times the address was reported by the SM. At the conclusion, each SMIP processor reads out those main memory unit addresses that were reported for each selection criterion. As if all this were not enough, the addresses must be sent on to an index translation unit, which converts the addresses, which are actually stored in the SM as cluster-relative addresses to reduce storage costs, into global main memory addresses before passing them on to the main memory processor. One last point: updates to SM were considered too time-consuming to be performed on the fly, so an additional look-aside RAM buffer, maintained by the structure memory controller (SMC), was used to temporarily store the updates. The SMC uses the look-aside buffer to alter the information reported from the SMP's before it is passed on to the SMIP. When a slack period arrives or the look-aside buffer fills, the updates are applied to the SM by the SMC.

With the exceptional performance gains witnessed in general-purpose microprocessors, design extremes of sort witnessed in DBC become unjustifiable in most situations, as pointed out in [Boral 83]. This is especially true in light of the almost inevitable requirement for performance growth, which for a machine such as DBC, may well require an entire redesign, or at least an extensive reengineering.

Nevertheless, functional specialization may be beneficial, especially in the case where the special purpose hardware has potential application in other areas besides database management. A good example is the sort engine included in the design of Delta. The heart of the RSP is a special-purpose hardware component called the relational database engine (RDBE), which is essentially a twelve-stage pipelined merge-sorter with associated I/O interfaces and comparator logic at the bottom to carry out joins, unions, duplicate elimination and the like. A design incorporating such a specialized component must address the speed-matching problem to ensure that no component becomes a bottleneck for the overall system. In the case of Delta, these considerations dictated high-speed dedicated unidirectional channels between the HM and the RDBE's, as well as the large HM staging memory. The extent to which the designers were able to push HM performance was in turn a limiting factor in the number of RDBE's that could be designed into the system (four).

4.4 Dynamic Load Balancing

Load balancing is a general technique that can be used to reduce the effects of algorithmic synchronization points. The idea is to distribute the processing burden as uniformly as possible among the processors, so that they will arrive at synchronization points in roughly equal amounts of time. Again, the ability to arbitrarily arrange relations in the relational model turns out to be a benefit in that it allows for some fairly simple dynamic load balancing strategies.

The hashed distribution of intermediate result tuples in GAMMA is one example of a load balancing strategy that can be used to great benefit. A more exotic scheme has been proposed by Frieder for cube-connected multiprocessor machines [Frieder 87]. The processor interconnection network in such a machine implements a binary n-dimensional cube, wherein each processor is able to communicate directly with its n neighbors.

In Frieder's scheme, the cube is partitioned into disjoint k-dimensional subcubes, of which there will be 2^{n-k}. Each subcube includes 2^k processors whose addresses in the n-cube are identical in the first $n-k$ bits, which can be though of as forming a unique subcube ID. Simultaneous inter-subcube communication between corresponding processors (those processors in the two subcubes whose n-cube

addresses match in the last *k* bits, which define a *column* of the *n*-cube) is possible in time proportional to the number of bit positions in which the subcube ID's differ. Each column is served by a local disk, which is attached directly to one of the processors in the column (say, the processor in the zero subcube).

Frieder makes use of three primitive communication patterns in order to efficiently perform binary relational operations and duplicate elimination within a subcube. *Tuple balancing* is a log(*k*) operation in which each processor in the subcube interacts with each of its subcube neighbors in turn, with neighbors exchanging tuples to correct any imbalances. Although a single iteration through the subcube dimensions does not guarantee a completely balanced tuple distribution, it should serve to correct most major imbalances. The second primitive operation, *merging*, similarly steps through the subcube dimensions, but this time neighbor pairs exchange tuples so as to cause each tuple to be present in both processors (that is, each processor ends up with the union of the two processors' prior tuple sets). The merging operation is carried as far as possible without overflowing any processor's local memory. The third operation, *cycling*, implements an intra-subcube communication pattern defined by a gray code on the column ID's. Thus, after 2^k steps in the cycling process, any particular data item will have visited all processors in the subcube.

Cycling is used as the basis of join and similar algorithms, where both relations are loaded into a subcube, and then one of the relations is cycled so as to allow all possible pairings. Tuple balancing and merging operations precede the cycling, the former as a means of dynamic load balancing, and the latter in order to reduce the cycle size (if merging can be carried through *j* steps, then a sub-subcube of *k-j* dimensions can be used to carry out the remainder of the operation).

4.5 The Role of the Control Processor

A valuable lesson was reported by the designers of DIRECT in [Boral 82] concerning the proper role of the control processor in a multiprocessor DBM. In the DIRECT design, it will be recalled, the control processor was made responsible for management of the paging memories. This meant that each query processor wishing access to a page was required to communicate its request to the control processor, which would then respond with a page frame number after loading the required page if necessary. The resulting traffic was found to be a major performance barrier.

These results were cited by the designers of GRACE as among their prime motivations in designing a dataflow style query execution mechanism with preallocation of resources. The result is that the entire query can proceed with no communication or other intervention required on the part of the control processor until the execution is complete.

The issue of control processor message traffic also contributed directly to the design of the GAMMA software system.[24] As was indicated earlier, each query executing on GAMMA is controlled by a scheduler process executing at one of the query nodes. The decision to place the scheduler processes in query nodes rather than the control processor was based on the relatively high level of interprocessor message traffic experienced by the scheduler process as compared to other processes executing in the control processor. Message processing in the GAMMA control processor was reported to be twice as expensive as in the node processors, due to the highly stripped-down operating system running in the

[24] Recall that GAMMA was designed by some of the researchers who designed DIRECT.

nodes (the control processor runs a full-blown Unix system).

4.6 The Teradata Y-Net

One final interesting point regarding interprocessor synchronization was reported by Teradata regarding their Y-Net processor interconnection [Shemer 84]. The Y-Net is a balanced binary tree interconnection network with the processors occupying only the leaf nodes. It was originally conceived as a mechanism to efficiently report results in a sorted order (not required by the relational model, but nevertheless required by many applications). The sorting function is achieved by means of a tournament-style sort implemented directly on the Y-net, where each node performs a merging operation on tuple streams received from its subtrees. It was noticed sometime after this design had been adopted that the Y-net could also serve as a general mechanism for implementing global synchronization. This is most easily seen by imagining that each processor generates a continuous stream of constant data, either 1 or 0 depending on whether or not the processor has reached the synchronization point in question. The control processor simply waits for the sorted tuple stream exiting from the top of the Y-Net to change from a stream of 0's to a stream of 1's in order to detect global synchronization. This is contrasted with the situation in a bus topology, where an individual message from each processor to the control processor would be required.

5. Conclusions

It is interesting to note that in some respects, database machine research has completed a full circle. Hardware went from general-purpose computers through exotic processors and finally back to collections of general-purpose computers. Memory systems went from hierarchical memories using RAM and disk technology with indexing software, through a barrage of exotic memory devices, and finally back to those same memory hierarchies with indexing, now duplicated among cooperating processors. Likewise, DBMS implementation moved from a software orientation, through a number of hardware orientations, and then back to a replicated software approach.

This is not to say that the intermediate designs were without merit and will not find useful application. In the first place, the effort served a learning function, and may have been the only route short of omniscience to what is now considered by many to be the most promising approach to database machines (the MBDS approach). Beyond this, it is not inconceivable that processing and memory technologies currently being explored, or not yet even conceived, will make certain of the early design elements extremely attractive. Some of the designs may find practical use in some specialized applications. For instance, large highly associative memories may be useful in situations where the high cost is justified by real-time needs of the application. Finally, some of the less exotic and less expensive designs have already found a commercial market, most notably the Britton-Lee device (IDM), but also including equipment available from Intel (iDBP). Low cost devices offering modest performance improvement appear to successfully fill an important practical need.

It is believed by this author that current research in the area of multi-backend database machines (MBDS) shows significant promise for satisfying, to a large extent, all of the motivational needs identified in the introduction. Such machines, utilizing proven technologies for the most part, will provide high performance, reliability, and flexible growth potential in both storage and computing capacities either as centralized network data utilities, or as participants in a network-based distributed database facilty. Although one hesitates to be overly optimistic concerning current DBM efforts in light of what can now be seen as a rush of premature optimism accompanying the early work, there appears to be evidence now supporting at least a more subdued optimism.

Bibliography

[Babb 79] E. Babb, "Implementing a Relational Database by Means of Specialized Hardware," *ACM Transactions on Database Systems*, 4(1):1-29, Mar 1979.

[Banerjee 78a] Jayanta Banerjee, David K. Hsiao, "Concepts and Capabilities of a Database Computer," *ACM Transactions on Database Systems*, 3(4):347-384, Dec 1978.

[Banerjee 78b] Jayanta Banerjee, David K. Hsiao, "Performance Study of a Database Machine in Supporting Relational Databases," in *Proceedings of the 4th International Conference on Very Large Data Bases*, West Berlin, pp. 319-329, Sep 1978.

[Banerjee 79] Jayanta Banerjee, David K. Hsiao, Krishnamurthi Kannan, "DBC--A Database Computer for Very Large Databases," *IEEE Transactions on Computers*, C28(6):414-429, Jun 1979.

[Batcher 80] Kenneth E. Batcher, "Architecture of a Massively Parallel Processor," in *Proceedings of the 7th Annual Symposium on Computer Architecture*, IEEE, pp. 168-173, May 1980.

[Batcher 85] Kenneth E. Batcher, "The Massively Parallel Processor System Overview," in J.L. Potter (ed.), The Massively Parallel Processor, MIT Press, Cambridge, MA, pp. 142-149, 1985.

[Baum 76] Richard I. Baum, David K. Hsiao, "Database Computers--A Step Towards Data Utilities," *IEEE Transactions on Computers*, C25(12):1254-1259, Dec 1976.

[Bernstein 87] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, MA, 1987.

[Berra 74] P. Bruce Berra, "Some Problems in Associative Processor Applications to Data Base Management," in *Proceedings of the National Computer Conference and Exhibition*, Chicago, pp. 1-5, May 1974.

[Berra 79] P. Bruce Berra, Ellen Oliver, "The Role of Associative Array Processors in Data Base Machine Architecture," *Computer*, 12(3):53-61, Mar 1979.

[Berra 87] P. Bruce Berra, Soon Myoung Chung, Nabil I. Hachem, "Computer Architecture for a Surrogate File to a Very Large Data/Knowledge Base," *Computer*, 20(3):25-32, Mar 1987.

[Boral 82] Haran Boral, David J. DeWitt, Dina Friedland, Nancy F. Jarrell, W. Kevin Wilkinson, "Implementation of the Database Machine DIRECT," *IEEE Transactions on Software Engineering*, SE8(6):533-543, Nov 1982.

[Boral 83] H. Boral, D.J. DeWitt, "Database Machines: An Idea Whose Time Has Passed? A Critique of the Future of Database Machines," in H.-O. Leilich, M. Missikoff (eds.), Database Machines, Springer-Verlag, Berlin, pp. 166-187, 1983.
Proceedings of the Third International Workshop on Database Machines, Munich, Sep 1983.

[Boyne 83] R.D. Boyne, "A Message-Oriented Implementation of a Multi-Backend Database System (MDBS)," in H.-O. Leilich, M. Missikoff (eds.), Database Machines, Springer-Verlag, Berlin, pp. 242-266, 1983.
Proceedings of the Third International Workshop on Database Machines, Munich, Sep 1983.

[Bray 79] Olin H. Bray, Harvey A. Freeman, Database Computers, Lexington Books, D.C. Heath and Company, Lexington, MA, 1979.

[Champine 79] G.A. Champine, "Trends in Data Base Processor Architecture," in *Proceedings of the 18th IEEE Computer Society International Conference*, San Francisco, pp. 69-71, Feb 1979.

[Codd 70] E.F. Codd, "A Relational Model for Large Shared Data Banks," *Communications of the ACM*, 13(6):377-387, June 1970.

[Copeland 73] George P. Copeland, Jr., G.J. Lipovski, Stanley Y.W. Su, "The Architecture of CASSM: A Cellular System for Non-Numeric Processing," in *Proceedings of the 1st Annual Symposium on Computer Architecture*, Gainesville, FL, pp. 121-128, Dec 1973.

[Coulouris 71] G.F. Coulouris, J.M. Evans, R.W. Mitchell, "A Hardware-Aided Approach to Content-Addressing in Data Bases," in *Proceedings of the 1971 IEEE Computer Society International Conference*, Boston, pp. 19-20, Sep 1971.

[Date 83] C.J. Date, An Introduction to Database Systems, Volume II, Addison-Wesley, Reading, MA, 1983.

[Date 86] C.J. Date, An Introduction to Database Systems, Volume I, Fourth Edition, Addison-Wesley, Reading, MA, 1986.

[**Demurjian 85**] Steven A. Demurjian, David K. Hsiao, "New Directions in Database-Systems Research and Development," in *Proceedings of the International Symposium on New Directions in Computing*, Trondheim, Norway, pp. 188-197, Aug 1985.

[**DeWitt 79**] David J. DeWitt, "DIRECT--A Multiprocessor Organization for Supporting Relational Database Management Systems," *IEEE Transactions on Computers*, C28(6):395-406, Jun 1979.

[**DeWitt 86**] David J. DeWitt, Robert H. Gerber, Goetz Graefe, Michael L. Heytens, Krishna B. Kumar, M. Muralikrishna, "GAMMA--A High Performance Dataflow Database Machine," in *Proceedings of the 12th International Conference on Very Large Data Bases*, Kyoto, pp. 228-237, Aug 1986.

[**Drawin 83**] M. Drawin, H. Schweppe, "A Performance Study on Host-Backend Communication," in H.-O. Leilich, M. Missikoff (eds.), Database Machines, Springer-Verlag, Berlin, pp. 135-153, 1983. *Proceedings of the Third International Workshop on Database Machines*, Munich, Sep 1983.

[**Epstein 81**] Robert Epstein, Louise Madrid, "The IDM 500--Communication Issues with Backend Processors," in *Proceedings of the 22nd IEEE Computer Society International Conference*, San Francisco, pp. 112-114, Feb 1981.

[**Flynn 72**] M.J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, C-21(9):948-960, Sep 1972.

[**Freeman 79**] Harvey A. Freman, "A Microprocessor-Based Design of the Data Base Computer," in *Proceedings of the 19th IEEE Computer Society International Conference*, Washington, DC, pp. 179-182, Sep 1979.

[**Frieder 87**] Ophir Frieder, "Issues In Query Processing on a Cube-Connected Multicomputer System," in *Proceedings of the 6th Annual Phoenix Conference on Computers and Communications*, Scottsdale, AZ, pp. 315-319, Feb 1987.

[**Fushimi 86**] Shinya Fushimi, Masaru Kitsuregawa, Hidehiko Tanaka, "An Overview of The System Software of A Parallel Relational Database Machine GRACE," in *Proceedings of the 12th International Conference on Very Large Data Bases*, Kyoto, pp. 209-219, Aug 1986.

[**Garcia-Molina 84**] Hector Garcia-Molina, Richard J. Lipton, Jacobo Valdes, "A Massive Memory Machine," *IEEE Transactions on Computers*, C33(5):391-399, May 1984.

[**Hawthorn 82**] Paula B. Hawthorn, David J. DeWitt, "Performance Analysis of Alternative Database Machine Architectures," *IEEE Transactions on Software Engineering*, SE8(1):61-75, Jan 1982.

[**He 83**] Xin-Gui He, Masanobu Higashida, Douglas S. Kerr, Ali Orooji, Zhong-Zhi Shi, Paula R. Strawser, David K. Hsiao, "The Implementation of a Multibackend Database System (MDBS): Part II--The Design of a Prototype MDBS," in David K. Hsiao (ed.), Advanced Database Machine Architecture, Prentice-Hall, Englewood Cliffs, NJ, pp. 327-385, 1983.

[**Hillyer 86**] Bruce K. Hillyer, David Elliot Shaw, Anil Nigam, "NON-VON's Performance on Certain Database Benchmarks," *IEEE Transactions on Software Engineering*, SE12(4):577-583, Apr 1986.

[**Hong 81**] Y.C. Hong, S.Y.W. Su, "Associative Hardware and Software Techniques for Integrity Control," *ACM Transactions on Database Systems*, 6(3):416-440, Sep 1981.

[**Hong 82**] Y.C. Hong, S.Y.W. Su, "A Protection Mechanism for Cellular Logic Devices," *IEEE Transactions on Software Engineering*, SE-8(6):583-596, Nov 1982.

[**Hsiao 79**] David K. Hsiao, "Data Base Machines Are Coming; Data Base Machines Are Coming!," *Computer*, 12(3):7-9, Mar 1979.

[**Hsiao 80**] David K. Hsiao, "Data Base Computers," in Marshall C. Yovits (ed.), Advances in Computers, vol. 19, Academic Press, New York, pp. 1-64, 1980.

[**Hsiao 83a**] David K. Hsiao (ed.), Advanced Database Machine Architecture, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[**Hsiao 83b**] David K. Hsiao, Douglas S. Kerr, Ali Orooji, Zhong-Zhi Shi, Paula R. Strawser, "The Implementation of a Multibackend Database System (MDBS): Part I--An Exercise in Database Software Engineering," in David K. Hsiao (ed.), Advanced Database Machine Architecture, Prentice-Hall, Englewood Cliffs, NJ, pp. 300-326, 1983.

[**Kamibayashi 82**] Noriyuki Kamibayashi, Kazuo Seo, "SPIRIT-III: An Advanced Relational Database Machine Introducing a Novel Data-Staging Architecture with Tuple Stream Filters to Preprocess Relational Algebra," in *Proceedings of the National Computer Conference*, Houston, TX, pp. 605-616, Jun 1982.

[Kiyoki 86] Yasushi Kiyoki, Kazuhiko Kato, Takashi Masuda, "A Relational Database Machine Based on Functional Programming Concepts," in *Proceedings of the Fall Joint Computer Conference*, Dallas, TX, pp. 969-978, Nov 1986.

[Korth 86] Henry F. Korth, Abraham Silberschatz, Database System Concepts, McGraw-Hill, New York, 1986.

[Lehman 84] Philip L. Lehman, "Systolic Arrays for Rapid Processing of Simple Database Transactions," *Ph.D. Thesis*, Department of Computer Science, Carnegie-Mellon University, May 1984. Avalailable as Technical Report CMU-CS-84-160.

[Leilich 78] H.-O. Leilich, G. Stiege, H.Ch. Zeidler, "A Search Processor for Data Base Management Systems," in *Proceedings of the 4th International Conference on Very Large Data Bases*, West Berlin, pp. 280-287, Sep 1978.

[Lin 76] C.S. Lin, D.C.P. Smith, J.M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," *ACM Transactions on Database Systems*, 1(1):53-65, Mar 1976.

[Lin 77] C.S. Lin, "Sorting With Associative Secondary Storage Devices," in *Proceedings of the National Computer Conference*, pp. 691-695, 1977.

[Lipovski 78] G.J. Lipovski, "Architectural Features of CASSM: A Context Addressed Segment Sequential Memory," in *Proceedings of the 5th Annual Symposium on Computer Architecture*, pp. 31-38, Apr 1978.

[McGregor 76] D.R. McGregor, K.G. Thomson, W.N. Dawson, "High Performance Hardware for Data Base Systems," in P.C. Lockemann (ed.), Systems for Large Data Bases, North-Holland, Amsterdam, pp. 103-116, 1976.
Proceedings of the 2nd International Conference on Very Large Databases, Brussels, Jul 1976.

[Moulder 73] Richard Moulder, "An Implementation of a Data Management System on an Associative Processor," in *Proceedings of the National Computer Conference and Exhibition*, New York City, pp. 171-177, Jun 1973.

[Murakami 84] K. Murakami, T. Kakuta, R. Onai, "Architectures and Hardware Systems: Parallel Inference Machine and Knowledge Base Machine," in *Proceedings of the International Conference on Fifth Generation Computer Systems 1984*, Tokyo, pp. 18-36, Nov 1984.
In session titled *ICOT Research and Development.*

[Murakami 85] Kunio Murakami, "Delta (Relational Database Machine)," *ICOT Journal*, 7:39-41, Mar 1985.

[Neches 85] Philip M. Neches, "The Anatomy of a Data Base Computer System," in *Proceedings of the 30th IEEE Computer Society International Conference*, San Francisco, pp. 252-254, Feb 1985.

[Neches 86] Philip M. Neches, "The Anatomy of a Data Base Computer System--Revisited," in *Proceedings of the 31st IEEE Computer Society International Conference*, San Francisco, pp. 374-377, Mar 1986.

[Ozkarahan 75] E.A. Ozkarahan, S.A. Schuster, K.C. Smith, "RAP--An Associative Processor for Data Base Management," in *Proceedings of the National Computer Conference*, Anaheim, CA, pp. 379-387, May 1975.

[Ozkarahan 86] Esen Ozkarahan, Database Machines and Database Management, Prentice-Hall, Englewood, NJ, 1986.

[Rudolph 72] J.A. Rudolph, "A Production Implementation of an Associative Array Processor--STARAN," in *Proceedings of the Fall Joint Computer Conference*, Las Vegas, NV, pp. 229-241, Nov 1972.

[Sakai 84] H. Sakai, K. Iwata, S. Kamiya, M. Abe, A. Tanaka, S. Shibayama, K. Murakami, "Design and Implementation of the Relational Database Engine," in *Proceedings of the International Conference on Fifth Generation Computer Systems 1984*, Tokyo, pp. 419-426, Nov 1984.

[Schuster 79] Stewart A. Schuster, H.B. Nguyen, Esen A. Ozkarahan, Kenneth C. Smith, "RAP.2--An Associative Processor for Databases and Its Applications," *IEEE Transactions on Computers*, C28(6):446-458, Jun 1979.

[Schweppe 83] H. Schweppe, H.Ch. Zeidler, W. Hell, H.-O. Leilich, G. Stiege, W. Teich, "RDBM--A Dedicated Multiprocessor System for Database Management," in David K. Hsiao (ed.), Advanced Database Machine Architecture, Prentice-Hall, Englewood Cliffs, NJ, pp. 36-86, 1983.

[**Shaw 79**] David Elliot Shaw, "A Hierarchical Associative Architecture for the Parallel Evaluation of Relational Algebraic Database Primitives," Technical Report CUCS-5-79, Columbia University Department of Computer Science, New York, NY, Oct 1979.

[**Shaw 81**] David Elliot Shaw, Salvatore J. Stolfo, Hussein Ibrahim, Bruce Hillyer, "The NON-VON Database Machine: An Overview," Technical Report CUCS-22-81, Columbia University Department of Computer Science, New York, NY, Sep 1981.

[**Shaw 8x**] David Elliot Shaw, "Relational Query Processing on the NON-VON Supercomputer," Technical Report (number unknown), Columbia University Department of Computer Science, New York, NY, 198x.

[**Shemer 84**] Jack Shemer, Phil Neches, "The Genesis of a Database Computer," *Computer,* 17(11):43-56, Nov 1984.

[**Shibayama 84**] S. Shibayama, T. Kakuta, N. Miyazaki, H. Yokota, K. Murakami, "Query Processing Flow on RDBM Delta's Functionally-Distributed Architecture," in *Proceedings of the International Conference on Fifth Generation Computer Systems 1984*, Tokyo, pp. 427-435, Nov 1984.

[**Slotnick 70**] D.L. Slotnick, "Logic per Track Devices," in Franz L. Alt, Morris Rubinoff (eds.), Advances in Computers, vol. 10, Academic Press, New York, pp. 291-296, 1970.

[**Su 77**] S.Y.W. Su, "Associative Programming in CASSM and Its Applications," in *Proceedings of the International Conference on Very Large Data Bases*, Tokyo, pp. 213-228, Oct 6-8, 1977.

[**Su 88**] Stanley Y.W. Su, Database Computers: Principles, Architectures, and Techniques, McGraw-Hill, New York, 1988.

[**Tanenbaum 81**] Andrew S. Tanenbaum, Computer Networks, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[**Ullman 82**] Jeffrey D. Ullman, Principles of Database Systems, Second Edition, Computer Science Press, Rockville, MD, 1982.