PARALLEL DATABASE MACHINES

Kjell Bratbergsengen
Department of Computer Systems and Telematics
The Norwegian Institute of Technology
University of Trondheim
Email: kjellb@idt.unit.no

Summary

The idea of database computers is more than 25 years old. During the first 15 years of this period much research was devoted to designing and building special purpose hardware for database computers. The results were never really promising; too expensive, questionable performance, lack of standardization and little market acceptance. It is easy to see, in retrospect that we did not know the right algorithms for parallel execution of database operations at that time. That situation improved in the mid eighties, and parallel execution methods are common knowledge today.

Parallel technology is fairly well accepted today. We are also including RAID technology to this area. The new challenges are to develop fault tolerant systems and database servers for “new” data types, notably film and video.

THE TRAUMATIC HISTORY OF DATABASE COMPUTERS

The concept of database computers has been known for many years. Research on database or back end computers were reported early in the seventies.The actual architectures have taken many directions. They can be classified into 5 categories.

1) Intelligent secondary storage devices. The typical solution was to put a processing element for each track or cell of data. This solution could give a tremendous search capability, however the problem was cost. A typical cell was a track on magnetic disk, or track (ring) of a bubble memory or CCD (Charged Coupled Device).

2) Filters. This solution did not require modification of the storage device itself. The filter processor was put into the controller. This is a more economical solution, however the capacity is less than for intelligent devices. This type of product has had some commercial success. The CAFS - Content Addressable File Storage has been marketed by ICL for many years.

3) Associative memory systems. Instead of increasing the search capacity of the permanent storage or searching data on the fly, this solution is based on fast handling of data after they are brought into memory. It requires special memories with built in logic for comparing data and setting off triggers on match. The problem is cost, it is hard to afford large enough memories. The more well known systems in this group are STARAN, NON-VON and RELACS.

4) Multiprocessor database computers. This group is by far the largest. The database computers with a certain commercial succes: the IDM from Britton-Lee, DBC/1012 from Teradata Co. and the Tandem Non Stop Systems are in this group. So are many of the more

Research on parallel systems in the Database Technology Group at The Department of Computer Systems and Telematics, NTH has been supported by NFR formerly NTNF, Intel Norway AS, Eltron AS and Norwegian Telecom.

known research database computers. We will mention Gamma and Direct built by professor David DeWitt’s group at University of Wisconsin, Grace built by a research group led by professor Kitsuregawa at University of Tokyo, SABRE built by a group headed by P. Valduriez and G. Gardarin of INRIA in France and later, European Community supported developments. Also the massively parallel search system based on the Connection Machine must be put into this group. However, this system is different since the parallel computer is of the SIMD-type.

5) Text processors. This group contains computer architectures specially designed for handling text, like searching strings and substrings, and complex software systems for free text searching and manipulation.

The more promising approaches are found in category 4, multiprocessor database computers. Our own work on database computers also falls into this group. A comprehensive history of the development of database computers, and a thorough description of the different methods, algorithms and architectures are found in [Su88]. The above classification is also taken from Stanley Su.

However, the above classification may not be complete. During recent years, several experimental systems storing the database in main memory have emerged. Four main memory database systems [Lela87], [Eich87], [NaMi87], [KerS87] was presented at the Fifth International Conference on Database Machines held in Karuizawa in October 1987. The rationale for main memory database machines is obvious: availability and cost of memory chips, now and in the future.

Also object oriented database systems will take great advantages of larger main memories.

Database computers are not dead, however ...

We have been through a period where a lot of special purpose hardware has been developed. The most remarkable evidence from this research, was that none of these designs ever made it to the market. Also the technical achievements were disappointing. The results were so negative that David DeWitt stated that “database computers are dead” [DeWi83] (Database Computers, an Idea Whose Time Has Passed) . The paper “proved” that it was not possible to get performance improvements beyond 4 nodes in a parallel system for doing algebra operations. Luckily, both David himself and others, later - proved this to be wrong.

Practically all parallel database computer developments where aimed at improving the performance of relational operations. These operations are demanding, and involve the handling of large data volumes. The emphasise on special purpose hardware for sorting and joining was wrong. What really needed improvements was the understanding of, and algorithms for - how to do relational operations in parallel. In the mid eighties the hash based algorithms for doing relational algebra were published. [Bra84], [DeWi85] and [Bra90]. These algorithms are very well suited for parallel execution and after this period we got tremendous improvements of the performance test results of relational database systems.

The hash based methods do not require special purpose hardware. They require a general parallel computer where each node is an ordinary complete computer. However, the communication system between the nodes has to be efficient. Round trip delay for short messages should be small and transfer capacity for large data volumes should be high. These requirements are common to most parallel computers, but very hard to satisfy.

The methods developed in the mid eighties have not been used in commercial systems until quite recently. Hash based parallel relational algebra algorithms now seems to be used in the Oracle/7 database system.

Database computers are not dead, however database computers built from very special purpose hardware have not proved to be viable.

New System Structures Opens up for Specialized Database Computers

The development of storage systems technology, communication systems and the general proliferation of PCs and workstations increase our ability to store and disseminate large amounts of data. At the same time - the usage and number of actual and potential users is also increasing rapidly. This is a driving force behind the development of new systems, new requirements, new functions.

The database community has so far only offered good solutions to a small segment of the users needs. The database technology has concentrated on formatted data, the register file type of data. Though important, these data constitutes only a fraction of the information to be stored and handled in a modern information system. The other data types which has attracted the lesser focus are:

- free text
- geometric data (product descriptions, cartographic information)
- images
- moving pictures and sound
- rules

There is a need for improved quality of service and extreme performance, for instance in telecommunication switching systems. Using database technology as a means for routing calls is more economic, modifiable, flexible and service enabling than the old hard wired routing techniques. The reliability and performance requirements are then dramatically increased compared to for instance, the database in a library information system. The telephone switching network should never be unavailable, the response to any question should come within a few milliseconds.

Technology Push

CPUs and memories have been steadily improving in speed and capacity over the years. The picture is more mixed for secondary storage media. The capacity has been steadily increasing, but the access times and transfer times has been nearly constant. Magnetic disks are the most

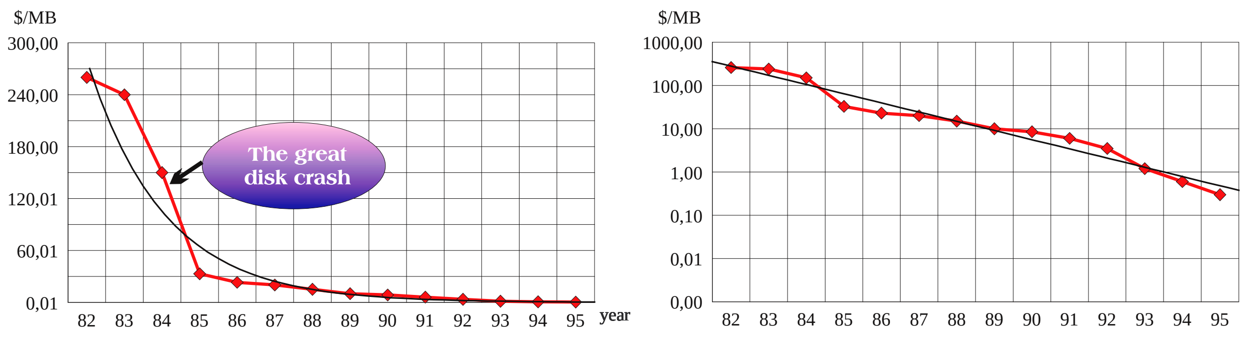

Fig. 1 Disk price in $ per MB. The great disk crash happened in 1984 and 1985. From the right diagram we can see that we had a minor disk crash recently. The price has dropped by a factor of 10 during the last three years. The disk price in 1995 is about 0.30 $ per MB. The left diagram shows price pr. Mb in linear scale, to the right in logarithmic scale.

important storage media for I/O intensive use. Other interesting media are optical and magneto optical disks, but their transfer capacity is still far too low.

As the storage capacity of magnetic disks has improved impressively, the price per stored unit has fallen correspondingly. Figure 1 shows the actual price in $/MB over the last 14 years. The price per MB has fallen from 200 $/MB in 1982, to nearly 0.2 $/MB today - a difference by a factor of 1000. Modern disks are virtually maintenance free, MTBF has increased from 30000 hours to 1000000 hours. They come in small packages and requires very little power, 10 to 25 Watt per disk.

1 GB disks are produced by a number of manufacturers, 2 and 4 GB by quite a few, and Seagate Technology has recently released a 9 GB disk. This opens up for a broad range of new applications where storage cost had prevented on line storage.

However, the transfer capacity has not improved too much. Large disks have transferred 3 MB/s for more than 10 years. Smaller disks usually had a sustained transfer rate between 1 and 2 MB/s. The faster disks today transfer from 3 to 6 MB/s. To overcome the I/O bottle neck, *disk array* technology was developed. RAID - Redundant Arrays of Inexpensive Disks was first described in 1988 [PaGiKa88]. RAID technology improves raw I/O capacity, however it is not the best solution when many accesses are required.

Research on Parallel Database Systems at NTH

To take advantage of the less expensive, powerful and readily available system components we have to build parallel systems. Our group has worked on parallel methods for database systems since 1977. Since 1984 we have built 5 experimental parallel computers.

The nodes has been based on Intel processors. The internal communication network connecting the nodes has been realized by dual ported RAM in an all to all topology or in a hypercubic topology.

The number of nodes has been 3, 8, 16, 16 and 64. All nodes have been equipped with one or

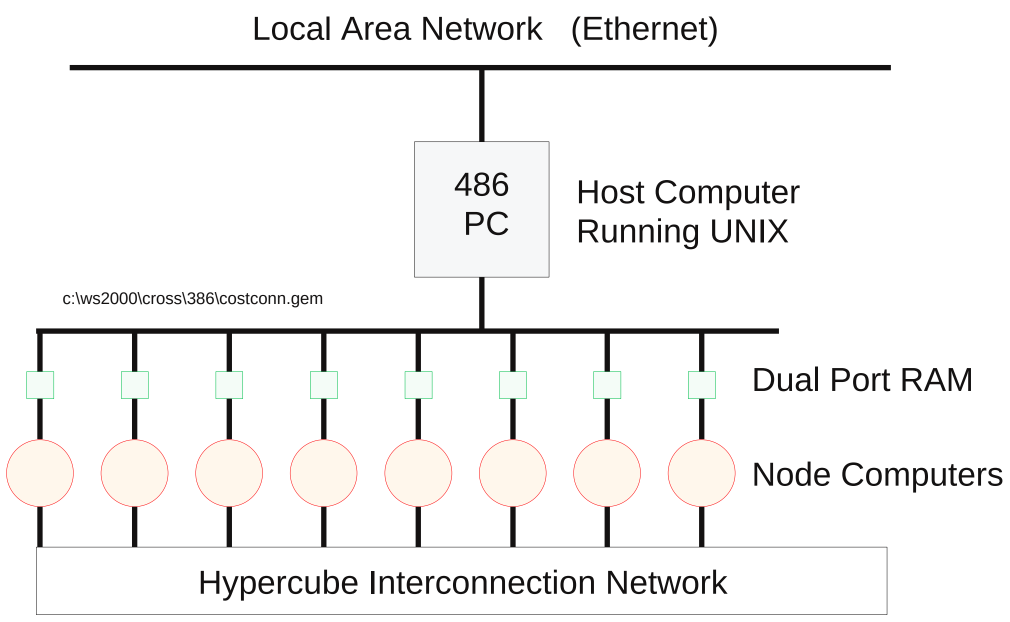
Local Area Network (Ethernet)
486 PC
Host Computer Running UNIX
c:\ws2000\cross\386\costconn.gem
Dual Port RAM
Node Computers
Hypercube Interconnection Network

Fig. 2 The structure of the experimental parallel computers developed by the Database Technology Group, IDT.

two SCSI ports. Each node is a complete computer and it is controlled by an ordinary operating system. Our first four parallel machines used DOS, the last machine is running UNIX.

Classification of Computer Systems

We are classifying computers in three groups: monoprocessor systems, multi processor - shared memory systems and distributed memory systems.

The monoprocessor system has one of everything; processor, bus and memory. The shared memory system has one bus and one memory, but multiple processors. The performance is limited by the maximum capacity of the bus or the memory system. The distributed memory systems consists of a number of independent computers of the first two classes connected together through a communication network. The nodes can be identical or different.

From a programming point of view, the great difference is between the distributed memory systems and the other two groups. In a distributed memory system; data and commands have to be transferred over a communication network if these are meant for a process located on a different node. This is adding a lot of new problems to the programming of the system: How should data and programs be distributed to minimize network traffic? How much delay is caused by the network? Will the network be congested? And so on.

It is easy to port a program from a monoprocessor system to a multiprocessor, shared memory system. It is not possible to directly port a database system to a distributed memory systems. Major rewriting is necessary. It is relatively easy to build distributed memory parallel computers and a lot of platforms are available today. However, the need for redesign of the programs and algorithms is the reason why there are so few commercial systems available on parallel platforms.

The potential for “unlimited capacity” and the best potential for building reliable and robust systems are the major advantages of distributed memory systems. This is the reason we have concentrated our research around this class of platforms.

The Interconnection Network

The interconnection network is a crucial part in a distributed memory system. The network should be nonblocking, deadlock free, affordable, scalable, symmetric and have low latency and high bandwidth.

A number of realizations are available: bus, ring, mesh, cube, hypercube, complete connection (all to all) switches.

It is not possible to meet all requirements in one system, one has to compromise. We have used a hypercubic topology in our larger systems - 16 nodes and above. The physical network topology should be hidden to the programs. Programs should be designed as a set of cooperating processes independent of the number of physical nodes and the network topology.

As faster commercial communication networks are emerging, the need for special communication hardware is reduced. It should be a good balance between the systems disk I/O capacity and the relocation capacity.

The communication is done by message passing. It is hard to avoid time losses in

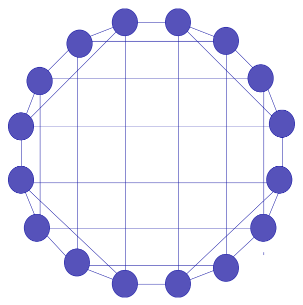
Fig. 3 A 4- dimensional hypercubic network.

message passing. Two parameters are of interest, transfer capacity and message delay. Message delay is most important in tightly cooperating processes. Transfer capacity is needed when we have to relocated large volumes of data. Delay is particularly hard to avoid because no matter how fast the message system is, handling messages at the receiving end requires context shift. Software overhead is hard to avoid.

EXAMPLES OF PARALLEL SOLUTIONS TO DATABASE APPLICATIONS

We will now walk through some examples of the exploitation of parallism in database applications.

The Centralized Large Data Banks

Is it a new golden era ahead for large centralized data banks? In my opinion the answer is yes. Reduced cost of high performance mass storage systems will increase the demand for large centralized data banks.

Todays library systems contain only information about books and magazines. The real information we are searching for is not available on electronic media. One reason is storage cost, another is the limited telecommunication capacity. Both these limiting factors are about to be relaxed.

Optical storage technology is still the least expensive direct access medium for large data volumes. Price pr. megabyte on optical disk varies from 30 cents down to 3 cents. When we compare this with magnetic disk technology, keep in mind that this is only media cost. Drives and auto changers must be added to make up the total cost.

The real difference between magnetic disks and optical disks is speed. While optical disks can deliver 0.3-0.6 MB/s, the magnetic disk can deliver 6 MB/s. In time critical systems this makes up the whole difference. One video disk can serve one movie. One magnetic disk can at the same time serve many video viewers, all at different places in the movie.

These types of data are so huge that it is not economically reasonable to store copies at several different places. There is no need either, modern communication networks facilitates access from all over the world. This can draw a huge number of users, and processing capacity could be a limiting factor. In this situation, parallel technology will help out. Through good design, systems can be built to meet the demand requested. Parallel technology is also used to improve reliability and availability. This can be designed to several levels, without redundancy, portions of the database will be unavailable when certain modules fail. If redundancy is employed, almost 100% availability can be obtained.

Lower storage cost, no absolute limit on processing capacity and good availability all favors the centralized parallel server for demanding tasks. Not to mention the potential for improved quality and reduced cost for maintenance and operations.

Complex Queries in Large Data Volumes

These problems has been investigated over several years and good methods has been developed. This has also been our major area of research. The methods are scalable and it is possible to design systems to any requirements within reasonable limits. There are three possible bottlenecks: Relocation capacity, disk I/O capacity and CPU capacity.

To give an impression of the performance level, we present the following example: Give 1KB of data to every Norwegian (4 millions). With a 64 node parallel computer of the same type we have developed in our laboratory; we are able to answer *any question* on these data within 7 seconds.

Relational algebra is the basic instruction set in all relational database systems. Relational algebra is well suited for parallel execution. We are also developing special relational algebra operations for handling geometric information. i.e. cartographic data.

The fundamental operation in relational algebra is to find records with some common properties, i.e. having identical values in given fields. Which fields will vary from operation to operation. Our methods are based on the fact that if two keys are equal, their hash values are also equal. At the outset, operand records are spread out on the disks of the parallel computer. All nodes are supposed to be similar, and the strategy for distributing records is to use a hash function on the primary key of the record. This will give a uniform distribution of records over the nodes, which is important for good load distribution.

To do a join $R = A \otimes B$ we redistribute the operands A and B, such that candidate matching records are stored at the same node. To do a join we have to read the operands from their local disk, compute target node, send record to target node and complete the join operation locally at each target node. All operations are done in parallel: disk reading, relocation, local join and writing result records to the disk locally. At the same time this algorithm is inherently distributing the load. The records can be unevenly distributed, but rehashing records to their target node will improve load distribution.

The bottlenecks of this algorithm may be in the disk reading, the relocation or in the CPU doing local join and general administration. An international set of tests, the DeWitt benchmark, has been developed to measure the performance of different hardware systems and different algorithm [BiDeTu83]. The DeWitt join test is joining two tables A and B. A contains 1000 records and B 10000 records. All records are 180 bytes long. All records in A will find exactly one matching record in B, hence the result table will contain 1000 records.

We have done several experiments on different relational algebra realizations. The first where reported in [Bra87]. The DeWitt join test took 1.8 seconds. Then the second in 1989 in [BraGje89]. Now, the join test time was reduced to 0.8 seconds. Later the scaling of our algorithms has been tested in [San92]. The join test has been reduced to about 0.35 seconds, but the more interesting result is that the algorithms gives an almost perfect scaling with the number of processors. We might design a parallel computer which is able to answer the most complex question within a certain time by adding the necessary number of nodes.

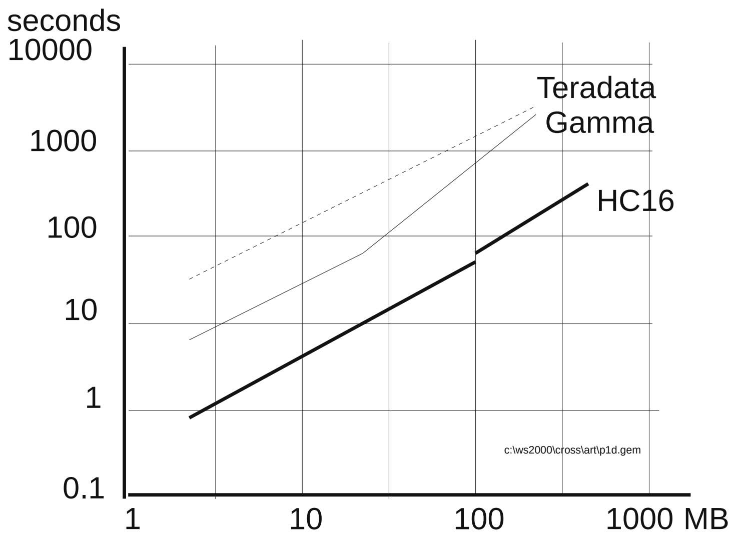

Fig. 4 Join of A and B. B is 10 times larger than A. Operation time in seconds as a function of total operand volume.

All our tests show that join is a piecewise linear function of operand volume. As long as the smallest operand is small enough to be stored entirely in main memory, the operation takes $O(A+B+R)$ seconds. For larger operands, the operation takes $O(3(A+B)+R)$ seconds. This holds when relocation takes less time than disk I/O. Whether this is true or not depend on the relative aggregated capacity of the communication system and the disk I/O system.

Parallel Sorting

Sorting is a classical application. Both sorting and relational algebra are limited by I/O transfer capacity. Fast CPUs combined with for example wide RAID systems would be simpler to program and provide the required capacity limited by the I/O system or the CPU.

We have tested two methods for parallel sorting.

Parallel Quick Sort

Quick sort is a recursive or repeated bisectioning sort which is particularly well suited for parallel execution on a hypercube system. It starts with one group of unsorted records. This group is then partitioned into two groups: records with keys less than the partitioning value and records with the key value greater than or equal to the partitioning value. The ideal partitioning value is dividing the group in two groups of half the size of the original group. For each new group this process is repeated until the group size is one record (or until all records in the group have the same key value). The parallel implementation is straight forward. The first division is along the most significant dimension, then the next lower dimension is used, and so on. At the end, the nodes will contain separate groups in sorted order.

The extra cost of parallel quick sort is for finding a partitioning value for each group, which involves communication between nodes in each group. The other contribution to extra cost is the transfer of records to their correct node. For each partitioning, about half the records will be moved. The total volume is *V* and the number of hypercube dimensions is *D*, then the transferred volume is $V_{rel} = 0.5DV$. Data under relocation are read from disk and written to disk for each step (dimension). It is faster to find partitioning values and the correct records to relocate, if all records are at least partially ordered prior to relocation. This partial presort will require one pass of the data volume. After data is relocated, each node contains a number of ordered files. The final sort is one pass of these files. The volume of data transported between disk and memory is then: $V_{file} = 2(2V + 0.5V_{rel}) = 2(2V + 0.5DV) = V(4 + D)$ Depending on the relative transfer capacity of the disk I/O system and the interconnection system either internode transfer or disk I/O will be the bottleneck. However, both operations can overlap.

Parallel Direct Relocation Sort

The writing and reading of intermediate files is killing the parallel quick sort for large data volumes. Baugstø and Greipsland [BauGre89] developed a parallel sorting method where records were directly relocated to the target node. To be able to do this, the partitioning key values between nodes (in sorted order) must be known prior to relocation. This is done by a sample sort. A small fraction of the records, 1 - 2 % is sampled at each node and sorted using parallel quick sort. This volume should be kept so small that intermediate file storage is unnecessary. After the sample parallel quick sort, the partitioning values are in each node. These are then broadcast to all nodes and relocation can start.

Incoming records at the target node are partially sorted in a reservoir sort. The sort is accomplished through a local merge. Volume of data transferred between disk and main memory is $V_{file} = 4V$ if merge is performed in one pass. Almost all data are relocated, only $\frac{V}{2^D}$ are at the right node from the beginning.

This last method was better, and fig. 5 shows sorting times in seconds measured at our first hypercubic parallel computer. It had 16 nodes, each with an Intel 186 (1 MIPS) processor, 1 MB RAM and a 30 ms access time disk. It was able to sort about 3 MB/s. The aggregated disk I/O capacity was about 16 MB/s. This computer had a relatively weak CPU, and it is seen from the curves that for shorter records this process was heavily CPU-bound. Each byte is transferred between disk and main memory 4 times. We would be I/O-limited if we are trying to sort more than $\frac{16\text{MB}/\text{s}}{4} = 4\text{MB}/\text{s}$.

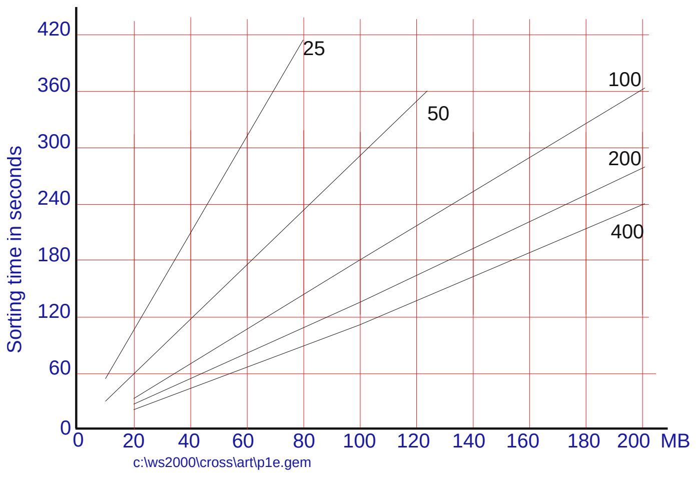

Fig. 5 Sorting time as a function of volume and record size. From [BauGre89]

Transaction Processing

General Remarks

A transaction consists of a number of operations. If these operations are independent they can all be performed in parallel. If there is a dependency, this will force a sequential execution or a partial sequential execution of the operations.

Example:

- read from input values a,b,c,d;
- update a in table A;
- update b in table B;
- update c in table C;
- store a,b,c,d in table LOG;
- commit transaction;

The three update operations and the store operation are all independant and can be executed in parallel within the transaction. One of the nodes is coordinator and supervisor. Each operation reports success or failure back to the coordinator. Failure is reported when resource conflicts make it impossible to continue - a deadlock situation. The actual operation is performed by an agent process at the node where the data are stored. At this node resource conflicts are also checked, if so - a further check for deadlock is necessary. Deadlock detection in a parallel system is quite similar to deadlock detection in distributed systems. There is no global view of the situation at one single node, unless we send all information about resource conflicts to this node. If we do so, it is easily seen that this node is a potential bottleneck. A parallel system can reduce the response time for each transaction and the total system throughput.

A Debit - Credit Type Example

In a transaction system, normally the I/O volume is small, but the number of transfers is large. Let us use the debit credit benchmark as an example. One transaction is updating three registers and writing to a fourth. The registers are of different size, one is pretty small, so it is probably most of the time in main memory buffers saving I/O accesses. However, including accesses to the log, it is hard to come below 5 I/O accesses per transaction. It is very simple to compute the minimum number of disks required to meet a certain number of transactions per second. Let the average access take 25 ms, then each disk will take up to 40 accesses or 8 transactions per second. A 1000 TPS system will need at least 125 disks.

Such a system was designed by our group for the Norwegian Telecom in 1990. It was also required to be fault tolerant which nearly doubled the number of accesses. The on line database was specified to be 500 GB which at that time required about 250 disks. One transaction (fault tolerant) takes at least 1 million instructions. To process 1000 TPS you simply need a 1000 MIPS system. Our design ended up with a 64 node hypercube where every node had a 20 MIPS processor. This gave the necessary number of MIPS and with 4 disks on every node, both storage and I/O capacity were comfortably met.

“The Double Spending Checker”

Recently we got a challenging problem of the same type. A database is holding 3×10^9 records, each 200 bytes long giving a total of 600 GB of data. Every day 10^8 10^8 new records are inserted and the same number is purged from the database. The average insertion rate is 2300 per second, maximum is 3000 per second. On insertion, key duplication is checked, the database should never hold two records with identical keys.

The key values are random and for each insertion we have to read the database and write the record back. This system might be looked upon as a gigantic randomized file being able to do at least 6000 accesses per second. As already stated, one disk is able to do 40 accesses per second. In this system we need 150 disks. This is also in complete harmony with the number of disks needed to store the database. Using 4 GB disks we need exactly 150 to store 600 GB. As this is a randomized file we cannot pack data 100%.We will need at least 10% more for extra space.

Can this run on a 64 node parallel computer? Yes and probably several configurations are possible: Use 16 nodes for external communication and 48 nodes for storing the database. The records are stored using a hash function, this determines node, disk and block number. The communication node will do the following:

- receive message,
- compute block number (node, disk, block)
- send record to target node
- receive record from target node
- send acknowledge to user.

Each communication node will have to handle 188 transactions or receiving and sending about 800 messages per second. The net volume received is only 40 KB. This is retransmitted to the target database node through the parallel computer interconnection system. This is less than 1 MB per second and should cause no problem. Each database node will have 4 disks which gives a total of 192. The number of accesses per node is approximately 125. This is an average of 30 per disk which is no problem.

We should also check the I/O transfer capacity. If block size is 4 KB, the aggregated I/O volume is 0.5 MB/s. This is well below even the slowest SCSI transfer mode of 1.5 MB/s. This means we can increase block size to increase the random file block factor. 4KB blocks will take up to 20 records 200 bytes. This is satisfactorily, it is not necessary to increase the block size.

The cost of this system, disks and programs excluded will be about 500000 $ which probably is affordable, performance and capacity compared.

Film and Video Servers

This is both a transfer challenge and a volume challenge. Depending on coding and quality, each client will need from 0.2 MB/s to 6 MB/s continuously over minutes and hours. Assume the following scenario: We are storing lectures in a database. The students have access to work stations where they will have access to course material including lectures. They are free to play lectures on demand. The film server will have to serve several hundred viewers and at the same time recording new lectures. One hour lecture at the lowest quality video amounts to 720 MB at a storage cost of about 5000 NOK. Due to the amount of storage required in this system a tertiary storage system is necessary. This be juke boxes for optical disks or tape stackers. Transferring data between secondary and tertiary storage is the third activity going on simultaneously.

The aggregated sustained I/O capacity is only about 30 MB/s with 100 simultaneous users. However, they are all reading (a few is writing) at different addresses. Then it is much better to have several independent disks rather than one powerful RAID. This is demonstrated with the following simple calculations.

Each viewer is given a 100 KB buffer. This holds for 0.5 seconds of video. We have two alternatives for storing the film:

1) A number of RAID systems with 20 ms access time and 20 MB/s transfer rate.

2) A number of disks with 20 ms access time and 3 MB/s transfer rate.

How many RAIDs and disks do we need?

One RAID system is able to serve: $n_R = \dfrac{0.5}{0.02 + \dfrac{0.1}{20}} = 20$ users.

We need 5 RAID systems to serve 100 users.

One disk is able to serve: $n_d = \dfrac{0.5}{0.02 + \dfrac{0.1}{3}} \approx 10$ users.

We need 10 disks to serve 100 users. It is obvious that 10 disks are much less expensive than 5 RAID systems.

Oracle and Silicon Graphics are now announcing “media servers”. Oracle is running on a hypercube connected, distributed memory platform, the nCube, which is designed for a maximum of 8182 nodes. They claim it would be able to serve 30000 viewers simultanously. This means that 6GB of data is continously flowing through the interconnection network.

PACS

We are working on PACS - Picture Archiving and Communication Systems for hospitals. The characteristics of these applications are huge data volumes and instantaneous retrieval on demand. The regional hospital in Trondheim produces 4TB of image data each year. By Norwegian law, the images have to be available for 10 years. Even when data compression is employed, one year worth of data is 1 TB.

In the screening phase, the digital system should be as good as the manual presentation system. 8 images with a quality equivalent to 2000×2000 points should come up in less than 0.5 seconds. This is again equivalent to a data transfer rate of 64 MB/s. The transfer rate is itself a challenge. Both the disk I/O transfer rate and the communication transfer rate will be reduced by using data

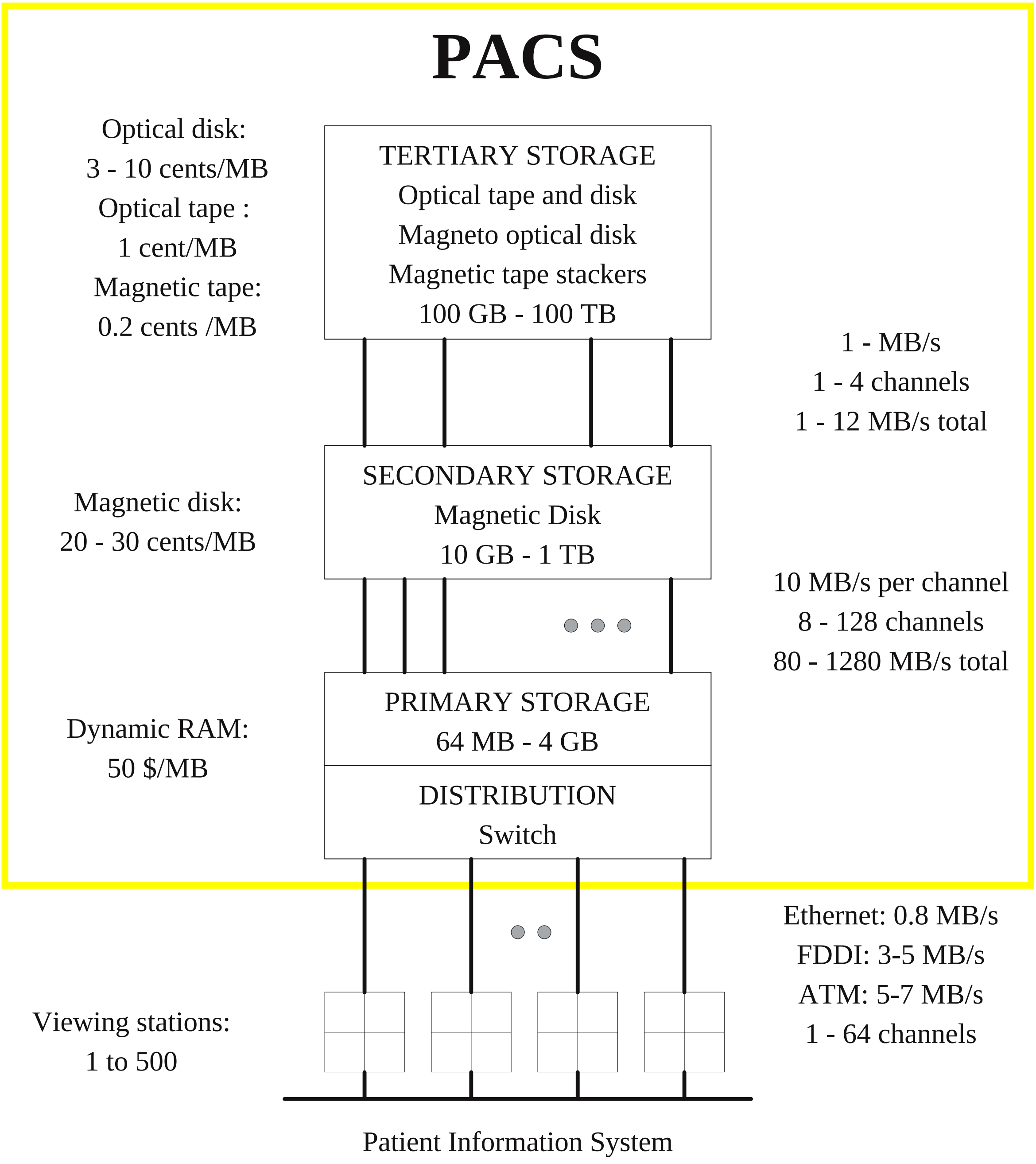

Fig. 6 PACS - Picture Archiving and Communication System . The image server is within the gray rectangle.

compression and decompression. Still we will have to handle 20 MB/s.

Looking at the image server, it should be dimensioned to handle several viewing stations simultaneously. It must also be able to do image compression, storing of new images and staging. These activities will have a lower priority than retrieval. Staging is the process of moving data between secondary and tertiary storage. Most of the data to be retrieved will be on secondary storage. Using magnetic disks as the secondary storage medium ensures fast retrieval. The image access time is further reduced by employing software striping. Retrieval time is determined by the number of disks and the number of channels used for storing one picture. One should balance the volume against transfer capacity by choosing disk size. One could always get the necessary number of disks by using smaller units. Storing 50 GB on secondary storage, using 2 GB disks, gives 25 units. They have an accumulated transfer rate of 100 MB/s - which is more than enough.

Retrieval of Data from Large Homogenous Data Volumes

Large raster images are found in several applications: topographic and tematic maps, satellite images, seismic data, etc. We will analyze the storage of and retrieval from three dimensional raster images. These may be the results from a seismic survey. To visualize the results, each density is represented by a certain color. From the indexes, we can transform back to the real world coordinates.

The methods are especially suited for fast retrieval, used in advanced image viewing systems.

How fast data is needed, depends on window size, window depth and “flying speed”. With window depth, we mean how far into “the colored cloud” we should be able to look. This is however, not influencing the required transfer rate. If our window is 500×500 points, and we fly through all frames using 25 new frames every second, we need 6.25 MB/s net transfer rate.

We should store the data in a way such that they are available at approximately the same rate in all directions. This is achieved by storing data in small cubes. To retrieve one cube, we must do an access, followed by a transfer. We can not rely on sequential reading of the disk. The effective retrieval rate is much less than the theoretical transfer rate for sequential transfer.

One question in this design is: How large should we make every cube? To get an idea we will compute the optimal size of a cube for retrieving a body: $V = w^2 l$. The body is a prism, both edges are *w* and the length is *l*. *l* is much larger than *w*. The prism is located anywhere in the “color cloud”, and the prism is not aligned with the cubes. The edge of the cube is *x*, the volume of each cube is $v = x^3$. To get the net volume *V*, we have to transfer some surrounding voxels, the actual volume to transfer is:

$$\tilde{V} = (w + x)^2 (l - x)$$

The actual transfer time is:

$$T = \frac{t\tilde{V}}{x^3}$$

if *t* is the time for transferring one cube.

We might express *t* as a function of *x*: $t = a + bx^3$, where *a* is the access time, and *b* is the transfer time per byte. We substitute for *t* and $\tilde{V}$ and get this expression for *T*:

$$T = \frac{(a+bx^3)(w+x)^2(l+x)}{x^3}.$$

To find an optimal *x* we will derive *T* with respect to *x*. We find an optimal *x* for:

$$2bx^5 + 2bwx^4 - ax^2 - 4awx - 3aw^2 = 0.$$

This equation must be solved numerically. We have also simplified the expression by letting

$l \approx l + x$, assuming $l \rangle\rangle x$.

To find a solution we let $w = 500$, $a = 25$ ms and $b = \frac{1000}{90 \times 30000}$.

The expression for *b* is based on 5400 rotations per second and that the disk has 30000 bytes on each track. For this case the optimal *x* is 81. The total time for reading all data while *l* = 2000 is 293 seconds. To get an impression of the importance of the cube size we have computed *T* for several *x* values.

Cube size, number of voxels in each dimension										
x	10	20	30	40	50	60	70	80	90	100
T (seconds)	13263	1909	739	452	353	314	298	293	294	299

Using too small cubes is detrimental to performance. On the other side, using large cubes requires a lot of memory. For the optimal cube, each cube is about 0.6 MB. To cover the window we need at least 50 cubes in memory which amounts to 30 MB. *l* is 2000 frames and with the speed of 25 frames per second, we are through in 80 seconds. We need at least 4 disks to meet this time constraint. If we reduce cube size to 20 voxels on each side, then we need only about 6 MB memory. But at the same time we need 1909 seconds to read all data with one disk. If we will read the same data in 80 seconds we need 24 independent disks.

As already mentioned we should be able to fly with the same speed in an arbitrary direction. We will have to use a parallel computer if we want speed. The less memory we have the more independent disks should be employed. To use a parallel computer we have to distribute data over the nodes in a way which will give a good load distribution regardless of the direction we fly in. This problem was solved by Ekeren and Aasheim, see [EkAas92] and [EkAas93].

Fault Tolerant Systems

Improving the quality of information systems by increasing the availability of data is regarded as a reasonable requirement in the near future. There is also an “internal” reason to take interest in reliability and availability for parallel systems. Parallel systems have a higher failure rate than ordinary systems. This is so, simply because parallel systems have more modules then ordinary monoprocessor systems. The more nodes in a system, the higher is the probability that one of the nodes will fail. To obtain fault tolerance the system must at least store two copies of the same data set. If we store one duplicate - after a node failure we have only one copy left. If we have a new failure and this failure hits the node with the only remaining copy, then we must restore data from archival copy and log. Until restore is completed, data is unavailable.

To improve availability we must keep the window of vulnerability as small as possible. This means that, after a fault, the system must start automatic repair immediately. The automatic repair implies reduplicating the single copy set onto the other nodes. To distribute load, some strategy must be employed to accomplish this.

Even during reduplication, the system should handle a normal transaction load. When the node is back in operation, the process should be reversed. Having fault tolerant systems is a quality in itself, it could also be used for reducing maintenance and servicing cost.

A research group at the Norwegian Telecom Research Facility i Trondheim, Norway are concentrating their efforts on fault tolerant high capacity transaction systems.

But as indicated above, fault tolerant systems are extremely complex and a lot of research is remaining. Relevant references: [Hva92], [HST91], [ToHv93] and [To94].

CONCLUSIONS

Parallel systems hold several promises since they are potentially: flexible, scalable, reliable and affordable in demanding database applications. It is not possible to take standard monoprocessor software and run it on parallel computers. The advantages of parallel computers come only with specially developed software. We are only in the starting phase of taking advantage of parallel computers. Parallel database systems will be an active research field for several years to come.

REFERENCES

[AsTo90] O.J.Aske and Ø.Torbjørnsen: “Communication on HC16-186 - A Study of Methods and Performance in a Hypercubic Network Based on Dual Port RAM.”, The Fifth Distributed Memory Computing Conference, Charleston, South Carolina April 1990.

[Astr79] N.M.Astrahan et al. “System R - A Relational Database Management System”, IEEE Computer Vol. 12, No 5, May 1979.

[BauGre89] B.A.W.Baugstø and J.Greipsland: “Parallel Sorting Methods for Large Data Volumes on a Hypercube Database Computer”, The Sixth International Workshop on Database Computers, France, June 1989

[BaGrKa90] B.A.W.Baugstø, J.F. Greipsland and J. Kamerbeek: “Sorting Large Data Files on POOMA”, Conpar, Sept. 1990.

[BeBiWi88] M. Beck, D. Bitton and W. K. Wilkenson “Sorting Large Files on a Backend Multiprocessor” IEEE Transactions on Computers, Vol 37, No 7, pp 769-778, July 1988.

[Bitt81] D.Bitton “Design, Analysis and Implementation of Parallel External Sorting Algorithms”, Ph. D. Thesis, University of Wisconsin, Madison, 1981.

[BiDeTu83] D.Bitton, D.DeWitt and C.Turbyfill “Benchmarking Database Systems. A Systematic Approach”, The 9th International Conference on Very Large Data Bases, Florence Oct. 1983.

[BorDew81] H.Boral and D.DeWitt “Processor Allocation Strategies for Multiprocessor Database Machines”, ACM TODS Vol. 6, No. 2, 1981.

[Bra80] K.Bratbergsengen, R.Larsen, O.Risnes and T.Aandalen “Neighbor Connected Processor Network for Performing Relational Algbra Operations” The 5th Workshop on Computer Architecture for Non-Numeric Processing, March 11-14, 1980 Pacific Grove, Ca.

[Bra84] K.Bratbergsengen “Hashing Methods and Relational Algbra Operations”, The 10th Conference on Very Large Data Bases, Singapore Aug. 1984.

[Bra90] K.Bratbergsengen “Parallel Execution of Relational Algebra Operations”, PRISMA Project Meeting, Nordwijk Sept. 1990. also in P. America (Ed.) Springer Verlag 1991.

[BraGje89] K.Bratbergsengen and T.Gjelsvik “The Development of the CROSS8 and HC16-186 Parallel (Database) Computers”, Division of Computing Systems and Telematics, The Norwegian Institute of Technology, The Sixth International Workshop on Database Machines, France June 19-23, 1989.

[DeWi83] D.J.DeWitt: “ Database Computers, an Idea Whose Time Has Passed”, The Second International Workshop on Database Machines, 1983.

[DeWi85] D.J.DeWitt and R.Gerber “Multiprocessor Hash-Based Join Algorithms”, The 11th Conference on Very Large DataBases, Stockholm Aug. 1985.

[DeWi87] D.J. DeWitt, S. Ghandeharizadeh, D. Schneider, R. Jauhari, M. Muralikrishna and A. Sharma “A Single User Evaluation of The Gamma Database Machine”, Proceedings of the 5th International Workshop on Database Machines, Japan, October 1987.

[DeWi88] D.J.DeWitt, S.Ghandeharizadeh and D.A.Schneider :"A Performance Analysis of the Gamma Database Machine",SIGMOD Proceedings, Vol. 17, No. 3 Sept. 1988.

[DWNaSc92] D.J.DeWitt, D.J. Naughton and D.A.Schneider “Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting” Proceedings First International Conference on Parallel and Distributed Info Systems, IEEE Press, Jan 1992.

[Eich87] M.Eich “MARS: The Design of a Main Memory Database Machine”, Database Machines and Knowledge Base Machines, Klüwer Academic Publishers, Boston 1988.

[EkAas92] J.H.Ekeren and Y.Tuseth Aasheim “ParaVision - Presentasjon av seismiske data på Hyperkuben”, Project Report IDT, May 1992, in Norwegian.

[EkAas93] J.H.Ekeren and Y.Tuseth Aasheim “ParaVision - A Visualization Tool for 3-D Datasets”, Masters Thesis IDT, May 1993.

[Good81] J.R. Goodman “An Investigation of Multiprocessor Structures and Algorithms for Database Management”, Technical Report UCB/ERL M81/33 Electronic Research Lab., College of Engineering, UCB, May 1981.

[Grae89] G.Graefe “Parallel External Sorting in Volcano” Oregon Graduate Center, TR No CS/E 89-008, June 1989.

[Hva92] S-O.Hvasshovd “HypRa/TR - A tuple Oriented Recovery Method for a DDBMS on a Shared Nothing Platform” Dr.ing. thesis, IDT 1992.

[HST91] S-O.Hvasshovd, T.Sæter and Ø.Torbjørnsen “Critical Issues in the Design of a Fault-Tolerant Multiprocessor Database Server”, in Proceedings from the 1991 Pacific Rim International Symposium on Fault-tolerant Systems, IEEE Computer Society Press, Sept. 1991.

[Kers87] M.L.Kerstens, P.M.G. Apers, M.A.W. Houtsma, E.J.A. van Kuyk and R.L.W. van de Weg “A Distributed, Main Memory Database Machine: Research Issues and Preliminary Architecture”, Database Machines and Knowledge Base Machines, Klüwer Academic Publishers, Boston 1988.

[Kits83] M.Kitsuregawa, H.Tanaka and T.Moto-oka “ Application of Hash to Data Base Machine and Its Architecture” New Generation Computing, Vol. 1, No. 1, 1983.

[Lela87] M.D.Palmer Leland and W.D. Roome “The Silicon Database Machine: Rationale, Design and Results”, Database Machines and Knowledge Base Machines, Klüwer Academic Publishers, Boston 1988.

[NaMi87] R.Nakano and M.Kiyama “MACH: Much Faster Associative Machine”, Database Machines and Knowledge Base Machines, Klüwer Academic Publishers, Boston 1988.

[Naka87] Shun-ichiro Nakamura, H.Minemura, T.Minohara, K.Itakura, M.Soga : “A High Speed Database Machine HDM”, The 5th Workshop on Database Machines, Karuizawa Oct. 1987, Proceedings: “Database Machines and Knowledge Base Machines”, edited by M.Kitsuregawa and H.Tanaka, Klüwer Academic Publishers 1988.

[Men86] J.Menon: “Sorting and Join Algorithms for Multiprocessor Database Machines” Database Machines, Modern Trends and Applications, Springer Verlag 1986.

[NyBaCvGrLo93] C.Nyberg, T.Barclay, Z.Cvetanovic, J.Gray and D.Lomet “AlphaSort: A RISC Machine Sort”, Digital San Francisco Systems Center, SFSC Technical Report 93.2, April 1993.

[PaGiKa88] D.A.Patterson, G.Gibson and R.H.Katz “A Case for Redundant Arrays of Inexpensive Disks (RAID)”, ACM Sigmod, Vol 17, No 3, September 1988.

[RiLuMi87] J.R.Richardson, H.Lu and K. Mikkilineni “Design and Evaluation of Parallel Join Algorithms”, ACM Sigmod Conference Proceedings, Vol. 16. No. 3, December 1987.

[Salz90] B.Salzberg, A.Tsukerman, J.Gray, M.Stewart, S.Uren and B.Vaughan "FastSort - A Distributed Single-Input Single-Output External Sort", SIGMOD 1990, pp 94- 101.

[San92] O.Sandstå "Utførelse av relasjonsalgebraoperasjoner", Thesis IDT May 1992 (in Norwegian).

SmiBro83] J. Smith and M. Brodie (eds) "Relational Database Systems. Analysis and Comparison" Springer Verlag 1983.

[Soo86] A.K.Sood, M. Abdelguerfi and W. Shu "Hardware Implementation of Relational Algebra Operations". Database Machines, Modern Trends and Applications, Springer-Verlag 1986.

[SoQu86] A.K. Sood and A.H. Qureshi (Eds) "Database Machines, Modern Trends and Applications", NATO ASI Series F: Computers and Systems Sciences, Vol. 24. Springer Verlag 1986.

[Ston76] M. Stonebraker et al. "The Design and Implementation of INGRES", TODS 2, sept. 1976.

[Su88] S.Y.W. Su "Database Computers, Principles, Architectures and Techniques" 497 pp, McGraw-Hill International Edition, 1988.

[Tor88] Ø.Torbjørnsen "Communication in a Failsoft Hypercube Database Machine", Department of Computing Systems and Telematics, The Norwegian Institute of Technology, Oct. 1988, The 6th International Workshop on Database Machines.

[ToHv93] Ø.Torbjørnsen and S-O.Hvasshovd "An Abstract Machine Supporting Fault-Tolerance in a Parallel Database Server", in Proceedings from the 1993 Pacific Rim International Symposium on Fault-Tolerant Systems, Melbourne Australia, Dec. 1993.

[Tor94] Ø.Torbjørnsen "Multi-Site Declustering Strategies for Very High Database Service Availability", Department of Computing Systems and Telematics, The Norwegian Institute of Technology, Doctoral Thesis, June 1994.

[TsuGra86] A.Tsukerman and J.Gray "FastSort - An External Sort Using Parallel Processors", Tandem Systems Review, Vol 3, No 4, December 1986, pp 57-72.

[VaGa84] P.Valduriez and G.Gardarin "Join and Semi-Join Algorithms for a Multiprocessor Database Machine", ACM Transactions on Database Systems 9, No 1, March 1984

[YaTa88] Y.Yamane, R.Take "Parallel Partition Sort for Database Machines and Knowledge Based Machines", in Kitsuregawa and Tanaka pp 1117 - 11130, Klüwer Academic Publishers, 1988.