A PARALLEL MULTI-STAGE I/O ARCHITECTURE WITH SELF-MANAGING DISK CACHE FOR DATABASE MANAGEMENT APPLICATIONS

J. C. Browne, A. G. Dale,
C. Leung and R. Jenevein[1]

TR-84-34 November 1984

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

---

[1]Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX 78712.

A PARALLEL MULTI-STAGE I/O ARCHITECTURE WITH SELF-MANAGING DISK CACHE FOR DATABASE MANAGEMENT APPLICATIONS

Introduction

Parallel structuring of input/output processing is essential for the system architectures of future generations of very high performance computers for all types of applications. The speed of external storage devices has not kept pace with the increase in speed of processing elements. Parallel organization of I/O and computational support in the I/O subsystem provides one strategy for dealing with this mis-match.

We are currently studying an architecture designed to serve as an I/O engine for multiprocessor configurations of varied topology and style. The conceptual elements of this architecture are:

- An interconnection network to couple processors to external memory via a disk cache.
- A disk cache composed of self-managing cells that implement garbage collection, data filtering, content-based searching, and consistency management.
- Network switch/processor nodes that will execute indexing (packet routing) functions in the processor-to-memory direction, and operation specific functions in the memory-to-processor direction.

The system can be regarded as a parallel processing engine whose functions are external memory I/O, data and object management, and on-the-fly processing of data as it moves from external memory to host processors. It also provides a capability for integrating indexing and parallel search. Incorporation of indexing as a hardware function supports locality of storage and processing and, as necessary, independence of location of use and storage to provide an efficient general purpose external memory for high performance parallel architectures.

In this paper we discuss the major components of the architecture and how relational database processing may be mapped to its structure. We are also investigating the mapping of rule-based processing to the architecture, which will be reported in a forthcoming paper.

1. Architecture

The proposed architecture is diagrammed in Figure 1. It is organized into four major levels:

- Host processors, which for the purpose of this paper are assumed to be general purpose processors but might also be specialized processors such as Lisp machines.

- An interconnection network with a banyan topology. The switching nodes will be processing elements capable of routing requests and data through the network, and of executing application-dependent functions on data flowing through the network.

- Cellularly organized disk cache of self-managing secondary memories (SMSM's), capable of data filtering and internal garbage collection.

- Conventional moving-head disks, with a special purpose controller (SSMU) for one or more drives.

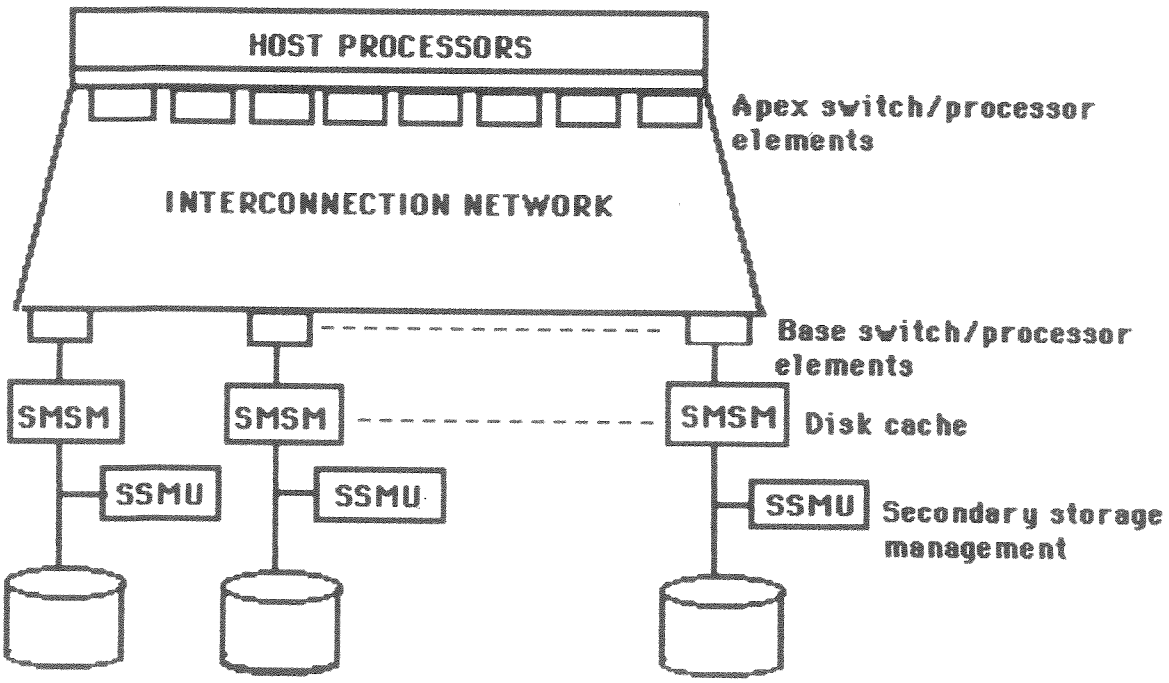

Figure 1.
Gross Architecture of Multi-stage I/O Processor.

The interconnection network and disk cache are discussed in more detail below.

Interconnection network (ICN)

The ICN consists of switch/function processors with local memory embedded in a banyan topology. The properties of banyan networks are summarized in Appendix I and may be referenced further in [GOKE73,76]. The ICN will support function domain mapping, such that a given process initiated by a host may specify functions to be executed in the ICN nodes supporting that process.

The network will support both packet switching and circuit switching under control of host processors.

A number of approaches may be taken to provide fault tolerance in the ICN. What is being proposed combines some of the attributes of X-Tree fault tolerance proposed by Despain and Patterson [DESP78] and that of Cherkassky, Opper and Malek [CHER84]. In the latter an additional redundant stage of the banyan was proposed for redundancy. This stage provides a means to connect to the equivalent node in an alternately rooted tree. The approach presented here provides this capability in an alternate manner.

At both the base and apex of the ICN there will be alternate paths between the replicated trees of the banyan. These alternate paths are in the form of a chain from one tree to the next and provide fault tolerant pathing should a switch/function node fail. Figure 2 shows switch S1-1 failed, thus preventing normal flow from S1-0 to I/O node 1. By using the alternate path to S3-0, S4-1 and S1-2, I/O node 1 can be accessed.

The nodes of the interconnection network will execute indexing (packet routing) functions in the processor to memory direction and operation specific functions (e.g. sort or merge) in the memory to processor direction. Thus both indexing and operation specific functions become highly parallel and distributed across the network. The network will also establish circuits between processors and memories. These circuits may map the RAM of the memory to be a portion of the address space of the processor or simply establish a fast channel between the processor and memory to bypass packet routing latencies in access.

Disk cache

The disk cache is organized into modules. Each module is termed an SMSM cell (Self Managing Secondary Memory cell), consisting of a cell controller and semiconductor memory. The architecture, memory organization, and performance characteristics of SMSM's of the type proposed for the disk cache are discussed in Appendix II, and are referenced in [RATH82,84]. An SMSM cell internal organization is shown in Figure 3.

SMSM cells are organized into clusters, with communication links between adjacent cells in a cluster. The configuration illustrated in Figure 4 assumes 4 SMSM cells per cluster, and one SMSM cluster per disk drive.

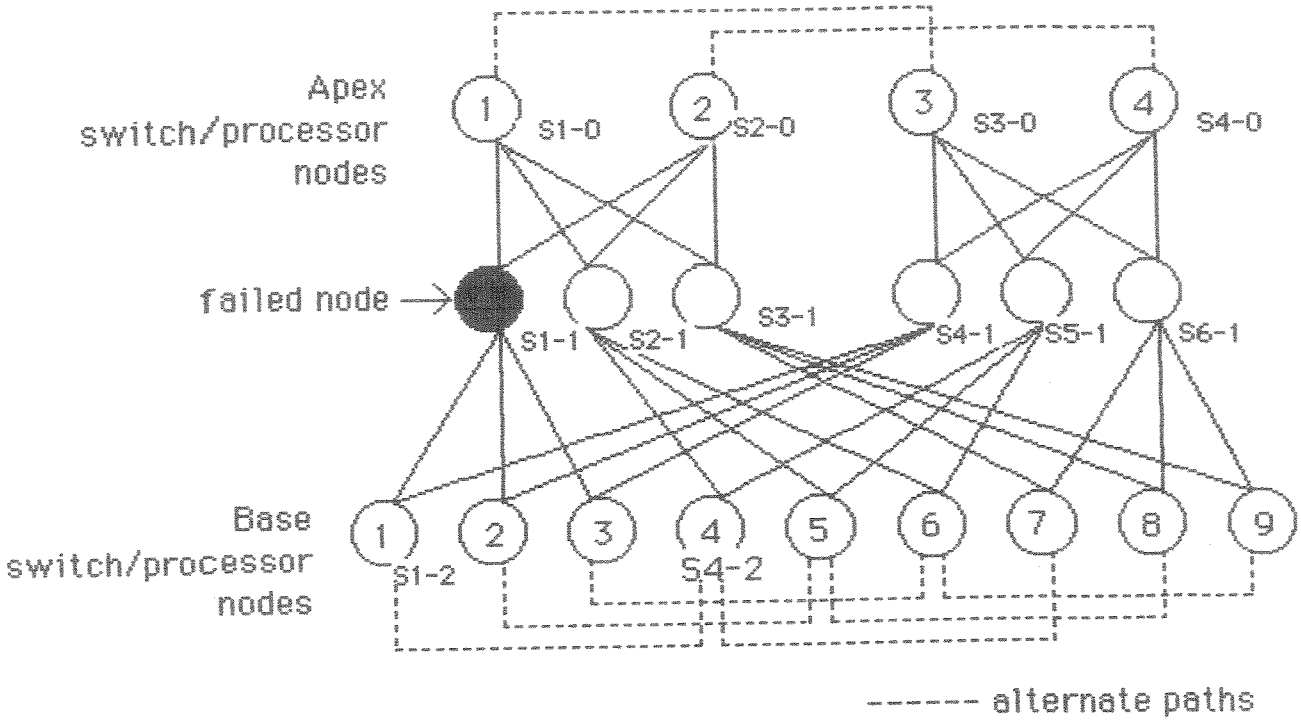

Figure 2.

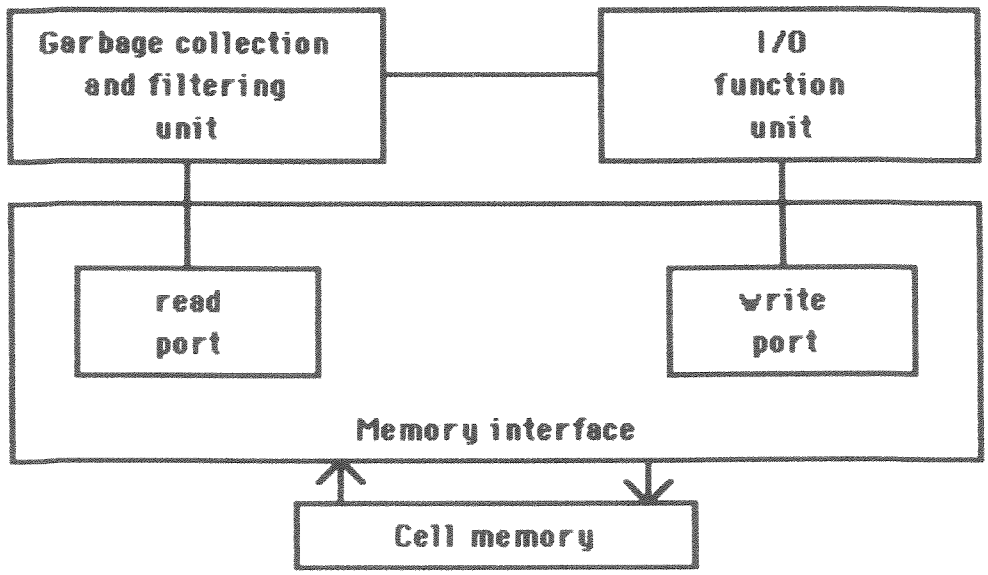

Figure 3.
SMSM Cell Controller Organization.

A cell priority ordering is defined within a cluster. Cell contents are searched in parallel. Cell I/O is sequential in priority order.

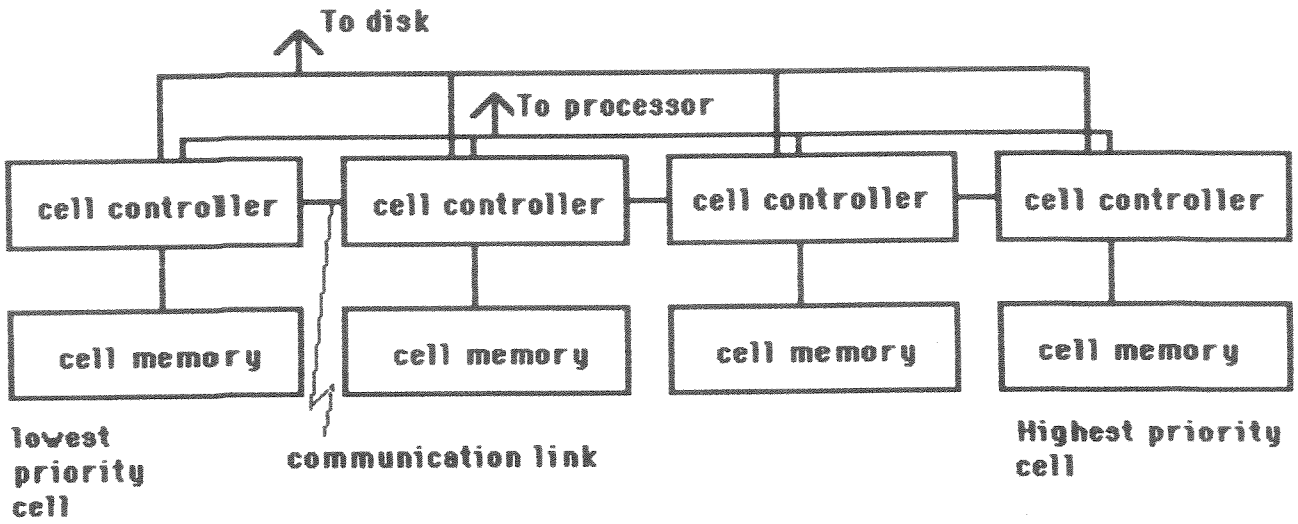

Figure 4.
SMSM Cell Cluster Organization.

Cells maintain directories to their internal contents. We propose a special controller (Secondary Storage Management Unit–SSMU) to manage data transfer between an SMSM cluster and disk. It will monitor request faults in the SMSM and initiate appropriate read/write transfers. It will maintain map entries for data blocks in the SMSM cluster containing information on the key, sector, and status of each block. The status word contains a clean/dirty bit which will be used in a standard manner to eliminate unnecessary transfers to/from disk. Writing to disk will only occur if the block status is dirty and SMSM space is needed (inward bound control).

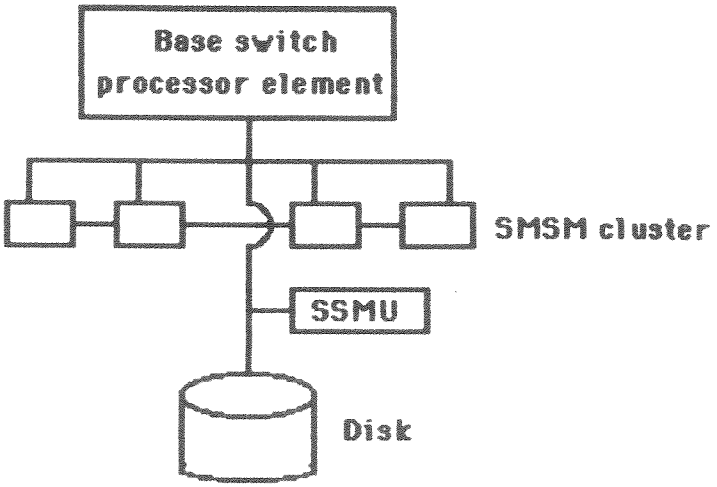

Figure 5.
SMSM-SSMU Organization.

Shadowing and redundancy can be achieved at the base level of the I/O network

relatively inexpensively by introducing bypass switches between adjacent SMSM clusters and alternate paths to disk, as shown in Figure 6.

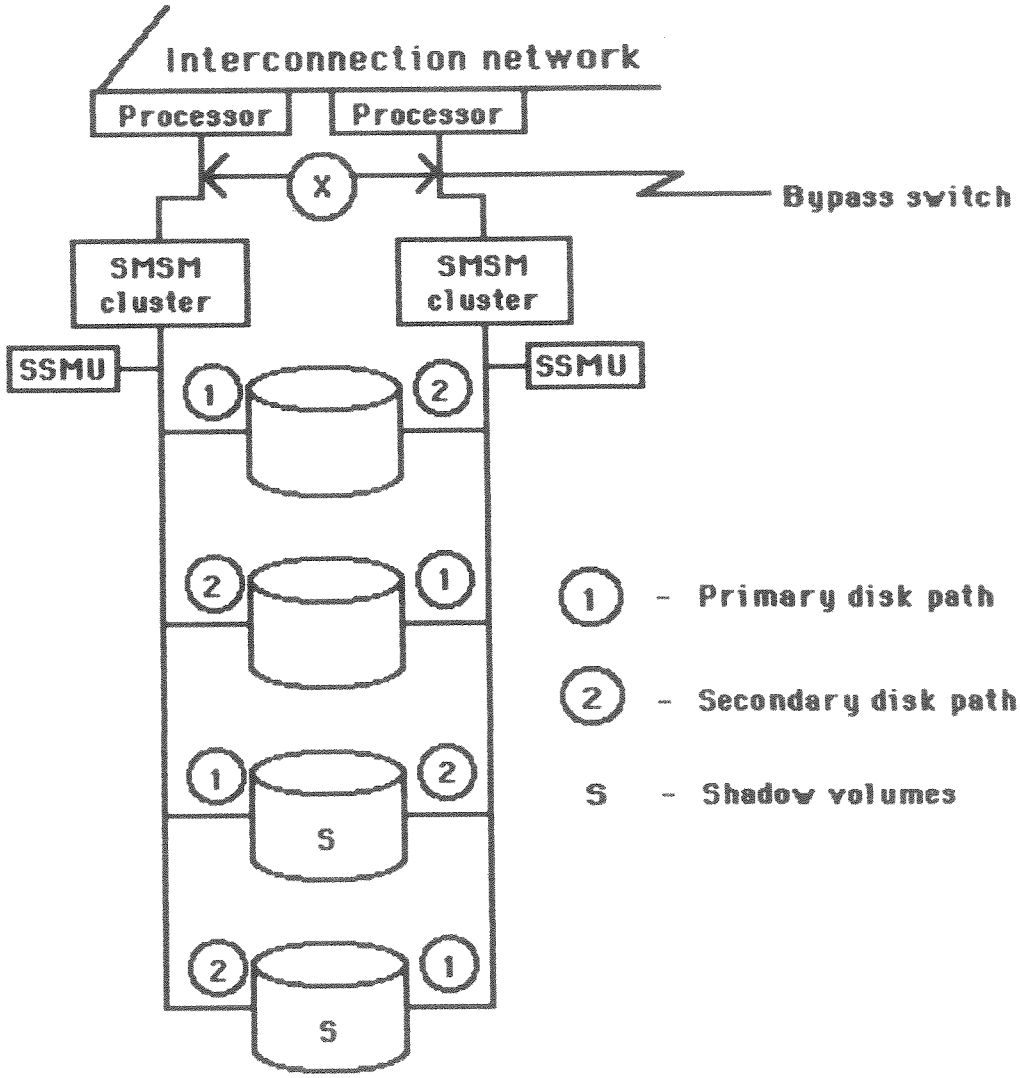

Figure 6.
Fault Tolerance for SMSM-SSMU-Disk Modules.

The processing capabilities associated with the base nodes will also implement selection and filtering operations on data elements before passing them to the network for transmission and possible further processing. These filtering operations will execute in conformance to parameters supplied by the processors initiating the retrieval request.

Summary

The proposed architecture provides a framework for realizing the following

functionality:

- Extremely high bandwidth from/to secondary storage, because of parallelism in disk and SMSM access.
- Content searching at the level of the disk cache.
- Data filtering at the level of the disk cache.
- Garbage collection and memory compaction within the SMSM's.
- Parallel data manipulation (e.g. sorting, merging, aggregate operations) within the interconnection network as data flows towards the host processor(s).
- Fault tolerance within the network, the cache, and the disk subsystems.
- Object-oriented referencing by the host machines.
- Packet and circuit switching under host machine control for efficient utilization of network bandwidth.
- Hardware based indexing within the network and the SMSM's.
- Hardware-implemented concurrency control at the cache level.
- Functional partitioning of the I/O system under host machine control.

The following section describes how relational database processing may be mapped to the architecture.

2. Relational Database System Access and Operations

For the purpose of illustration assume that relations are distributed across disk storage in logical cylinders. That is, relations are partitioned into approximately equal chunks of tuples such that a set of tuples for a given relation resides in a physical disk cylinder (or adjacent group of cylinders) on each drive. Note that an attribute-based schema could be organized in a similar manner. In the case of a relational database application, and assuming a tuple-based schema, query processing selection and projection operations will be done on-the-fly at the level of the SMSM's as data flows into the I/O network. Base level processor-switch nodes are responsible for duplicate elimination (in the case of SMSM projection operations), attribute permutation, and sorting and merging of tuples delivered by SMSM cells in the cluster under its control. Higher level nodes are responsible for further merging of sorted tuple streams delivered by lower level processor/switch nodes. Multi-relational operations (e.g. join) are performed at the host

level, by final merge operations.

Assuming a tuple-based schema and logical cylinder organization, a host processor is responsible for query decomposition and broadcast of selection and projection commands for the SMSM's. In the case of tuple addition the host processor can establish a network routing to store tuples in a selected drive (or SMSM cluster). Other update operations (tuple deletion, value modification) would be broadcast and implemented at the level of the SMSM's.

We believe that the proposed I/O architecture addresses the critical problem in the processing of very large conventional databases, namely the improvement in I/O bandwidth. Boral and DeWitt [BORA83] make a convincing argument that this, rather than the engineering of a special-purpose database engine, should be the focus for current research. Their proposals for further investigation include exploiting the logical cylinder concept, design of customized disk controllers, use of a very large disk cache, and more effective index strategies to map from logical to physical space. The paper also argues that under foreseeable development we must still rely on conventional moving head disk technology for large secondary storage.

Our proposed architecture differs from other current developments in the database machine area. The Japanese ICOT database machine effort [SHIB84] incorporates a special-purpose relational algebra engine and a hierarchical memory subsystem incorporating a 128 Mbyte semiconductor RAM cache, with conventional moving head disks as secondary storage. However, the organization of the memory subsystem differs significantly from our design. For example, the disk cache is controlled by a single general purpose processor. Overall, the Japanese design puts major emphasis on the relational engine component of the architecture, whereas our proposed architecture seeks to exploit I/O parallelism and distributed on-the-fly support in the I/O subsystem for algebraic operations in a general purpose host processor or processors.

Some of the secondary storage access strategies proposed in earlier architectures (e.g. DBC) can be adapted to our design, although our implementation is significantly different. The philosophy of the Munich group [KIES 84] also is consistent with our view of the importance of intelligent disk subsystems for high performance database architectures and the need to offload operations to the I/O system that can be efficiently processed at that level. However, our proposed implementation and distribution of functionality within the I/O subsystem differs from this reported work.

The proposed architecture has some of the properties and functionality of the Hypertree configuration proposed by Goodman [GOOD 81]. The Goodman architecture has a binary tree topology with additional regular connections among nodes at the same level. Goodman studied in detail the possibilities of exploiting a variety of distributed algorithms to support relational database processing in the Hypertree structure.

Although our architecture differs in interconnection topology and in the incorporation of the self-managing disk cache, many of Goodman's conclusions supporting the desirability of a tree structured processor organization for relational database access and operations are applicable.

A backend machine architecture of interconnected processing nodes has been proposed by Shultz [SHUL 81]. However, the REPT architecture is oriented to the execution of relational algebraic operations by the processing nodes, and the mapping of a query decomposition by the host processor to the network of processors. It thus differs in philosophy from our proposed approach.

Other proposed or experimental systems discussed in the literature incorporate some of the features specified for our architecture, but differ in completeness, functionality and architecture. The VERSO system being developed at INRIA is a bus-organized arrangement of functionally specialized processors, including a user interface processor, filter processor, and memory management processor. The importance of low level filtering is recognized. But the concept differs significantly from our proposal in that parallelism and hierarchical functional processing is not exploited [BANC 83].

The SABRE proposal, based on mapping virtual processes to a variety of architectures, will be implemented by a bus-organized sequence of processing modules with a disk cache memory. The possibility for significant parallelism is not evident in the brief description given [GARD 83].

RDBM, the University of Brunswick project, intends to exploit on-the-fly filtering and partial sort-merge operations to support join processing, an important feature in our design. However, the architecture is composed of a number of specialized processors; shared memory is assumed for part of the system. The processor interconnection method is not yet specified, although the possibility of some type of bus is mentioned [SCHW 83].

DBMAC recognizes the importance of filtering and of exploiting parallel disk transfer to increase I/O bandwidth. However, the design as discussed in [MISS 83] does not incorporate a disk cache, and assumes bus-connected processor units at one level, with special purpose selection processors interfaced with disk storage.

In summary, evidence from the recent literature supports the potential viability of our gross architecture for relational database query processing. Our current investigations are directed to the resolution of the following questions as a preliminary to detailed design synthesis and evaluation:

- Identification of required functionality to support relational algebraic operations, intra- and inter-query parallelism, and consistency management.

- Distribution of functionality across the architecture.
- Capacity balance across the architecture with respect to processing modules and network bandwidth.
- Identification of most appropriate routing/broadcast algorithms for supporting I/O requests.

Performance evaluation studies should provide answers to these questions and allow for intelligent decisions concerning tradeoffs.

3. References

[BANC 83] Bancilhon, F. *et al*, "VERSO: A Relational Backend Database Machine," in David K. Hsiao (Ed.), *Advanced Database Machine Architecture*, Prentice-Hall, 1983, 1-18.

[BORA 83] Boral, H. and DeWitt, D., "Database Machines: An Idea Whose Time has Passed?" *Third International Workshop on Database Machines*, October 1983.

[CHER 84] Cherkassky, V., Opper, E. and Malek, M., "Reliability and Fault Diagnosis Analysis of Fault-Tolerant Multistage Interconnection Networks," *Proc. of the 14th International Conference on Fault-Tolerant Computing*, pp. 246-251, 1984.

[DESP 78] Despain, A. M. and Patterson, D. A., "X-Tree: A Tree Structured Multi-Processor Computer Architecture," *Proceedings of the 5th Annual Symposium on Computer Architecture*, pp. 144-151, 1978.

[GARD 83] Gardarin G. *et al*, "SABRE: A Relational Database System for a Multiprocessor Machine," in David K. Hsiao (Ed.), *Advanced Database Machine Architecture*, Prentice-Hall, 1983, 19-35.

[GOKE 73] Goke, L. R. and Lipovski, G. J., "Banyan Networks for Partitioning Multiprocessor Systems," *Proceedings of the First Annual Symp. on Computer Architecture*, pp. 21-28, 1973.

[GOKE 76] Goke, L. R., "Banyan Networks for Partitioning Multiprocessor Systems," Ph.D. Dissertation, University of Florida, 1976.

[GOOD 81] Goodman, J. R., "An Investigation of Multiprocessor Structures and Algorithms for Data Base Management," Ph.D. Dissertation, Univer-

sity of California, 1981.

[KIES 84] Kiessling, W., "Tuneable Dynamic Filter Algorithms for High Performance Database Systems," Institut fur Informatik, Technische Universitat Munchen, 1984.

[MISS 83] Missikoff, M. and Terranova, M., "The Architecture of a Relational Database Computer Known as DBMAC," in David K. Hsiao (Ed.), *Advanced Database Machine Architecture*, Prentice-Hall, 1983, 87-108.

[RATH 82] Rathi, Bharat Deep, "Principles of Operation of TRAC's Self-Managing Secondary Memory," Technical Report *TRAC-25*, Departments of Computer Sciences and Electrical Engineering, The University of Texas at Austin, 1981.

[RATH 84] Rathi, Bharat Deep, "The Design and Performance Analysis of a Self Managing Secondary Memory," Technical Report *TR-84-09*, March 1984, Department of Computer Sciences, University of Texas at Austin.

[SHIB 84] Shibayama S. *et al*, "A Relational Database Machine with Large Semiconductor Disk and Hardware Relational Algebra Processor," *New Generation Computing* 2, 1984.

[SCHW 83] Schweppe, H. *et al*, "RDBM—A Dedicated Multiprocessor System for Database Management," in David K. Hsiao (Ed.), *Advanced Database Machine Architecture*, Prentice-Hall, 1983, 36-86.

[SHUL 81] Shultz, R., "A Multiprocessor Computer Architecture for Database Support," Ph.D. Dissertation, Iowa State University, 1981.

Appendix I

Banyan Networks

Banyans are one of a large class of graph representations currently being explored for use in Multi-Stage Interconnection Networks(MINs). The structure of a banyan is essentially an overlay of tree structures and provides full connectivity between apex(roots) and base(leaves) without redundancy. Regular and rectangular banyans are special cases of L-level banyans in which restrictions are placed on the indegree and outdegree of the vertices of the graph [GOKE 73].

A banyan is a Hasse diagram of a partial ordering in which there is one and only one path from any base to any apex. An apex is defined as any vertex with no arcs incident into it, a base is is defined as any vertex with no arcs incident out of it, all other vertices are called intermediates.

The path from any base to any apex is called the base-apex path. A subbanyan is a subgraph of a banyan which is itself a banyan. It follows that in a banyan there is at most one path from any vertex to any other vertex.

An L-level banyan is a banyan in which all base-apex paths are of length L. It is an (L+1)-partite graph in which the partitions may be linearly ordered from 0 to L such that arcs exist only from the i-th partition to (i+1)-th partition (0 <= i < L). The vertices in the i-th partition are called vertices at level i and referred to as Vi.

A regular banyan is an L-level banyan in which the indegree of every vertex except apexes is s and the outdegree of every vertex except bases is f. An (s,f,L) regular banyan has (f**(L-i)$_*$**i) vertices at level i, and (f**(L-i) $_*$ s**(i+1)) arcs from vertices in level i to those in level i+1 (0 <= i <= L). From the above, an (s,f,L) regular banyan has s**L apexes and f**L bases. A (2,3,2) regular banyan is shown in Figure A1.

A rectangular banyan is one in which s=f, hence there are s**L vertices at each level and s**(L+1) arcs from each level to the next. If we let s=f=d then these banyans can be described as a (d,L) rectangular banyan.

These definitions and properties, presented here are primarily due to the work of Goke described in his Ph.D. dissertation [GOKE 76].

Properties of banyan networks have been extensively studied at the University of Texas at Austin, and a prototype has been implemented as part of the Texas Reconfigurable Array Computer (TRAC) at the University.

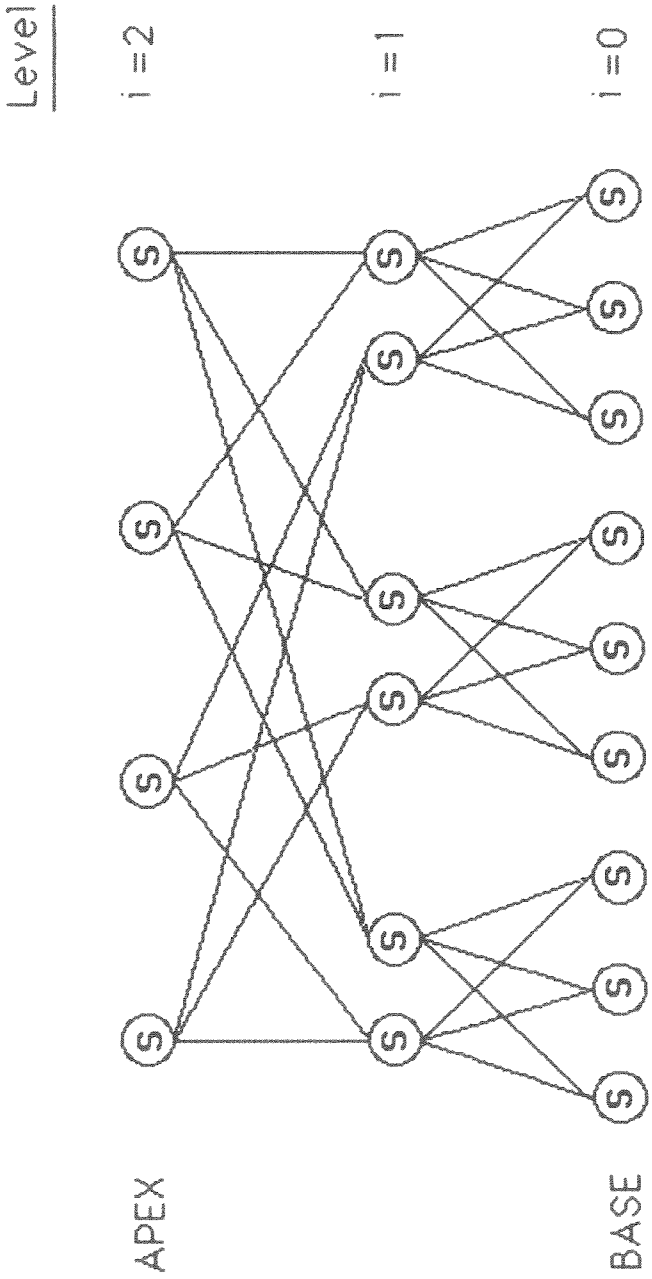

Figure A 1: (2, 3, 2) Banyan Interconnection Network

Appendix II

Rathi Self-Managing Secondary Memory

Rathi has reported a number of possible approaches to the design of a self-managing secondary memory (SMSM) system [RATH 82, RATH 84]. He points out that the evolution in semiconductor technology in terms of both price and performance has widened the access time gap between primary and secondary memory. As processor performance increases and demands of array processor structures are addressed, the gap can only widen even more. The work, in which he reports on the design and performance of an SMSM, will reduce this gap between primary and secondary memory systems. He has included the following memory technologies in his study:

1. Static and Dynamic RAMs

2. CCD

3. Bubble Memory

4. Fixed Head Drum and Disk

5. Moving Head Disk

6. Magnetic Tape Cartridge

He further reports that by adding controlling hardware, based on SIMD processors, a modularly expandable SMSM can be designed. His modular expansion implies that doubling the memory capacity can result in either none or insignificant increases in latency or retrieval time, as long as one doubles the number of controlling SIMD processors. Within the SMSM architecture all information is accessed via a name-oriented associative address scheme. This means the host has no need for directory information which may be kept within the SMSM controlling processor. The SMSM also supports both memory allocation and garbage collection, hence again the host need not concern itself with these primitive support functions. They are the domain of the SMSM controllers as they should be if high performance is to be achieved by the secondary memory system. The reported architecture of the SMSM supports variable length segment storage and retrieval. This capability makes it attractive as a paging device for multiprocessor systems, distributed systems and object-oriented architectures.

The fundamental objective of the SMSM design is the development of a cache memory between primary memory and the electromechanical permanent storage devices. A valid approach will provide a cost effective solution for the reduction of the access time gap between these two memories. Rathi reports three possible ways to achieve this:

1. The use of intermediate semiconductor memory

2. Reduce the software overhead for access and maintenance

3. Use of SIMD, logic-per-track and tagged architecture concepts

The design criterion reported here has the potential of applying all three of these factors and achieving an extremely effective solution to the problem. There is enough flexibility in his methods that either high performance may be obtained or a less costly solution at the sacrifice of some performance.

SMSM Structure

The SMSM is organized into modules, each of which is called an SMSM cell. The cell can be further divided into two components, the cell controller and the cell memory. The processor sees the SMSM as a single module, while internally it may consist of one or several cells. When more than one cell exists there are communication links between them which provide for the necessary coordination. There exists a fixed left to right priority ordering defined in the hardware and conveyed via the inter-cell communication links. Each cell controller is responsible for communicating data between it and the host and, in addition, communicating control information with the host and its neighboring cells' controllers. For garbage collection purposes it may be necessary for a cell to communicate data between itself and neighboring controllers. Thus each cell controller can perform all SMSM functions and is the intelligence of the memory management hardware. From this structure it should be clear that the addition of more SMSM cells should not substantially increase access time to data.

Segments and Garbage Collection

Information is stored in segments within the cell memory. These segments consist of two parts, a fixed length header and variable length data. The type of information in memory is controlled by tags. Upon initialization all memory is written with EOF tags, then as segments are written, the tag is changed to HD or DT to imply header or data parts of the segment. A segment may overlap to another cell memory which will require coordination during both read and write operations. This split segment is completely handled within the SMSM cell controllers. It was mentioned earlier that the SMSM supports garbage collection in the hardware. This is accomplished by using a fourth tag "GB" to imply that the word is garbage to be handled by the collection routine. Garbage collection occurs "on the fly" and is continuously running. It will pick up GB tagged words and "float" them to the end of memory where they become EOF tags. When garbage words are created and no EOF tags exist in this controller, it will ask the next lower priority controller to transfer data to it. The amount of data to be transferred will be equal to the number of garbage words in its cell memory.

Cell Memory Organization

The choice of memory technology will affect the cost/performance of the system. Rathi has related the connection between these memory technologies and factors such as random versus sequential access, and fixed versus variable access speed of memory. He describes three basic memory organizations for the cell. The first of these is circular organization. Here all access is sequential in a fixed direction. There is a starting address and an ending address such that after reaching the ending address the starting address is then used. Those memory technologies which are inherently circular such as rotating disk or drum, CCD and magnetic bubble, are ideal for this organization. Data is usually bit wise serial in these structures. A second type is linear organization in which each cell as before contains a starting and ending address. Unlike the circular organization above, contents may be randomly accessed as whole words. This, of course, implies that only semiconductor RAM memory technology is suitable for this organization. Still a third type of organization consists of a data cell memory with a separate directory memory. The primary purpose behind this approach is the improvement of segment search speed. The data memory stores segments and the directory memory stores the segment headers and their corresponding data memory address. The directory is used only by the cell controller and is not accessible by the host processor, so name-oriented accessing is preserved. The directory only improves performance, the SMSM/Host interface remains as described above.

Search Strategy

The search strategy on the SMSM is divided into two phases. In phase-1 each cell searches its memory to locate the required segment and, if found, informs all other controllers (via GIC signal). An LRNC signal between controllers is used to decide the order in which the cells are activated. If the required segment is not found, the highest priority controller terminates the operation and sets an error condition in the status register. Phase-2 is responsible for performing the appropriate action, read/write/modified under the command of the host. At a given point in time only one cell will be active since it contains the data required at that instant in time.

Phase-1 Algorithms

The first algorithm presented, SCIR, assumes that the cell's memory is circularly organized, hence any memory technology is applicable to it. All cells start the search at the same time and try to match the required segment name with the names in the headers from memory. Two algorithms are presented, a general form for multiple occurrences of the segment name and a restricted form for single occurrences (only one segment with that name). The latter is more efficient and could be used for virtual paging. It should be noted that when only one occurrence of a segment is possible, the transfer could start immediately (phase-2 algorithm). A small scratch pad memory (SCRPAD) is required for holding the addresses of segments when multiple matches may occur. Both rely on an "on the fly" compare between the required segment name and the name in the segment header coming from memory. Let "m" be the number of

words in the cell's memory and "tc" be the controller clock cycle time and Tsp1 be the phase-1 search time, then

$$Tsp1 = m_{*}tc$$

If a search occurs for a single segment then m in the above equation may be substituted for $0.5_{*}m$ since the search will terminate as soon as a match occurs.

The next two phase-1 algorithms require directory support. The first of these is a directory hash approach (DHASH). The data memory is assumed to be a linear organization as described earlier. The directory memory is indexed via the hashing algorithm operating on a segment name. Since the directory entry contains the data cell memory address and linear memory is randomly accessible, the retrieval of the segment is quite fast. If there is more than one occurrence of the segment name, a linked list is formed from the first entry (link pointer) at the hash index address in the directory memory. If more than one segment is hashed into the same index, then a linked list entry will also be appropriate. An obvious restriction of this algorithm is the requirement for semiconductor RAM memory for both data and directory memory. Let "Ld" be the cell's average link list length, then the average time taken for this algorithm to execute is

$$Tsp1 = Ld_{*}tc$$

The third phase-1 algorithm (DASSM) uses an associative memory to implement the directory. The segment name in the header is used for the associative match. There are control tags in each directory word and if a match occurs a tag is set in that word. This technique also allows more than one occurrence of a segment name. Performance should be quite good, but it does require an associative memory structure. Let "n" be the average number of segments in a cell with the same header and "k" be the number of cycles to do an associative match, then the time taken by this algorithm is

$$Tsp1 = (k + n)_{*}tc$$

Phase-2 Algorithms

The first two phase-2 algorithms (SFS and SVS) are for circularly organized memory. They assume that memory can only be accessed sequentially. the SFS algorithm assumes a memory technology capable of only one speed of operation, while SVS assumes the memory is capable of more than one speed. The advantage of SVS is that it can advance at high speed until the segment address is reached then slow down for transfer to/from the host. This latter capability can only be used with semiconductor RAM, CCD and magnetic tape cartridge, since others are not capable of the speed variation.

The last phase-2 algorithm (SRAND) assumes linear organization of cell memory. This limits the application to RAM memory, but provides for a fast access to the cell address in order to start the transfer to/from the host. No overhead time is accumu-

lated in locating the segment, so the time required is the segment read time.

Garbage Collection and Memory Allocation

Rathi describes three basic garbage collection algorithms for circular memory organization. In order of decreasing execution time they are: RGC, LAGC and Imp-LAGC. It was found that for linear organization since garbage collection was inherently sequential within a cell, it was treated as a circular memory. Any algorithm for circular memory organization is suitable for linear memory. When directory memory is involved, both memories must have garbage collection units which communicate with each other.

Memory allocation is much less complex than garbage collection because all available memory (i.e. EOF tagged words) words are at one end of the overall SMSM memory. Since more than one memory may have available space, the highest priority cell executes the memory allocation algorithm first. When its memory is exhausted the next priority cell's memory is allocated. If no lower priority cell exists, then the status register is written with the error condition. Two parameters control the memory allocation in each cell, a memory available flag and an up/down counter. The counter points to the available memory address. The garbage collection algorithm can set the flag and decrement the counter. The memory allocation algorithm can clear the flag and increment the counter. The flag would be cleared whenever the counter reached the ending address of the cell memory.

Summary

The structure and design seem versatile enough to span all current memory technologies. Rathi reports that he did not address optical memory because it was read only. The algorithms allow for conservative, less costly designs or high performance, more costly designs. A key to its success is the inclusion of garbage collection in the hardware. The fact that the collection algorithms will run on all memory organizations make them very powerful additions to the approach. Finally, having a name-oriented associative address scheme frees the host and its operating system of the laborious task of maintaining directories and allocating sectors.