PARALLEL DATABASE SYSTEMS

Database systems are computer systems designed specifically to manage large volumes of information. Since their inception in the 1960s, these systems have evolved into a diverse collection of architectures designed for different purposes and optimized along different metrics. The profound impact of technological advancement coupled with the diverse optimization strategies have yielded quite an array of database systems.

This article presents a survey of research that has been done in the area of parallel database systems. It provides a method of classifying database systems by architecture and presents the reader with a brief introduction to database systems in general, with emphasis on parallel database systems in particular.

A database machine is a computer system dedicated and tailored to carrying out the functionality of a database management system. Database machines run the gamut from small, single microcomputer conventional database systems to architectures with tens (or hundreds) of microprocessors

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*. Copyright © 1999 John Wiley & Sons, Inc.

operating on a single database, on the order of gigabytes, to multisite systems composed of high-end servers connected by extensive communication networks. It can be a highly special-purpose processor used to perform specific database operations. In addition, a database machine can be a single or multicomputer system designed to perform a variety of database operations.

The architecture of database machines is broken down into the two main areas: hardware database machines that exploit custom processors and software database machines that run on a collection of off-the-shelf processing elements. The categorization of software database machines excludes the class of distributed database management systems, which are built as a distinct layer of software that interfaces to some existing homogeneous or heterogeneous conventional database management system. The optimizing of software database machines is essentially an exercise in exploiting parallelism in these configurations. Hence, in recent years, the design of parallel database systems has been centered around conventional multiprocessor architectures. These architectures allow the designer of software database systems to harness parallelism in a seemingly natural way.

CLASSIFYING DATABASE SYSTEMS

Database systems can be categorized according to the taxonomy of Fig. 1. First, all database systems can be classified as either conventional, distributed, or parallel. The parallel systems are further subdivided into those that have a strong hardware orientation and those which are software oriented.

Conventional database systems are the common, monolithic architectures. Distributed database systems are those featuring a collection of autonomous, geographically dispersed systems which communicate via wide area communication network. The proposals of hardware database machines throughout the database management system literature were an attempt to provide high-performance operations by taking advantage of parallelism at the hardware level. The proponents of software database machines approach the performance problem of database management systems is to use off-the-shelf commodity processing elements and vendor-provided operating systems coupled with database software to design high-performance database systems.

CONVENTIONAL DATABASE SYSTEMS

A conventional database system consists of database management system (DBMS) software running on a conventional Von Neumann type computer system called a host. Figure 2 depicts a conventional database management system. The DBMS software runs on the host and is managed by the host's operating system. The database is stored on the secondary storage devices dedicated to the host processor. The architecture of the conventional computer does not match well with the requirements of a database management system. It results in a number of serious limitations and bottlenecks. Lim-

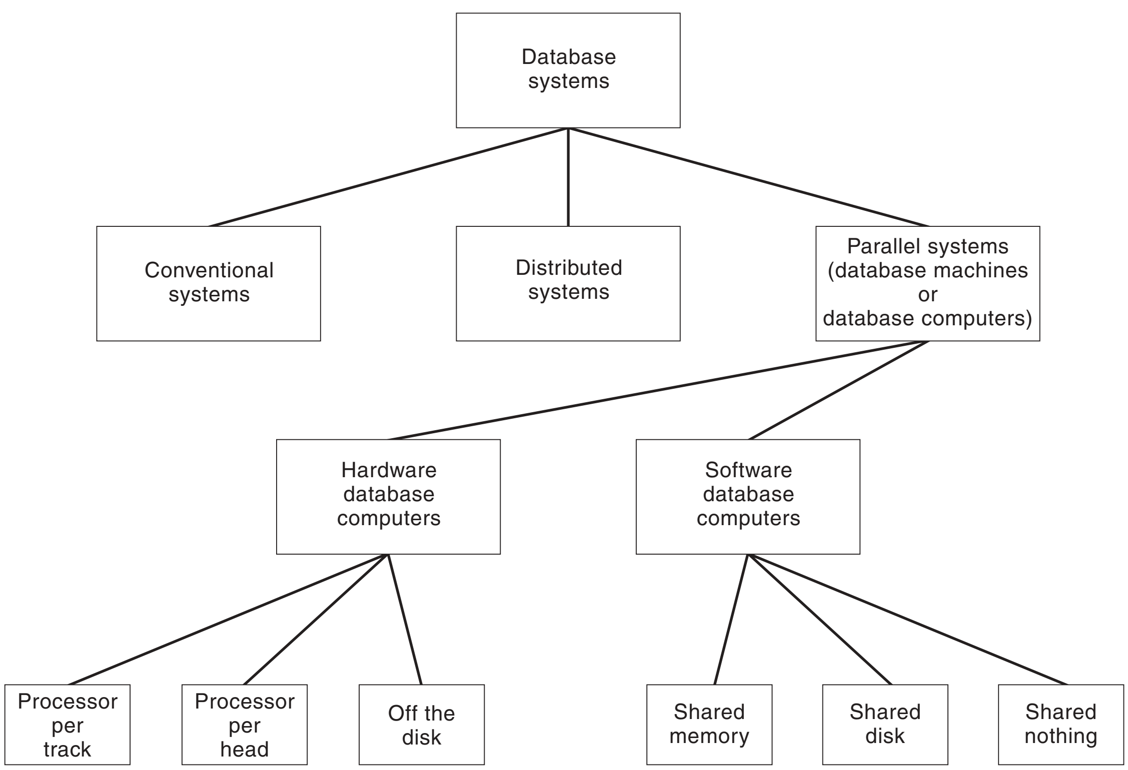

Figure 1. Taxonomy of database systems. This taxonomy classifies all database systems along the three main categories of conventional, distributed, or parallel database systems. The parallel systems are further subdivided into those that have a strong hardware orientation and those that are software oriented.

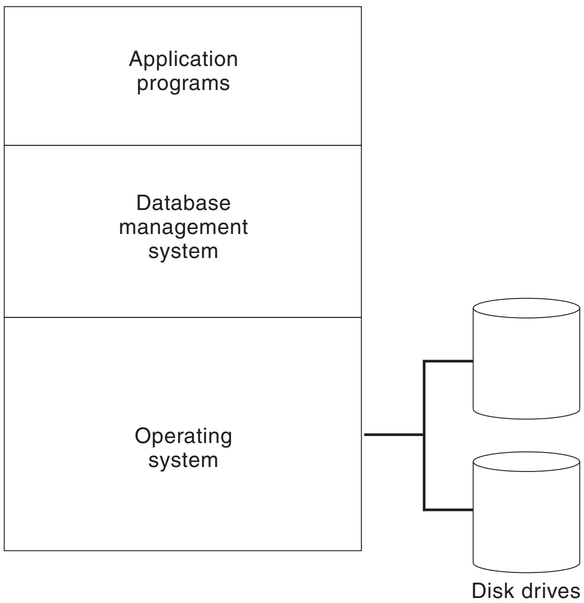

Figure 2. Conventional database system. Conventional database sytems are the common monolithic database architectures that consist of database management system software running on a conventional computer system called a host. The DBMS software runs on the host computer and is managed by the host's operating system. The physical database (or data repository) is stored on the secondary storage devices dedicated to the host processor.

itations of this architecture include reduced capacity, less reliability and availability, and the fact that the DBMS must contend with other applications for the host resources. Performance upgrades in conventional database systems can be costly and disruptive, requiring replacement of expensive hardware or modification of software. In short, such systems are not extensible, where extensibility of a database management system is defined as the capability of the system for upgrade with (1) no modification of existing software, (2) no additional programming, (3) no modification of existing hardware, and (4) no major disruption of system activity when additional hardware is being added. Examples of conventional database management systems are INGRES (1), Oracle (2,31), and Sybase (3).

Limitations of the Conventional Computer Architecture

Conventional computers execute programs by moving both instructions and data from the secondary storage devices to the central processing unit (CPU) via main memory and the associated controllers. This mode of operation results in limitations in secondary storage, main memory, and the processor. In addition, this architecture has potential bottlenecks due to the secondary storage to main memory and main memory to processor interfaces.

Secondary Storage

The conventional secondary storage devices are limited by their inability to process data locally. The read/write mechanisms of these devices are used exclusively for data transmission. Secondary storage devices are also limited by their speed. This is a direct consequence of the electromechanical nature of these devices. Quantitative performance data of computing equipment gathered over the last couple of decades has indicated that it is much more difficult to improve the performance of mechanical devices in comparison to improving the performance of electronic devices such as processor, which is related to improvements in the underlying silicon-based technology. The speed of secondary storage will always be a factor in determining performance in this architecture. Additional limitations result from the fact that data transmission can only be done one physical block at a time, through a single read/write head, and that data are stored and accessed by address rather than by content. The last issue limits the ability of storage devices to freely move data in order to maximize the usage of the storage space, without paying the high cost of maintaining address references (which from the database user's perspective is strictly an overhead).

DISTRIBUTED DATABASE SYSTEMS

A distributed database is one which is not stored in its entirety at a single physical location, but rather is spread across a network of geographically dispersed locations and connected via communication links (28,33). In general, a distributed database system (DDBS) consists of a collection of sites, or nodes, connected together via a communication network, and each site, in turn, has an autonomous database system, i.e., there is no central controller component. Figure 3 illustrates this approach. Each site has its own database, and a processor running its own local database management system. A distributed database system may be either heterogeneous or homogeneous, and the database may be replicated, parti-

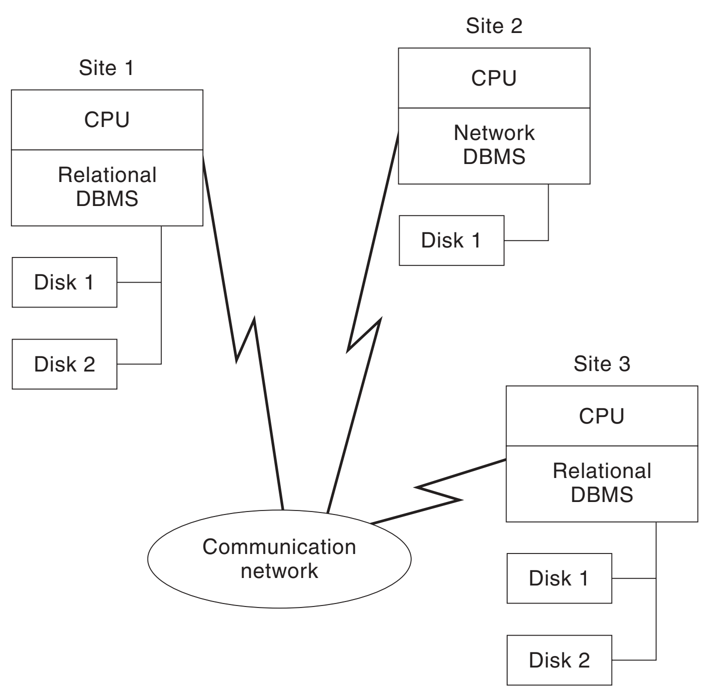

Figure 3. Distributed database system. Distributed database systems feature a collection of autonomous, geographically dispersed systems which communicate via a wide area communication network. Each site has its own database and a processor running its own local database management system.

tioned, or a combination of the two. Often the expense of large data transfers and the need to locate data where they are actually needed require duplicate databases.

The advantages of this approach include local autonomy, capacity and incremental growth, increased reliability and availability, and flexibility. Disadvantages include the need to duplicate databases, and complex concurrency control and security algorithms, which require large number of expensive control messages to be passed across the communication network (39). Examples of distributed database systems include Distributed INGRES (4), R* (5), and Distributed Sybase (3).

PARALLEL DATABASE SYSTEMS

The third main class of database systems shown in Fig. 1 contains those systems known as parallel database systems. The limitations of the conventional database systems and the sheer size of today's databases led to the notion of creating special-purpose computers to address the DBMS performance concerns. These architectures are further divided into those based on hardware approaches and those based on software approaches.

HARDWARE DATABASE MACHINES

Hardware database machines subscribe to the general theme of relying on special-purpose hardware to achieve the desired performance goals. These consist of some number of the following set of criteria in some desired order. The criteria include (1) low response time, (2) high data availability, (3) ability to store very large volume of data, order of tens of gigabytes, and (4) almost unlimited scaleability. The earliest of these approaches focused on architectures that were based on building associative disks, which basically fall into three main categories, namely: processor-per-track (PPT), processor-per-head (PPH), and Off-The-Disk (OTD).

The general idea behind these systems is to eliminate the limitations of the conventional secondary storage devices for database applications, by building more intelligence into the secondary storage device (37). This serves to increase the processing capabilities of the read/write mechanism. Hence, data stored on these devices can be directly searched and manipulated. The objective is to make the secondary storage devices intelligent enough so that they can select only the relevant portion of the data and then transfer them to the main memory for further processing. Thus, these systems process the data "on-the-fly" while they are read from the disks. Special-purpose processors, which are associated with the secondary storage devices, are utilized to perform this processing.

Processor-Per-Track Systems

The processor-per-track devices, also referred to as cellular logic devices, may be regarded as an upgrade from fixed-head disk. This approach seeks to overcome the limitations by assigning a dedicated processor to each track of a rotating memory device. All the processors can perform the same search operation in parallel, enabling the entire disk to be searched in one revolution. If the whole database is stored on a number of such storage devices, then the entire database can be searched in one revolution. More complex database operations can also be carried out by these devices, but they would involve more memory revolutions.

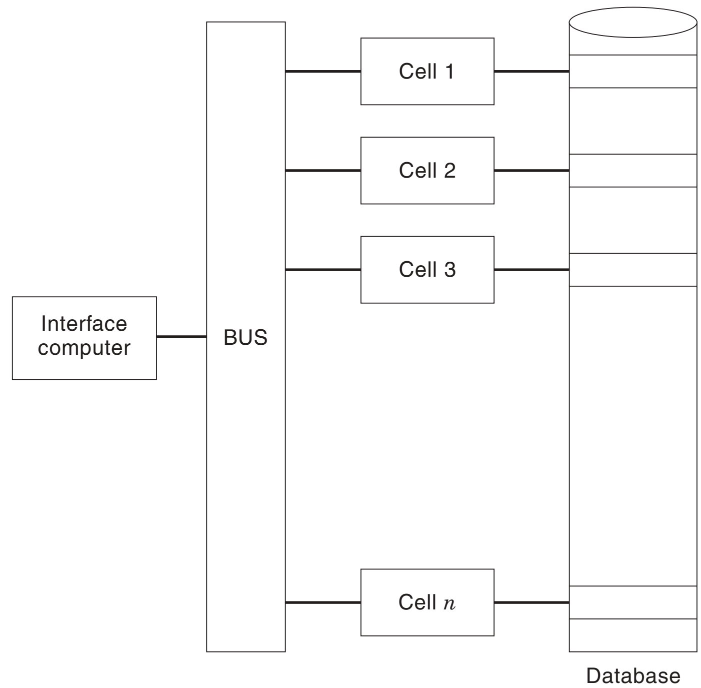

Figure 4. The context-addressed segment-sequential memory system architecture. The context-addressed segment sequential memory system architecture is an example of the processor-per-track or cellular logic device architecture. The database is stored on a fixed-head disk with a processing element dedicated to each read/write head. The controller is responsible for communication with the host, query distribution, and collation and processing of intermediate and final results.

The context addressed segment sequential memory (CASSM) (6), shown in Fig. 4, is an example of this type of architecture. It was designed to support the network, hierarchical, and relational data models. The database is stored on a fixed-head disk with a simple processing element dedicated to each read/write head. The entire system is supervised by a controller processor responsible for communications with the host, query distribution, and collation and processing of intermediate and final results. Data items in CASSM are stored as ordered pairs (⟨attribute, value⟩), and selections, performed on-the-fly by the cell processors, can be accomplished in three or four revolutions of the disk.

Other processor per track systems include the relational associative processor (RAP) (7), RARES (8), and Chang's Major/Minor Loop Machine (9), a magnetic bubble memory implementation.

Processor Per Head Systems

The cellular-logic systems offer tremendous parallel processing capability since each dedicated processor can process a portion of the database. The database segments are normally a full track of data. Unfortunately, the cost of building such a device can be quite high. We can reduce the number of special-purpose processors required for a large database by extending the size of the memory element. For example, a single track can be extended to the entire surface of a disk.

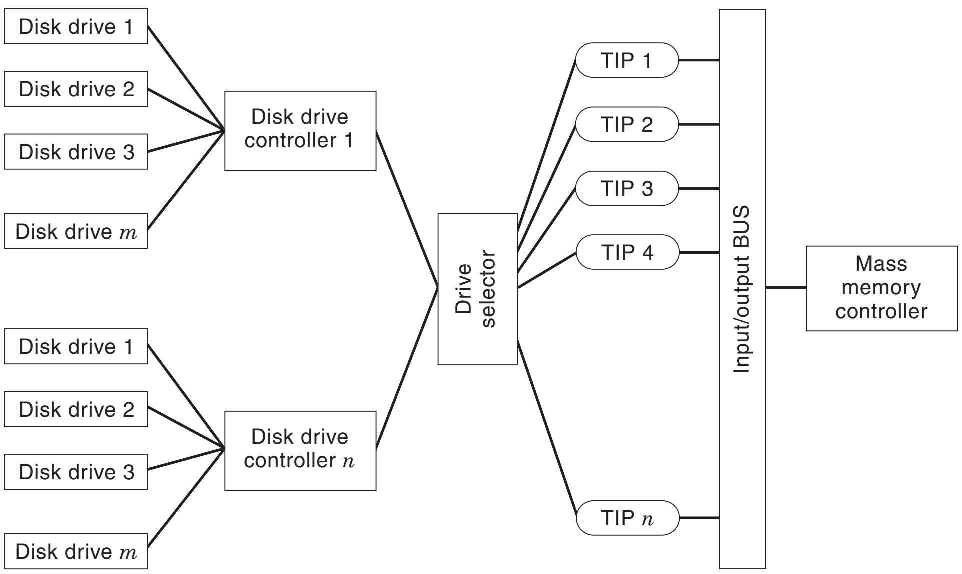

Figure 5. The mass memory architecture of the database computer. The mass memory architecture is an example of the processor-per-head approach. The database computer uses the mass memory unit to store its database. The mass memory uses several moving-head disks modified with parallel readout capability and connected by a switch to a number of processors which perform search operations on-the-fly.

This would change the configuration of the storage device to become an intelligent moving-head disk in which the processing element (read/write heads) can be dynamically moved to a selected track. The tracks under the processing elements form a cylinder, the contents of which can be processed in parallel. This is an alternative approach to the general principle of intelligent secondary storage, the so-called processor-per-head systems.

The processor-per-head systems employ one processor per surface of the disk, hence the amount of data that can be processed on-the-fly during one revolution is one track per surface, or one cylinder. Moving the processors between cylinders requires a seek operation. The processor per head approach may be viewed as an upgraded form of the moving head disk.

An example of an architecture incorporating processor-per-head technology is the database computer (DBC) (10), developed at Ohio State University. A functionally organized multiprocessor system, the database computer employs the processor per head approach as the basis for its mass memory unit, where the database is stored. Figure 5 depicts the mass memory architecture of the database computer. The mass memory uses several moving-head disks modified with parallel readout capability and connected by a switch to a number of processors which perform search operations on-the-fly. This type of operation is possible because every track of a cylinder is actually processed by a separate processing unit, called a track information processor (TIP), with dedicated buffer space (10). Based on the attribute model, the database computer stores a database as a collection of records, each containing a record body and a set of variable-length attribute–value pairs.

In addition, groups of records forming likely response sets are clustered on the disk devices. Query conjunctions are broadcast by the mass memory controller and stored in each of the track information processors. The track information processors simultaneously evaluate the query against their corresponding incoming record streams being read off the disks. This is accomplished in the following manner: First, each track information processor reads a record from the track as part of one data stream and the query conjunction as another data stream. Then, it performs bit-by-bit comparisons of the two streams. The track information processors then compare the value portion of the attribute–value pairs to determine if the query predicate is satisfied. Another processor-per-head architecture is the SURE search processor developed at the University of Braunschweig (11).

Off-the-Disk Systems

The off-the-disk category (also called processor per disk) employs conventional moving-head disks with a conventional disk controller, but interposes a filtering processor between the disk controller and the channel. This filtering processor applies search logic on-the-fly, eliminating unnecessary data. The off-the-disk approach provides the functionality of the processor-per-track and processor-per-head systems at a lower cost because there is less custom hardware. However, the performance of the off-the-disk systems is less than the performance of the processor-per-track or processor-per-head systems.

Figure 6 shows the architecture of the content-addressable file store (CAFS) (12), which is an example of the off-the-disk design. In CAFS a special processor and random-access bit-addressable memory are positioned between the rotating storage device and the host. Records are read from the conventional disk devices into the key register area in the content-addressable file store. These registers can compare query predicates with attribute values in parallel. Results are forwarded to the search evaluation unit, where qualifying records are selected. Projections on the applicable records are then performed by the retrieval unit and forwarded to the user at the host.

Other systems classified as off-the-disk architectures are many and varied. They include the Britton-Lee IDM-500 (13), Delta Machine (14), SM3 (15), and VERSO (16).

The literature of database management systems has numerous references to the so-called back-end machine architecture. Back-end machines attempt to solve the database management system problem by off-loading the database management system functionality onto a back-end machine.

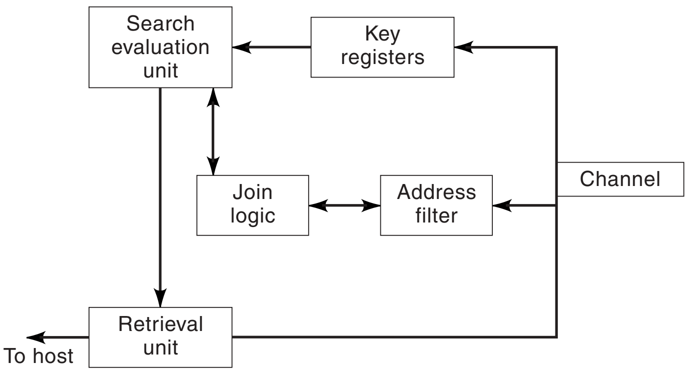

Figure 6. The context-addressable file store system architecture. The context-addressable file store system architecture is an example of the off-the-disk design. In this configuration a special processor and random-access bit-addressable memory are positioned between the rotating storage device and the host. The key registers perform parallel evaluation of query predicates against data read from the disk devices, before forwarding results to the search evaluation unit, where qualifying records are selected.

This approach results in excessive message traffic when trying to off-load a record-at-a-time language. It has been shown that only relational systems can be successfully off-loaded.

The rapidly declining price of off-the-shelf commodity general-purpose processing units makes custom hardware unattractive. Also, a database designer would prefer an architecture in which multiple processors could be used to provide any needed degree of performance on a user application. Both of these have contributed toward the trend of developing software database systems using standard hardware components. Hence, the best approach for the design of high-performance database systems should be based on conventional multiprocessor techniques.

SOFTWARE DATABASE MACHINES

Software database systems are those which do not employ a significant amount of special-purpose hardware and where most of the functions of database management are done in software. There are three possible architectures to exploit multiprocessor parallelism: (1) shared memory, (2) shared disk, and (3) shared nothing. In the shared memory configuration, a number of processors are attached to the memory bus and each has access to a common memory. This architecture is very pervasive throughout the UNIX server market. In the shared disk architecture, a number of processors with local memory can access a shared disk system. This architecture has been very popular in the recent collection of massively parallel systems from Thinking Machines Corporation, Intel, and N-cube. The VAXcluster, by Digital Equipment Corporation, is a more conventional shared disk architecture. In the shared nothing architecture, a collection of processors with private memory and disks are connected together via an interconnection network. The interconnection network varies based on the proximity of the processors and fall into the following generic categories: (1) massively parallel processor (MPP) network, (2) local area network, and (3) wide area network.

Taxonomy of Parallel Architectures

In this section a taxonomy is presented to lend some credence to the breadth of design alternatives for multiprocessors and the context that has led to the development of the dominant form of multiprocessors. The alternative and rationale behind them will be briefly described.

The idea of using multiple processors both to increase performance and to improve reliability dates back to the very inception of the electronic computers when Flynn proposed a simple model for categorizing all computers (17). This model is still useful today. He looked at parallelism in the instructions and data streams called for by the instructions, and placed all computers in one of four categories:

1. Single instruction stream, single data stream (SISD). This is the conventional uniprocessor architecture.
2. Single instruction stream, multiple data stream (SIMD). In this architecture the same instruction is executed by multiple processors using different data streams. Each processor has its own data memory (hence multiple data), but there is a single instruction and memory and control processor, which fetches and dispatches instructions. The processors are typically special purpose, since full generality is not required.
3. Multiple instruction streams, single data stream (MISD). No commercial machine of this type has been built to date.
4. Multiple instruction streams, multiple data streams (MIMD). In this architecture each processor fetches its own instruction and operates on its own data. The processors are normally off-the-shelf microprocessors.

This is a coarse model, as some machines are hybrids of these categories. However, it serves the purpose of putting a framework on the design space. Many of the early multiprocessors were SIMD, and the SIMD model received renewed attention in the 1980s. SIMD works best in dealing with arrays in for-loops. Hence, to have the opportunity for massive parallelism in SIMD there must be massive amounts of data parallelism. The SIMD model is once again suffering from waning interests as a general-purpose multiprocessor architecture, for two main reasons. First, it is too inflexible. A number of important problems cannot use this style of machine, and the architecture does not scale down in a competitive fashion; that is, small-scale SIMD machines often have worse cost/performance compared with that of the alternative. Second, SIMD cannot take advantage of the tremendous performance and cost advantages of microprocessor technology. Instead of leveraging this low-cost technology, designers of SIMD machines must build custom processors for their machines.

In recent years MIMD has clearly emerged as the architecture of choice for general purpose multiprocessors. Two factors are primarily responsible for the rise of the MIMD machines: (1) they offer flexibility, and (2) MIMD can build on the cost performance of off-the-shelf microprocessors. MIMD machines fall into two classes, depending on the number of processors, which in turn dictate the memory organization and interconnection strategy. The two classes are centralized shared-memory and distributed-memory.

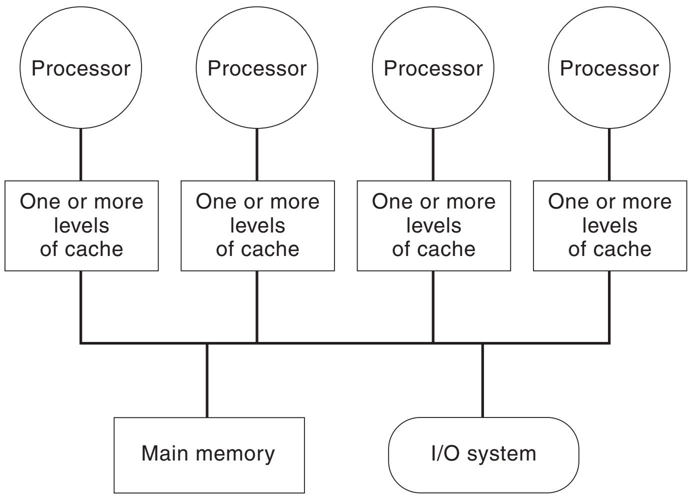

Figure 7. Centralized shared-memory multiprocessor architecture. The centralized shared-memory multiprocessor architectures have at most a collection of a few dozen processors that are interconnected and share a single centralized memory via a bus. This architecture offers excellent performance because the memory demands of the processors can be satisfied with large caches connected to the individual processors.

Shared Memory

The centralized shared-memory architectures, Fig. 7, have at most a collection of a few dozen processors that are interconnected and share a single centralized memory via a bus. Multiprocessors with a small number of processors are able to share a single centralized memory in this manner and get excellent performance, primarily because the memory demands of the processor can be satisfied with large caches connected to the individual processors (17,36). The single main memory in this architecture provides uniform access times to all of the processors. This type of shared-memory architecture is currently the most popular architecture by far for general-purpose computing.

The ability to separate process allocation from the parallel execution of selection operations is an issue of significant importance in the parallel execution model of queries. A shared-memory architecture has the distinct advantage of uncoupling the implementation of the parallel execution model of queries from the process allocation strategy (30,34). Each operator in a query tree of a selection operation can be assigned one or more processes to perform execution of the activities associated with the operator. A scheduler process is used to assign processes to available processors that can provide the required service. The coordination and synchronization of the activities and interaction of the processes is done through the centralized shared memory. The typical consumer–producer operation of inter-operator interaction is coordinated through shared memory. This is also true of the intra-operator interaction associated with parallelism within an operator (27), due primarily to horizontal partitioning of the relation. Parallel accessed structures are protected using mutual exclusion semaphore, an operating system construct that is also a consequence of the shared-memory architecture.

Shared Nothing

The second class of MIMD computers consists of machines with physically distributed memory. The bandwidth demands of the larger number of processors required that the memory be distributed among the processors, otherwise the memory system would be saturated by the memory demands of the processors (17,36). Not having a centrally shared memory for information exchange, the distributed-memory organization requires a high bandwidth interconnection network. Figure 8 illustrates the architecture for these machines.

The distribution of memory among the nodes of the distributed-memory architecture has two major advantages: (1) it provides a low-cost method to scale the memory bandwidth, and (2) the latency for access to the local memory is reduced. These advantages play a key role in making distributed memory architecture attractive at smaller processor count as processors get faster and require more memory bandwidth and lower memory latency. The major disadvantage of this architecture is that interprocessor communication is more complex.

The method of communication among distributed-memory machines partitions this class into two major architectural types. In the first category, the physically separate memories can be addressed as one logically shared address space, i.e., a memory reference can be made by any processor on any memory location. These machines are called distributed shared memory or scaleable shared-memory architectures (17,35). The BBN GP1000 (18,26), by BBN Butterfly and Monarch, is an example of this architecture.

Alternatively, the address space can consist of multiple private address spaces that are logically disjoint and cannot be accessed by a remote processor (17,35). Each processor-memory module is essentially a separate computer, forming the so-called shared nothing configuration with the interconnection network determining the degree of coupling. These machines are also referred to as message-passing multicomputers. The Intel iPSC 860 (17), a hypercube-connected collection of i860s, is based on this architecture. More recent machines such as Intel Paragon (17) have used networks with lower dimensionality and higher individual links. The Thinking Machines CM-5 (19,32,38) makes use of off-the-shelf microprocessors and a fat tree interconnect. It provides user-level access to the communication channel, thus significantly improving communication latency. In 1995, these were the state of the art in message-passing multicomputers. In the next section we describe the Gamma database machine running on an Intel iPSC/2 hyper-cube which is a shared-nothing machine.

Gamma Database Machine

The Gamma database machine (20) is a relational database system operating on an Intel iPSC/2 hyper-cube with 32 processors. Each processor is configured with a 386 CPU, 8 megabytes of RAM, and a 330 megabyte MAXTOR SCSI disk drive. The nodes are interconnected via custom VLSI routing modules forming a hyper-cube. Gamma is built on top of the NOSE operating system (20) which is designed specifically for supporting database management systems. NOSE uses non-preemptive scheduling to prevent convoys. In addition, NOSE provides lightweight processes and an interprocess communication mechanism based on the reliable message passing hardware of the Intel iPSC/2 hyper-cube. NOSE provides file services based on the Wisconsin Storage System (WiSS) (21,40).

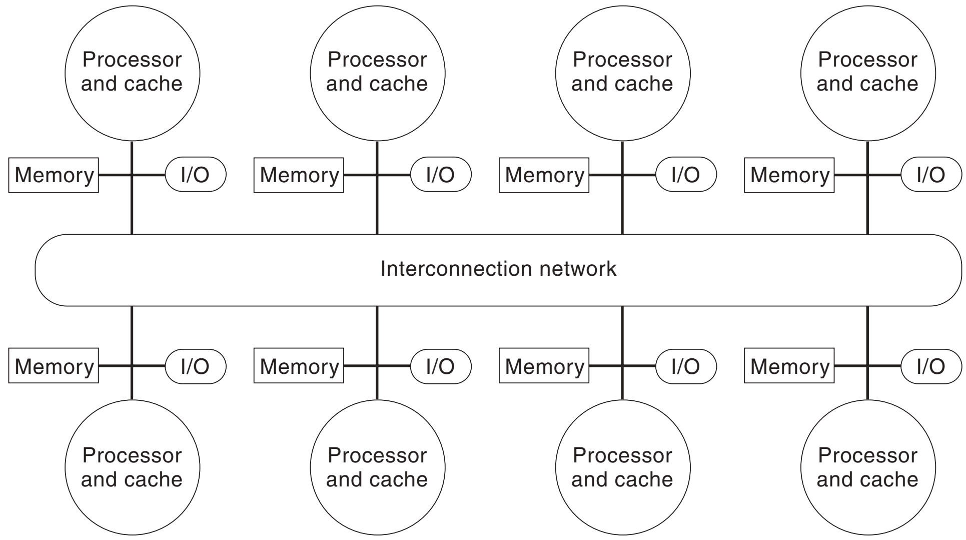

Figure 8. Distributed-memory multiprocessor architecture. The distributed-memory multiprocessor architecture is an example of the MIMD computers in which memory is physically distributed among the processors. Information exchange is facilitated by way of a high-bandwidth interconnection network.

The relations in Gamma are horizontally partitioned across the disk drives enabling the database software to exploit the available system I/O bandwidth. The declustering of a relation makes the task of parallelizing a selection operator easier. This is because the selection is reduced to that of starting a copy of the operator on each processor. During query optimization, partition information for the source relation is incorporated into the query plan and is used by the query scheduler to determine the set of processors to be involved in the execution of the selection query. For range partitioned relations, the scheduler restricts the execution to those processors whose range overlaps the range of the selection predicate.

The Gamma software database machine is organized around a number of processes. The Catalog Manager acts as a central repository of all conceptual and internal schema information for individual databases. The schema information is loaded whenever a database is first opened. A Query Manager is associated with each active Gamma user. A Scheduler process is responsible for the management of a multisite query. The scheduler process activates the operator processes used to execute the nodes of a compiled query tree. Operator processes correspond to operators in a query tree. One or more operator processes are executed at each processor participating in the execution of the operator.

The Gamma database machine uses conventional relational techniques for query parsing, optimization, and code generation. The complexity of the query optimization process is reduced because Gamma only utilizes hash-based parallel algorithms for joins and other complex operations. In addition, queries are compiled into a left-deep tree of operators with each operator being executed by one or more operator processes at each participating node.

The Gamma database machine architecture utilizes three main concepts which allows it to scale to hundreds of processors: (1) relations are horizontally partitioned across multiple disk drives attached to separate processors, allowing relations to be scanned in parallel, (2) hash-based parallel algorithms are used to process complex relational operators such as joins and aggregate functions, and (3) dataflow scheduling techniques, based on the bracket operator model (22), are used to coordinate execution of multioperator queries. The use of these techniques facilitates the execution of very complex queries with little or no coordination. This feature is necessary for machine configurations with large processor counts.

OTHER SOFTWARE DATABASE SYSTEMS AND CONCEPTS

Volcano Query Processing System

The Volcano Query Processing System (22) is a dataflow query processing system that is extensible by adding new operators. The system was designed to parallelize single threaded query processing algorithms without modifying their implementations. This was done using the operator model (this form of parallelizing a query is described subsequently).

The bracket model of parallelism has a generic template that can send and receive data and execute one operator at a time. The Gamma database machine uses this model. The template code invokes the operator which then controls execution. A major disadvantage of this model is that each locus of control has to be created. This is done using a scheduler process and requires additional software development beyond the operator functionality, for each operator in the set of query processing algorithms (22). Thus, this model is not well suited for system extensibility.

The operator model of parallelizing a query evaluation engine is focused around the reuse of single-threaded query processing code, resulting in self-scheduling parallel processing. Execution control is localized in an operator that provides a standard iterator interface to operators above and below it in a query tree. The exchange iterator module encapsulates parallelism and thus reduces the complexity of implementing parallel database algorithms. The iterator has open, next, and close procedures and therefore can be inserted in multiple places in a complex query tree. Figure 9 shows a complex query execution plan that includes data processing operators.

The exchange iterator provides the necessary support for vertical and horizontal parallelism in Volcano. The open call creates a child process which enters into a producer–consumer relationship with the parent process. The pro-

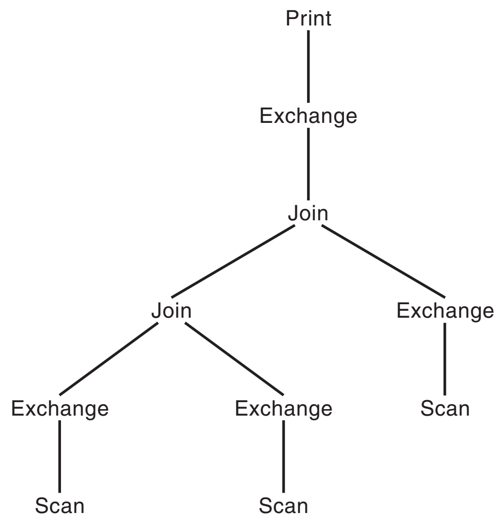

Figure 9. Operator model of parallelization. The operator model of parallelizing of a query evaluation engine is focused around the reuse of single-threaded query processing code. Execution control is localized in an operator that provides a standard iterator interface to operators above and below it in the query tree, called the exchange operator. The exchange iterator encapsulates parallelism and thus reduces the complexity of implementing parallel database algorithms.

ducer–consumer relationship of exchange uses the data-driven dataflow paradigm and uses standard UNIX operating system interprocess communication mechanisms for synchronization.

Active Areas of Research in Software Database Systems

One of the primary areas of current interest in software database systems seems to be the design and development of efficient execution strategies for parallel evaluation of multi-join queries. A comparative performance evaluation of four popular execution strategies was presented in Ref. 23. The execution strategies are sequential parallel, synchronous execution, segmented right-deep, and full parallel. The study was conducted by implementing the four approaches on the same parallel database system, PRISMA/DB. PRISMA/DB (23) is a full-fledge parallel, main-memory relational database management system that runs on a 100-node shared-nothing multiprocessor.

Another area of interest is load balancing with skewed data distribution. High start-up time, interference between execution entities, and poor load balancing can limit the performance of parallel query execution. A solution to reduce the impact of these factors in DBS3, a shared-memory architecture, was presented in Ref. 24. DBS3 (Database System for Shared Store) (25) is a parallel database system for shared-memory multiprocessors. DBS3 runs on a KSR1 multiprocessor. The KSR1 (29) is a distributed shared-memory configuration and provides a shared-memory programming model in a scalable and highly parallel architecture.

The decision to choose a particular hardware architecture for implementing a software database machine depends on a number of factors; one has to make trade-off decisions based on the metrics of importance being considered. A shared-memory system will be the cheapest option until performance demands exceed its upper bound. A shared-nothing system can scale almost arbitrarily to contain as many processors as deemed necessary to perform the task. A shared-disk system offers neither the cheapness of a shared-memory configuration nor the scalability of a shared-nothing system. This implies that the shared disk is the least attractive of the three alternatives.

SUMMARY AND CONCLUSIONS

This article presented a detailed discussion of parallel database systems. A taxonomy consisting of three major categories of systems (conventional, distributed, and parallel database systems) was presented. The parallel systems were further categorized into hardware-oriented and software-oriented architectures.

The early parallel systems were the hardware database machines that exploited custom hardware to improve performance. The more recent systems are primarily software database machines using standard multiprocessor technology; most of which fall into the multiple instruction streams, multiple data streams architecture. The MIMD configuration seems to be the architecture of choice primarily because it allows for almost unlimited scalability, a property that is deemed necessary for very large databases.

The current research efforts in software database machines include the design and development of efficient execution strategies for parallel evaluation of selection queries. This is crucial to maximizing performance in the shared-nothing architecture that uses message passing for interprocess communication. The research and commercial multiprocessor database systems and the availability of low-cost processors have made supporting very large databases a reality.

BIBLIOGRAPHY

1. M. Stonebraker et al., The design and implementation of INGRES, *ACM Trans. Database Sys.,* **1** (3), 1976.
2. F. D. Rolland, *Relational Database Management with Oracle,* Reading, MA: Addison-Wesley, 1992.
3. R. Gillette, D. Muench, and J. Tabaka, *Physical Database Design for SYBASE SQL Server,* Englewood Cliffs, NJ: Prentice-Hall, 1995.
4. M. Stonebraker and E. Neuhold, A Distributed Database Version of INGRES, *Proc. 2nd Berkeley Workshop on Distributed Databases and Computer Networks,* May 1976.
5. R. Williams, R*: An Overview of the Architecture, Announcement, Los Angeles, 1986.
6. S. Y. W. Su and G. J. Lipovski, CASSM: A cellular system for very large databases, *Proc. VLDB Conf.,* 1986.
7. E. A. Ozkarahan, S. A. Schuster, and K. C. Smith, RAP—Associative processor for database management, *AFIPS Conf. Proc.,* **44**, 1975.
8. S. C. Len, D. C. P. Smith, and J. M. Smith, The design of a rotating associative memory relational database applications, *ACM Trans. Database Sys.,* **1** (1): 1976.
9. D. Smith and J. Smith, Relational database machines, *IEEE Comp.,* March 1979.
10. J. Banerjee, R. I. Baum, and D. K. Hsiao, Concepts and capabilities of a database computer, *ACM Trans. Database Sys.,* **3** (4): 1978.

11. H. O. Leilich, G. Stiege, and H.Ch. Zeidler, A search processor for database management systems, *Proc. 4th Conf. on Very Large Databases,* 1978.
12. R. W. Mitchell, Content addressable file storage, *Proc. Online Database Technology Conf.,* Online Conferences Ltd., April 1976.
13. Britton Lee, Inc., IDM 500 Intelligent Databases Machine, Product Announcement, 1980.
14. S. Shibayama et al., A relational database machine with large semiconductor disk and hardware relational algebra processor, *New Gen. Comp.,* **2**, 1984.
15. C. Baru and S. Su, The architecture of SM3: A dynamically partitionable multicomputer system, *IEEE Trans. Comp.,* **C-35** (9), 1986.
16. H. O. Leilich and M. Missikoff, *Database Machines,* Springer-Verlag, 1983.
17. D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach,* San Francisco, CA: Morgan Kaufmann, 1996.
18. R. D. Rettberg et al., The Monarch parallel processor hardware design, *IEEE Comp.,* **23**, 1990.
19. W. D. Hill, *The Connection Machine,* Cambridge, MA: MIT Press, 1985.
20. D. J. DeWitt et al., The gamma database machine project, *Reading in Database Systems,* San Mateo, CA: Morgan Kaufmann, 1994.
21. D. J. DeWitt, J. F. Naughton, and D. A. Schneider, Parallel sorting on a shared-nothing architecture using probabilistic splitting, *Reading in Database Systems,* San Mateo, CA: Morgan Kaufmann, 1994.
22. G. Graefe, Encapsulation of parallelism in the volcano query processing system, *Reading in Database Systems,* San Mateo, CA: Morgan Kaufmann, 1994.
23. A. N. Wilschut, J. Flokstr, and P. M. G. Apers, Parallel evaluation of multi-join queries, *Parallel Computation: Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O,* Klagenfurt, Austria: September 1996.
24. L. Bouganium, D. Florescu, and B. Dageville, Skew handling in the DBS3 parallel database system, *Parallel Computation: Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O,* Klagenfurt, Austria: September 1996.
25. B. Gergsten, M. Couprie, and P. Valduriez, Prototyping DB3, a shared-memory parallel database system, *International Conference on Parallel and Distributed Information Systems,* Florida, 1991.
26. BBN Laboratories, Butterfly parallel processor overview, Tech. Rep. 6148, Cambridge, MA: BBN Laboratories, 1986.
27. N. Biscondi et al., Encapsulation of intra-operator parallelism in a parallel match operator, *Parallel Computation: Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O,* Klagenfurt, Austria: September 1996.
28. A. R. Bobak, *Distributed and Multi-Database Systems,* Norwood, MA: Artech House, 1996.
29. H. III Burkhardt et al., Overview of the KSR1 computer system, Tech. Rep. KSR-TR-9202001, Boston, MA: Kendel Square Research, 1992.
30. P. Ciaccia and A. Veronesi, Dynamic declustering methods for parallel grid files, *Parallel Computation: Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O,* Klagenfurt, Austria: September 1996.
31. D. Ensor and I. Stevenson, *Oracle Design,* Sebastopol, CA: O'Reilly & Associates, 1997.
32. E. F. Gehringer, D. P. Siewiorek, and Z. Segall, *Parallel Processing: The Cm* Experience,* Bedford, MA: Digital Press, 1987.
33. T. M. Ozsu and P. Valduriez, *Principles of Distributed Database Systems,* Englewood Cliffs, NJ: Prentice-Hall, 1991.
34. W. Hong and M. Stonebraker, Optimization of parallel query execution plans in XPRS, *Reading in Database Systems,* San Mateo, CA: Morgan Kaufmann, 1994.
33. M. Stonebraker, The case for shared nothing, *IEEE Database Engineering,* **9** (1): March 1986.
36. H. Stone, *High Performance Computers,* New York: Addison-Wesley, 1991.
37. S. Y. W. Su, *Database Computers: Principles, Architectures and Techniques,* New York: McGraw-Hill, 1988.
38. R. J. Swan et al., The implementation of the Cm* multi-processor, *Proc. AFIPS National Computing Conf.,* 1977, pp. 645–654.
39. A. S. Tanenbaum and S. J. Mullender, Operating systems requirements for distributed database systems, in H. J. Schneider (ed.), *Distributed Data Bases,* Amsterdam: North Holland, 1982.
40. H-T. Chou et al., Design and implementation of the Wisconsin storage system (Wiss), *Software Practices and Experiment,* **15** (10), October 1985.

DENVER WILLIAMS
MOSTOFA A. BASSIOUNI
ALI OROOJI
University of Central Florida

PARALLEL MACHINES. See DATAFLOW AND MULTITHREADED ARCHITECTURES.