MASSIVELY PARALLEL PROCESSING: ARCHITECTURE AND TECHNOLOGIES

Prem N. Mehra

INSIDE

System Components, Hardware Configurations, Multiprocessors: Scalability and Throughput, Software Layers, DBMS Architecture

INTRODUCTION

Organizations today require a great deal more processing power to accommodate new kinds of applications, such as an information delivery facility, data mining, and electronic commerce. This need for large computing cycles may be met relatively inexpensively through massively parallel processing.

Although massively parallel processors have been around for some time, their use has been limited primarily to scientific and technical applications, and to applications in the defense industry. Today, parallel processing is moving out to organizations in other industries, which find their own data volumes growing exponentially. The growth is not only from traditional transaction systems but also from new data types, such as text, images, audio, and video. Massively parallel processing is enabling companies to exploit vast amounts of data for both strategic and operational purposes.

Parallel processing technology is potentially very powerful, capable of providing a performance boost two to three orders of magnitude greater than sequential uniprocessing and of scaling up gracefully with data base sizes. However, parallel processing is also easy to misuse. Data base management system (DBMS) and hardware vendors offer a wide variety of products, and the technical features

PAYOFF IDEA

Massively parallel processing, once exclusively limited to scientific and technical applications, offers organizations the ability to exploit vast amounts of data for both strategic and operational purposes. This article reviews parallel processing concepts, current and potential applications, and the architectures needed to exploit parallel processing — especially massively parallel processing — in the commercial marketplace.

of parallel processing are complex. Therefore, organizations must work from a well-considered vantage point, in terms of application, information, and technical architectures.

This article reviews parallel processing concepts, current and potential applications, and the architectures needed to exploit parallel processing — especially massively parallel processing — in the commercial marketplace. Article 3-02-48 focuses on implementation considerations.

PARALLEL PROCESSING CONCEPTS

Massively parallel computing should be placed within the context of a more general arena of parallelism, which has been in use in the commercial marketplace for some time. Many variations and algorithms for parallelism are possible; some of the more common ones are listed here:

- *OLTP.* Concurrent execution of multiple copies of a transaction program under the management of an online terminal monitor processing (OLTP) manager is one example. A program written to execute serially is enabled to execute in parallel by the facilities provided and managed by the OLTP manager.
- *Batch.* For many long-running batch processes, it is common to execute multiple streams in parallel against distinct key ranges of files and merge the output to reduce the total elapsed time. Operating systems (OS) or DBMS facilities enable the multiple streams to serialize the use of common resources to ensure data integrity.
- *Information delivery facility (data warehouse).* The use of nonblocking, asynchronous, or pre-fetch input/output (I/O) reduces the elapsed time in batch and information delivery facility environments by overlapping I/O and CPU processing. The overlap, parallelism in a loose sense, is enabled by the DBMS and OS without user program coding specifically to invoke such overlap.

Recent enhancements in DBMS technologies are now enabling user programs, still written in a sequential fashion, to exploit parallelism even further in a transparent way. The DBMS enables the user SQL (structured query language) requests to exploit facilities offered by the new massively parallel processors by concurrently executing components of a single SQL. This is known as intra-SQL parallelism. Large and well-known data warehouse implementations at retail and telecommunications organizations (such as Sears, Roebuck and Co. and MCI Corp., respectively) are based on exploiting this form of parallelism. The various architectures and techniques used to enable this style of parallelism are covered in this article.

Facets of Parallelism

From a hardware perspective, this article focuses on a style of parallelism that is enabled when multiple processors are arranged in different con-

figurations to permit parallelism for a single request. Examples of such configurations include symmetric multiprocessors (SMP), shared-disk clusters (SDC), and massively parallel processors (MPPs).

From an external, high-level perspective, multiple processors in a computer are not necessary to enable parallel processing. Several other techniques can enable parallelism, or at least give the appearance of parallelism, regardless of the processor environment. These styles of parallelism have been in use for some time:

- *Multitasking or programming.* This enables several OLTP or batch jobs to concurrently share a uniprocessor, giving the appearance of parallelism.
- *Specialized processing.* Such functions as I/O managers can be executed on specialized processors to permit the CPU's main processor to work in parallel on some other task.
- *Distributed processing.* Similarly, peer-to-peer or client/server style of distributed processing permits different components of a transaction to execute in parallel on different computers, displaying yet another facet of parallelism.

Multiprocessors, important as they are, are only one component in enabling commercial application program parallelism. Currently, the key component for exploiting parallelism in most commercial applications is the DBMS, whose newly introduced intra-SQL parallelism facility has propelled it to this key role. This facility permits users to exploit multiprocessor hardware without explicitly coding for such configurations in their programs.

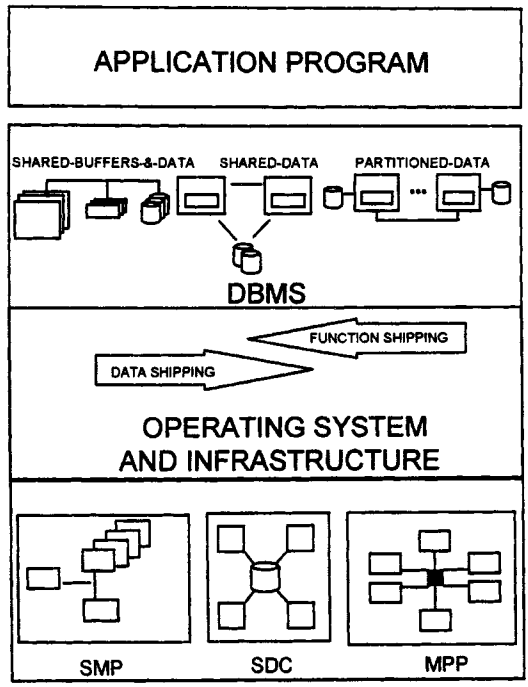
System Components

In general, from the perspective of commercial application programs, such facilities as parallelism are characteristics of the entire system, not merely of its components, which include the hardware, operating system, infrastructure, and DBMS. However, each of the components plays an important role in determining the system characteristics.

Exhibit 1 shows one way to depict a system in a layered fashion. The hardware layer, which consists of a multiple processor configuration, offers the base component. The operating system (OS) and the function and I/O shipping facilities, called *infrastructure*, influence the programming model that the DBMS layer uses to exhibit the system characteristics, which is exploited by the application program.

This article first discusses the hardware configurations and their characteristics, including scalability. It then discusses the DBMS architectures, namely shared data and buffer, shared data, and partitioned data. The ar-

EXHIBIT 1 — System Components



ticle focuses on the DBMS layer because it is generally recognized that the DBMS has become the key enabler of parallelism in the commercial marketplace. The OS and infrastructure layer will not be explicitly discussed, but their characteristics are interwoven in the hardware and the DBMS discussion.

HARDWARE CONFIGURATIONS

Processors are coupled and offered to the commercial users in multiple processor configurations for many reasons. These have to do with a number of factors — performance, availability, technology, and competition — and their complex interaction in the marketplace.

Multiprocessors provide enhanced performance and throughput because more computing resources can be brought to bear on the problem. In addition, they also provide higher data availability. Data can be accessed from another processor in the event of a processor failure. Historically, enhanced data availability has been a key motivation and driver for using multiple processors.

Multiprocessor Classification

A formal and technically precise multiprocessor classification scheme developed by Michael J. Flynn is widely used in technical journals and by research organizations. Flynn introduced a number of programming models, including the multiple instruction multiple data stream (MIMD), the single instruction multiple data stream (SIMD), the single instruction single data stream (SISD), and the multiple instruction single data stream (MISD). This taxonomy, based on the two key characteristics of processing and data, is still used. For the purposes of this article, however, multiprocessors can be classified into three simple types:

1. SMP (symmetric multiprocessors)
2. SDC (shared-disk clusters)
3. MPP (massively parallel processors)

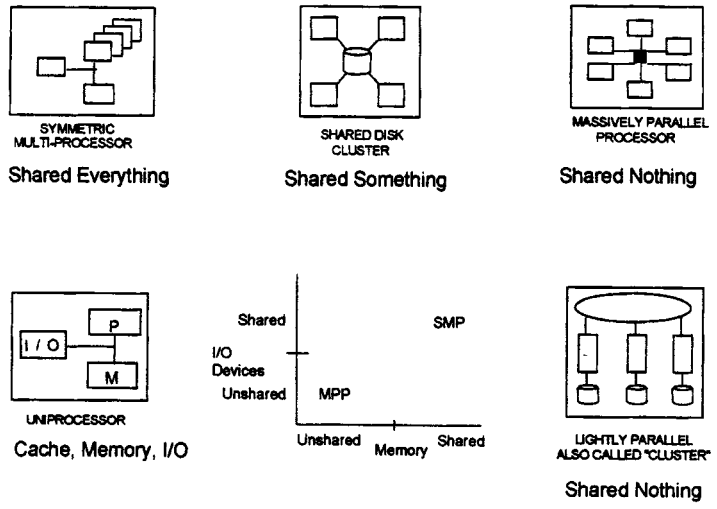
Because this simplification can lead to misunderstandings, it is advisable to clarify these terms early in any discussion to avoid the pitfalls of oversimplification.

How are multiprocessors distinguished from uniprocessors? The basic building block for other configurations, a uniprocessor is a single processor with a high-speed cache and a bus to access main memory and external I/O. Processors are very fast and need data at high speeds to avoid wasting valuable processing cycles. On the other hand, memory, the supplier of data, is made of slow-speed, inexpensive dynamic random access memory (DRAM) chips, which cannot keep up with the demands of these processors. Cache provides a small amount of buffered data at high speeds to the processor to balance the fast data requirements of the processor and the slow speed of DRAM memory. Cache is not as fast as a processor but much faster than main memory, and is made from fast but expensive static random access memory (SRAM) chips.

Even though SMPs, clusters, and MPPs are being produced to provide for additional processing power, vendors are also continuing to enhance the uniprocessor technology. Intel's 386, 486, Pentium, and Pro-Pentium processors exemplify this trend. The enhancements in the PowerPC line from the alliance among Apple, IBM, and Motorola is another example. Although research and development costs for each generation of these processors require major capital investment, the competitive pressures and technological enhancements continue to drive the vendors to develop faster uniprocessors, which are also useful in reducing the overall elapsed time for those tasks that cannot be performed in parallel. This is a very significant consideration and was highlighted by Amdahl's law.

An SMP is a manifestation of a *shared-everything* configuration (see Exhibit 2). It is characterized by having multiple processors, each with its own cache, but all sharing memory and I/O devices. Here, the processors are said to be tightly coupled.

EXHIBIT 2 — The Symmetric Multiprocessor



SMP configuration is very popular and has been in use for over 2 decades. IBM Systems/370 158MP and 168MP were so configured. Currently, several vendors (including COMPAQ, DEC, HP, IBM, NCR, SEQUENT, SGI, and Sun) offer SMP configurations.

A very important consideration that favors the choice of SMP is the availability of a vast amount of already written software, for example, DBMS, that is based on the programming model that SMP supports. Other alternative platforms, such as loosely coupled multiprocessors, currently do not have as rich a collection of enabling software.

The fact that multiple processors exist is generally transparent to a user-application program. The transparency is enabled by the OS software layer, which hides the complexity introduced by the multiple processors and their caches, and permits increased horsepower exploitation relatively easily. It is also efficient because a task running on any processor can access all the main memory and share data by pointer passing as opposed to messaging.

The subject of how many processors can be tightly coupled together to provide for growth (i.e., scalability) has been a topic of vast interest and research. One of the inhibitors for coupling too many processors is the contention for memory access. High-speed cache can reduce the contention because main memory needs to be accessed less frequently; however, high-speed cache introduces another problem: *cache coherence.* Because each processor has its own cache, coherence among them

has to be maintained to ensure that multiple processors are not working with stale copy of data from main memory.

Conventional wisdom has held that because of cache coherence and other technical considerations, the practical upper limit for the number of processors that could be tightly coupled in an SMP configuration is between 10 and 20. However, over the years, a number of new schemes, including Central Directory, Snoopy Bus, Snoopy Bus and Switch, Distributed Directories using Scalable Coherent Interconnect (SCI), and the Directory Architecture for Shared memory (DASH), have been invented and implemented by SMP vendors to overcome the challenge of maintaining cache coherence. Even now, new innovations are being rolled out, so SMPs have not necessarily reached their scalability limits. These newer techniques have made it possible to scale well beyond the conventionally accepted limit, but at an increased cost of research and development. So, economics as well as technology now become important issues in SMP scalability. The question then is not whether SMP can scale, but whether it can scale economically, compared to other alternatives, such as loosely coupled processors (e.g., clusters and MPPs).

Clusters

A *cluster* is a collection of interconnected whole computers utilized as a single computing resource. Clusters can be configured using either the shared-something or shared-nothing concept. This section discusses the shared-something clusters, and the MPP section covers the shared-nothing clusters.

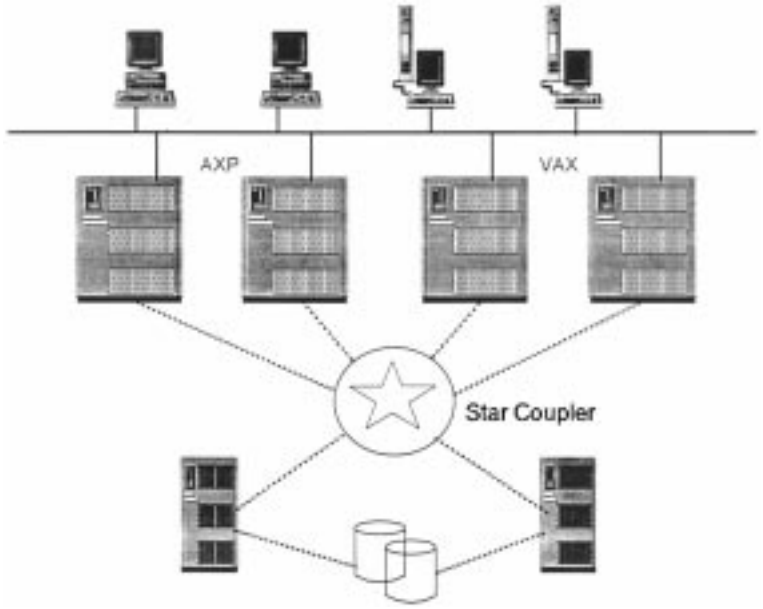
Shared-something clusters may share memory or I/O devices. Shared I/O clusters are very popular; various DEC clusters and IBM Sysplex are examples. Because of their wide use, people colloquially use the word *cluster* to mean a shared-disk configuration. However, sharing disks is not a requirement of a cluster; for that matter, no sharing is necessary at all for clustering.

Digital Equipment Corporation has offered VAXCluster since the early 1980s (see Exhibit 3), and followed it with Open VMSCluster and, more recently, DECTrueClusters.

IBM Parallel Sysplex configuration uses a shared electronic storage (i.e., a coupling facility) to communicate locking and other common data structures between the cluster members. This helps in system performance.

A shared-disk cluster (SDC) presents a single system image to a user application program. OS, DBMS, and load balancing infrastructure can hide the presence and complexities of a multitude of processors. Nonetheless, a user application program can target a specific computer if it has an affinity for it for any reason, such as the need for a specialized I/O device.

EXHIBIT 3 — The VAXCluster Configuration



Shared-disk clusters offer many benefits in the commercial marketplace, including high availability, incremental growth, and scalability. If one processor is down for either planned or unplanned outage, other processors in the complex can pick up its workload, thus increasing data availability. Additional processors can be incrementally added to the cluster to match the workload, as opposed to having to purchase a single, large computer to meet the eventual workload. Also, the scalability challenge faced by the SMP is not encountered here because the independent processors do not share common memory.

Clusters face the challenge of additional overhead caused by interprocessor message passing, workload synchronization and balancing, and software pricing. Depending on the pricing model used by the software vendor, the sum of software license costs for equivalent computing power can be higher for a cluster as compared to an SMP, because multiple licenses are required.

The Massively Parallel Processor (MPP)

In an MPP configuration, each processor has its own memory and accesses its own external I/O. This is a shared-nothing architecture, and the processors are said to be *loosely coupled*. An MPP permits the coupling of

thousands of relatively inexpensive off-the-shelf processors to provide billions of computing instructions. Because an MPP implies coupling a large number of processors, an interconnection mechanism (e.g., buses, rings, hypercubes, and meshes) is required to provide for the processors to communicate with each other and coordinate their work. Currently, many vendors, including IBM, ICL, NCR, and nCUBE, offer MPP configurations.

From a simplistic I/O and memory sharing point of view, there is a distinction between MPP and SMP architectures. However, as will be discussed later in the article, operating systems (OSs) or other software layers can mask some of these differences, permitting some software written for other configurations, such as shared-disk, to be executed on an MPP. For example, the virtual shared-disk feature of IBM's shared-nothing RS/6000 SP permits higher-level programs, for example Oracle DBMS, to use this MPP as if it were a shared-disk configuration.

Before the mid-1980s, the use of MPPs was essentially limited to scientific and engineering work, which typically requires a large number of computing instructions but limited I/O. Such a task can often be split into many individual subtasks, each executing on a processor and performing minimal I/O. Computational fluid dynamics is an example of this. Subtask results are then combined to prepare the final response. The decomposition of tasks and combination of results required sophisticated programming tools and skills that were found in the scientific and engineering community.

Traditional commercial work, on the other hand, requires relatively fewer computing cycles (though this is changing) but higher I/O bandwidth. In addition, typical programming tools and required skills for exploiting MPPs in commercial environments were not available. The combination of these factors resulted in minimal use of this technology in the commercial arena.

In the late 1980s, offerings introduced by NCR's DBC1012 Teradata data base machine started to permit online analytical processing (OLAP) of commercial workloads to benefit from the MPP configurations. More recently, innovations introduced by several data base vendors, such as Oracle and Informix, have further accelerated this trend.

An important point is that a key ingredient for the acceptance of MPP in the commercial marketplace is the ease with which parallelism can be exploited by a DBMS application programmer without the programmer needing to think in terms of parallel programs or needing to implement sophisticated and complex algorithms.

The technique used by the DBMSs to exploit parallelism is the same as previously mentioned for the scientific work. A single task (i.e., an SQL request) is split into many individual subtasks, each executing on a processor. Subtask results are then combined to prepare the answer set for the SQL request. In addition, the packaging of processors in an MPP configuration offers several administrative benefits. The multitude of pro-

cessors can be managed from a single console, and distribution and maintenance of software is simplified because only a single copy is maintained. The system makes the same copy available to the individual nodes in the MPP. This administrative benefit is very attractive and at times used for justification of an MPP solution, even when no parallelism benefits exist.

At times, financial requests for additional hardware may also be less of an administrative concern, because the upgrade (i.e., the addition of processors and I/Os) is made to an existing installed serial number machine and does not require the acquisition of a new one, which may involve more sign-offs and other administrative steps.

Currently, DBMS vendors focus on MPP technology to implement strategic information processing (e.g., data warehouse) style of applications. They have placed only a limited emphasis on operational applications. As more experience is gained and technology matures, it is likely that more operational applications, such as OLTP, may find a home on the MPP platform.

However, the current reduction in government grants to scientific and defense organizations, the slow development of the software, and fierce competition in the MPP marketplace have already started a shakedown in the industry. One analysis by the *Wall Street Journal* has suggested that by the time the commercial marketplace develops, quite a few suppliers will "sputter toward the abyss."

In addition to MPP, the shared-nothing configuration can also be implemented in a cluster of computers where the coupling is limited to a low number, as opposed to a high number, which is the case with an MPP. In general, this shared-nothing lightly (or modestly) parallel cluster exhibits characteristics similar to those of an MPP.

The distinction between MPP and a lightly parallel cluster is somewhat blurry. The following table shows a comparison of some salient features for distinguishing the two configurations. The most noticeable feature appears to be the arbitrary number of connected processors, which is large for MPP and small for a lightly parallel cluster.

Characteristic	MPP	Lightly Parallel Cluster
Number of processors	Thousands	Hundreds
Node OS	Homogeneous	Can be different, but usually homogeneous
Inter-node security	None	Generally none
Performance metric	Turnaround time	Throughput and turnaround

From a programming model perspective, given the right infrastructure, an MPP and shared-nothing cluster should be transparent to an application, such as a DBMS. As discussed in a later section, IBM's DB2 Common Universal Server DBMS can execute both on a shared-nothing,

lightly parallel cluster of RS/6000 computers and on IBM's massively parallel hardware, RS/6000 SP.

Lightly parallel, shared-nothing clusters may become the platform of choice in the future for several reasons, including low cost. However, software currently is not widely available to provide a single system image and tools for performance, capacity, and workload management.

It is expected that Microsoft Corp.'s architecture for scaling Windows NT and SQL Server to meet the enterprisewide needs may include this style of lightly parallel, shared-nothing clustering. If this comes to pass, the shared-nothing clusters will become very popular and overshadow their big cousin, MPP.

The various hardware configurations also offer varying levels of scalability (i.e., how much useable processing power is delivered to the users when such additional computing resources as processors, memory, and I/O are added to these configurations). This ability to scale is clearly one of the major considerations in evaluating and selecting a multiprocessor platform for use.

MULTIPROCESSORS: SCALABILITY AND THROUGHPUT

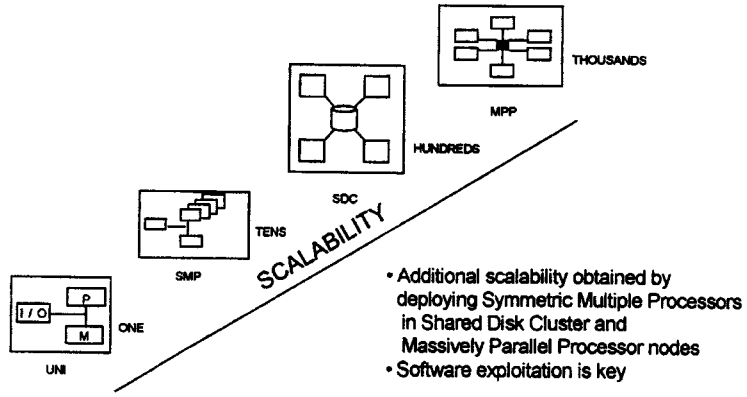
An installation has several hardware options when requiring additional computational power. The choice is made based on many technical, administrative, and financial considerations. From a technical perspective alone, there is considerable debate as to how scalable the various hardware configurations are.

One commonly held opinion is that the uniprocessors and massively parallel processors represent the two ends of the spectrum, with SMP and clusters providing the intermediate scalability design points. Exhibit 4 depicts this graphically.

As discussed in the previous section, a commonly held opinion is that an SMP can economically scale only to between 10 and 20 processors, beyond which alternative configurations, such as clusters and MPPs, become financially more attractive. Whether that is true depends on many considerations, such as hardware technology, software availability and pricing, and technical marketing and support from the vendors. Considering all the factors, one can observe that hardware technology, by itself, plays only a minor role.

SMPs currently enjoy the most software availability and perhaps a software price advantage. Shared-disk clusters, in general, suffer the penalty from higher software prices because each machine is a separate computer requiring an additional license, although licenses are comparatively less expensive, as the individual machines are smaller than a corresponding equivalent single SMP. However, new pricing schemes based on total computing power, number of concurrent logged-on users, or other fac-

EXHIBIT 4 — Scalability



tors are slowly starting to alter the old pricing paradigm, which put clusters at a disadvantage.

Clusters and MPPs do not suffer from the cache coherence and memory contention that offer SMP design challenges in scaling beyond the 10-to-20 range. Here, each processor has its own cache, no coherence needs to be maintained, and each has its own memory, so no contention occurs. Therefore, from a hardware viewpoint, scaling is not as challenging. The challenge lies, however, in interconnecting the processors and enabling software so that the work on the separate processors can be coordinated and synchronized in an efficient and effective way.

To provide more computing resources, vendors are including SMP as individual nodes in cluster and MPP configurations. NCR's WorldMark 5100M is one example in which individual nodes are made of SMPs. Thus, the achievement of huge processing power is a multitiered phenomenon: increasing speeds of uniprocessors, the combination of faster and larger number of processors in an SMP configuration, and inclusion of SMPs in cluster and MPP offerings.

SOFTWARE LAYERS

All the approaches to parallelism discussed to this point have touched on multiprocessing. All the approaches manifest and support parallelism in different ways; however, one underlying theme is common to all. It may not be very obvious, but it needs to be emphasized. In almost all cases, with few exceptions, a commercial application program is written in a sequential fashion, and parallelism is attained by using some external mechanism. Here are some examples to illustrate the point.

During execution, an OLTP manager spawns separate threads or processes, schedules clones of the user's sequential program, and manages storage and other resources on its behalf to manifest parallelism. In batch, multiple job streams executing an application program are often organized to processes different files or partitions of a table to support parallelism. In the client/server model, multiple clients execute the same sequential program in parallel; the mechanism that enables parallelism is the software distribution facility. Another way of looking at client/server parallelism is the execution of a client program in parallel with an asynchronous stored procedure on the server; the facility of remote procedure calls and stored procedures enables parallelism.

In the same way, a single SQL call from an application program can be enabled to execute in parallel by a DBMS. The application program is still written to execute sequentially, requiring neither new compilers nor programming techniques. This can be contrasted with building parallelism within a program using special constructs (e.g., ~~doall~~ and ~~foreach)~~, in which the FORTRAN compiler generates code to execute subtasks in parallel for the different elements of a vector or matrix. Such constructs, if they were to become widely available in the commercial languages, will still require significant retraining of the application programmers to think of a problem solution in terms of parallelism. Such facilities are not available. In addition, from an installations point of view, the approach of sequential program and DBMS-enabled parallelism is much easier and less expensive to implement. The required new learning can be limited to the designers and supporters of the data bases: data base and system administrators. Similarly, tools acquisition can also be focused toward the task performed by such personnel.

Now, SQL statements exhibit varying amounts of workload on a system. At one extreme, calls (generally associated with transaction-oriented work) that retrieve and update a few rows require little computational resource. At the other extreme, statements (generally associated with OLAP work) perform large amounts of work and require significant computational resources. Within this wide range lie the rest. By their very nature, those associated with OLAP work, performed within a data warehouse environment, can benefit most from the intra-SQL parallelism, and those are the current primary target of the parallelism.

If the user-application code parallelism is attained only through the use of other techniques listed earlier (e.g., OLTP), it is questionable whether DBMS-enabled parallelism is limited to the OLAP data warehouse-oriented SQL. This is really not valid. New enhancements to the relational data base technology include extensions that permit user-defined functions and data types to be tightly integrated with the SQL language. User-developed application code can then be executed as part of the DBMS using these facilities. In fact, Oracle7 allows execution of business logic in parallel using user-defined SQL functions.

DBMS ARCHITECTURE

As discussed earlier, from an application programming perspective, exploitation of hardware parallelism in the commercial marketplace was limited because of lack of tools and skills. It is now generally recognized that DBMS vendors have recently stepped up their efforts in an attempt to address both of these challenges. They are starting to become the enablers of parallelism in the commercial arena. Additionally, more and more applications are migrating to DBMS for storage and retrieval of data. Therefore, it is worthwhile to understand how the DBMS enables parallelism.

This understanding will help in choosing an appropriate DBMS and, to some extent, the hardware configuration. Also, it will help in designing applications and data bases that perform and scale well. Lack of understanding can lead to poorly performing systems and wasted resources.

In a manner similar to the three hardware configurations, DBMS architectures can also be classified into three corresponding categories:

- Shared data and buffer
- Shared data
- Partitioned data

There is a match between these architectures and the characteristics of the respective hardware configuration, (SMP, shared disk clusters, and MPP), but a DBMS does not have to execute only on its corresponding hardware counterpart. For example, a DBMS based on shared-data architecture can execute on a shared-nothing MPP hardware configuration.

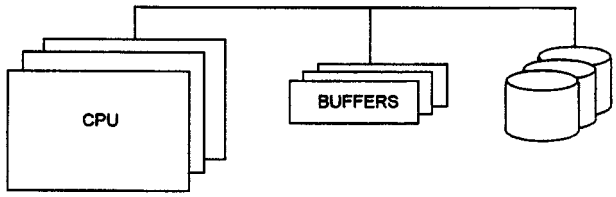
When there is a match, the two build on each other's strengths and suffer from each other's weaknesses. However, the picture is much cloudier when a DBMS executes on a mismatched hardware configuration. On the one hand, in these cases, the DBMS is unable to build on and fully exploit the power of the hardware configuration. On the other hand, it can compensate for some of the challenges associated with using the underlying hardware configuration.

Shared Data and Buffer

In this architecture, a single instance of the DBMS executes on a configuration that supports sharing of buffers and data. Multiple threads are initiated to provide an appropriate level of parallelism. As shown in Exhibit 5, all threads have complete visibility to all the data and buffers.

System administration and load balancing are comparatively easier with this architecture because a single instance of the DBMS has full visibility to all the data. This architecture matches facilities offered by SMP configuration, in which this architecture is frequently implemented.

EXHIBIT 5 — Shared Data and Buffers



Characteristics:
- Data Partitioning not needed
- Systems management & load balancing easier
- Scalability concerns

Examples:
- Informix PDQ
- Oracle7 Parallel Query
- DB2 (MVS) CPU Parallelism

When executed on an SMP platform, the system inherits the scalability concerns associated with the underlying hardware platform. The maximum number of processors in an SMP, the cache coherence among processors, and the contention for memory access are some of the reasons that contribute to the scalability concerns.

Informix DSA, Oracle7, and DB2 for MVS are examples of DBMSs that have implemented the shared-data-and-buffer architecture. These DBMSs and their SQL parallelizing features permit intra-SQL parallelism; that is, they can concurrently apply multiple threads and processors to process a single SQL statement.

The algorithms used for intra-SQL parallelism are based on the notion of program and data parallelism. Program parallelism allows such SQL operations as scan, sort, and join to be performed in parallel by passing data from one operation to another. Data parallelism allows these SQL operations to process different pieces of data concurrently.

Shared Data

In this architecture, multiple instances of the DBMS execute on different processors. Each instance has visibility and access to all the data, but every instance maintains its own private buffers, and updates rows within it.

Because the data is shared by multiple instances of the DBMSs, a mechanism is required to serialize the use of resources so that multiple instances do not concurrently update a value and corrupt the modifications made by another instance. This serialization is provided by the global locking facility of the DBMS and is essential for the shared data architecture. Global locking may be considered an extension of DBMS

local locking facilities, which ensures serialization of resource modification within an instance, to the multiple instances that share data.

Another requirement, buffer coherence, is also introduced, because each instance of the DBMS maintains a private buffer pool. Without buffer coherence, a resource accessed by one DBMS from disk may not reflect the modification made by another system if the second system has not yet externalized the changes to the disk. Logically, the problem is similar to cache coherence, discussed earlier for the SMP hardware configuration.

Combined, global locking and buffer coherence ensure data integrity. Oracle Parallel Server (OPS), DB2 for MVS Data Sharing, and Information Management System/Virtual Storage (IMS/VS) are examples of DBMSs that implement this architecture.

It must be emphasized that intraquery parallelism is not being exploited in any of these three product feature examples. The key motivation for the implementation of this architecture is the same as those discussed for the shared-disk hardware configuration, namely: data availability, incremental growth, and scalability. However, if one additionally chooses to use other features of these DBMSs in conjunction, the benefits of intraquery parallelism can also be realized.

As can be seen, there is a match between this software architecture and the facilities offered by shared-disk clusters, where this architecture is implemented. Thus, the performance of the DBMS depends not only on the software but also on the facilities provided by the hardware cluster to implement the two key components: global locking and buffer coherence.

In some implementations, maintenance of buffer coherence necessitates writing of data to the disks to permit reading by another DBMS instance. This writing and reading is called *pinging*. Depending on the locking schemes used, pinging may take place even if the two instances are interested in distinct resources but represented by the same lock. This is known as false pinging. Heavy pinging, false or otherwise, has a negative effect on application performance because it increases both the I/O and the lock management overhead.

System vendors use many techniques to minimize the impact of pinging. For example, IBM's DB2 for MVS on the Parallel Sysplex reduces the pinging I/O overhead by using the coupling facility, which couples all the nodes in the cluster through use of high-speed electronics. That is, disk I/Os are replaced by writes and reads from the coupling facility's electronic memory. In addition, hardware facilities are used to invalidate stale data buffers in another instances.

Even if the hardware and DBMS vendors have provided adequate facilities to minimize the adverse impact of pinging, it is still an application architect and data base administrator's responsibility to design applica-

tion and data bases such that the need for sharing data is minimized to get the best performance.

It is interesting to note that shared-data software architecture can be implemented on hardware platforms other than shared disk. For example, Oracle Parallel Server, which is based on shared-data software architecture, is quite common on IBM's RS 6000 SP, an implementation based on shared-shared-nothing hardware configuration. This is achieved by using the virtual shared disk feature of RS/6000 SP.

In this case, Oracle7's I/O request for data residing on another node is routed by RS/6000 SP device drivers to the appropriate node, an I/O is performed, if necessary, and data is returned to the requesting node. This is known as *data shipping* and contributes to added traffic on the node's interconnect hardware. The inter-node traffic is a consideration when architecting a solution and acquiring hardware.

In general, for the data base administrators and application architects, it is necessary to understand such features in detail because the application performance depends on the architecture of the DBMS and the OS layers.

Partitioned Data

As the name implies, the data base is partitioned among different instances of the DBMSs. In this option, each DBMS owns a portion of the data base and only that portion may be directly accessed and modified by it.

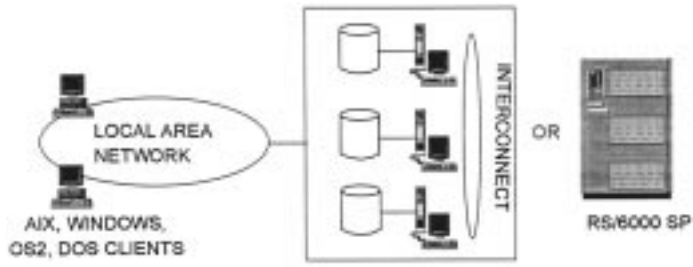
Each DBMS has its private or local buffer pool, and as there is no sharing of data, the kind of synchronization protocols discussed earlier for shared data (i.e., global locking and buffer coherence) are not required. However, a transaction or SQL modifying data in different DBMS instances residing on multiple nodes will need some form of two-phase commit protocol to ensure data integrity.

Each instance controls its own I/O, performs locking, applies the local predicates, extracts the rows of interest, and transfers them to the next stage of processing, which may reside on the same or some other node.

As can be seen, there is a match between the partitioned-data option and the MPP hardware configuration. Additionally, because MPP provides a large amount of processing power, and the partitioned-data architecture does not need the synchronization protocols, some argue that this combination offers the highest scalability. Thus, it has been the focus of recent development in the DBMS community. The partitioned-data architecture requires frequent communication among the DBMS instances to communicate messages and transfer results. Therefore, low latency and high bandwidth for the interconnect are required if the system is to scale up with the increased workload.

As mentioned earlier, NCR's Teradata system was one of the earliest successful commercial products based on this architecture. Recent new

EXHIBIT 6 — DB2 Parallel Edition as a Partitioned-Data Implementation



UNIX DBMSs offerings from other vendors are also based on this architecture. IBM DB2 Parallel Edition, Informix XPS, and Sybase MPP are examples.

In this architecture, requests for functions to be performed at other DBMS instances are shipped to them; the requesting DBMS instance receives only the results, not a block of data. This concept is called *function shipping* and is considered to offer better performance characteristics, compared to data shipping, because only the results are transferred to the requesting instance.

The partitioned-data architecture uses the notion of data parallelism to get the benefits of parallelism, and data partitioning algorithms play an important role in determining the performance characteristics of the systems. Various partitioning options are discussed in a later section.

The partitioned-data architecture also provides an additional flexibility in choosing the underlying hardware platform. As one can observe, partitioned data matches well with the MPP hardware configuration. It also matches with the shared-nothing lightly parallel clusters. The distinction between these clusters and MPP is primarily based on the number of nodes, which is somewhat arbitrary.

For illustration, DB2 Parallel Edition is considered a partitioned-data implementation. As shown in Exhibit 6, Parallel Edition can execute both on RS/6000 SP, an MPP offering, and on a cluster of RS/6000 computers, which are connected by a LAN. However, for a number of technical and financial reasons, the RS/6000 cluster solution is not marketed actively. On the other hand, there are conjectures in the market that similar system solutions are likely to become more prevalent when Microsoft becomes more active in marketing its cluster offerings.

THE THREE DBMS ARCHITECTURES: SUMMARY

There is great debate over which of the three software models — shared buffer and data, shared data, or partitioned data — is best for the com-

mercial marketplace. This debate is somewhat similar to the one that revolves around the choice of hardware configurations (i.e., SMP, clusters, or MPP). Although one might assume that making the choice of a software architecture would lead to a straightforward choice of a corresponding hardware configuration, or vice versa, this is not the case.

OS and infrastructure layers permit cohabitation of a DBMS architecture with a non-matching hardware configuration. Because of the mismatch, it is easy to observe that the mismatched components may not fully exploit the facilities of its partner. It is somewhat harder to appreciate, however, that the shortcomings of one may be compensated to some extent by the strengths of another. For example, a shared-data software architecture on an MPP platform avoids the management issues associated with repartitioning of data over time as the data or workload characteristics change.

This variety and flexibility in mix-and-match implementation presents tradeoffs to both the DBMS and hardware vendors and to the application system developers. Even after an installation has made the choice of a DBMS and a hardware configuration, the application architects and the data base and system administrators must still have a good understanding of the tradeoffs involved with the system components to ensure scalability and good performance of the user applications.

CONCLUSION

The future seems extremely promising. Making faster and faster uniprocessors is not only technically difficult but is becoming economically prohibitive. Parallel processing is the answer. In the near future, all three hardware configurations are likely to find applicability:

- SMP on desktops and as departmental servers
- Shared-disk SMP clusters as enterprise servers
- MPPs as servers of choice for strategic information processing and as multimedia servers

Image and video servers requiring large I/O capacity seem ideal for parallel processing. This need can be satisfied by a variety of current MPP, which emphasize scalability of I/O more than the scalability of processing power.

Lightly parallel shared-nothing clusters based on commodity hardware and ATM or Ethernet interconnect are likely to become popular because of their low cost as soon as software for workload management becomes available.

Making long-range predictions in this business is unwise. However, one can be assured that parallel processing solutions in the commercial marketplace will be driven less by the hardware technology, and more

by the software innovations, systems management offerings, pricing schemes, and most important by the marketing abilities of the vendors.

Prem N. Mehra is a senior architectural engineer with Microsoft Corp. and a former associate partner in the technology integration services-worldwide organization of Andersen Consulting. This article is adapted from Auerbach's forthcoming book, *Network-Centric Computing: Computing, Communications, and Knowledge,* by Hugh W. Ryan and associates.