AIXpert DIGITAL VERSION IBM

TABLE OF CONTENTS

FEBRUARY
1995

AIXpert BACK TO MAIN MENU

"I Write the Songs..."

In the last issue (November 1994) we discussed Symmetric Multiprocessing (SMP) from the viewpoint of the designers who made the components play together in the "SMP Band." In this issue, we hear from the solution developers who are "writing the songs" for the SMP Band to play. These developers, who saw the advantages of the SMP architecture, are writing their programs to capitalize on those advantages.

Because database programs are among the most demanding applications for computer resources, it is not surprising that the first SMP-enabled applications to appear are database programs. We asked the database developers to tell us about the SMP design experience and how their programs exploit the SMP architecture to satisfy their customer's needs. This issue includes five articles from some major database developers to give you a taste of what it's like to work in the SMP environment.

If you're interested in OOP, you will want to read the detailed description of the OO tools environment used in a large OO project in IBM's Rochester programming lab ("Building a Comprehensive OO Development Environment on AIX"). And be sure to look at "Building Object-Oriented Frameworks"—Part 1 of a two-part article from Taligent (Part 2 will appear in the next issue of *AIXpert*).

Because it discusses features and options of the XL Fortran compiler, Fortran programmers may find the article about porting from a Cray® computer to be useful even if they are not currently running on Cray hardware. System administrators should read the TTY subsystem article, and hardware mavens can brush up on POWER2 performance and learn about the SystemGuard™ processors in the new SMP machines from IBM. Anyone who writes code for, or administers a network should also take a look at the NetView Association article.

What's New

In this issue we are inaugurating a new feature that we hope will be helpful in your work with AIX. In the PowerNotes column we will try to keep you up to date on the latest tips and news for AIX® developers. Be sure to read that column and let us know how you like it.

We've heard from readers who were tired of ripping off the cover of their *AIXpert* so that their friends could submit a subscription. So, we are introducing a new subscription form in this issue. You can tear out the form without damaging the magazine. Use the convenient self-mailer to either subscribe or give us your comments about the magazine.

George Noren

George Noren, IBM Corporation, Internal Zip 2830, 11400 Burnet Road, Austin, TX 78758. Internet: geo@austin.ibm.com. Since joining IBM in September 1979, Mr. Noren has written manuals for System/34, System/36™, and AIX on both the RT® and RISC System/6000® platforms, and was a member of the InfoExplorer™ design team. He has also worked as system administrator for several AIX server machines and their clients, and is currently responsible for the Prototype Evaluation Labs in Austin. Mr. Noren studied engineering at Illinois Institute of Technology, holds a BA in English from the University of Minnesota and an MBA from St. Edwards University in Austin.

George Noren

DB2 for AIX

By Gene Kligerman and Ming Wu

This article describes the capabilities of IBM DATABASE 2™ for AIX (DB2 for AIX) in uniprocessor and SMP environments. It highlights the enhancements incorporated into Version 2.1, provides an introduction to the DB2 process architecture, and discusses performance issues.

DB2® is an integrated family of Relational Database Management Systems (RDBMSs) for a variety of platforms, including MVS™, VM, VSE, OS/400®, OS/2®, AIX, Sun® Solaris®, and HP/UX. IBM plans to support additional platforms in the future.

Introduction

IBM brings the dependability, integrity, and proven design of DB2 to the UNIX® environment. DB2 for AIX delivers reliable performance, high availability, outstanding recoverability features, and seamless access to DB2 on the host through the DB2 Distributed Database Connection.

DB2 for AIX has many features to help customers fine-tune application performance while ensuring data integrity: a cost-based optimizer, stored procedures, row blocking, compound SQL, multiple levels of concurrency, and a granular locking scheme.

Advanced recovery features protect data, even between backups. High availability is achieved through online backup. DB2 for AIX delivers full transaction support, ensuring that transactions are completed normally before changing any data. Row-level locking and declarative referential integrity contribute to even safer data handling.

DB2 for AIX supports an SQL that is compatible with DB2 for MVS. Since it also provides an interface that is similar to DB2 on other platforms, DB2 applications and skills are easily transferred to a client-server environment. DB2 for AIX also supports a wide variety of applications running on DOS, Windows™, OS/2, AIX, HP/UX, and Solaris clients. New applications can be developed on these clients using DB2 software developers' kits, which have a variety of languages and access methods.

Both DB2 Version 1 and Version 2 support AIX Versions 3.2.4, 3.2.5, and 4.1.1 (in uniprocessor and SMP environments). Because Symmetric Multiprocessor (SMP) hardware supported by AIX became available after DB2 Version 1 and before Version 2, the implementation of SMP support differs slightly between the two DB2 versions.

DB2 Version 1 Support of AIX 4.1

DB2 Version 1 was architected as a multiprocessing application, with each asynchronous unit of activity implemented as an AIX process.

DB2 creates AIX processes called *database agents* that each manage one database connection. The DB2 Database Administrator (DBA) can specify a maximum number of database agents (the total number of users that can connect to the database) and the maximum number of concurrent agents (the number of agents that can actively execute SQL statements simultaneously). If more connections are attempted than the maximum value allows, the additional agents must wait. This enables a DBA to control the use of system resources.

Since DB2 was implemented as a multiprocessing application from the beginning, it was not necessary to redesign the product to support the new AIX SMP environment.

When DB2 Version 1 was first released in October 1993, it supported AIX 3.2.4, AIX 3.2.5, and uniprocessor IBM RISC System/6000s available at the time. Providing support for the new SMP machines has two implications.


Ming Wu


Gene Kligerman

- DB2 must support AIX 4.1.1 in both uniprocessor and SMP environments.
- DB2 must use the new AIX 4.1.1 operating system services to enable it to run safely in an SMP environment.

Supporting AIX 4.1.1 was easy because DB2 uses only standard AIX interfaces that are highly compatible with AIX 3.2.5. IBM verified DB2's support for AIX 4.1 in a uniprocessor environment by successfully running a full regression test.

The main advantage of IBM's SMP architecture is the direct portability of most software developed for uniprocessors. Few changes were required for DB2 Version 1 to support the SMP environment. Care was taken to ensure that SMP-enabling of DB2/6000™ did not affect the performance of the uniprocessor DB2 installations. The multiprocessing architecture of DB2 Version 1 provided a ready-made basis for exploiting SMP.

Two changes were necessary for Version 1 of DB2 to support SMP hardware under AIX 4.1:

- The code was modified to use the new latching services that were introduced in AIX 4.1 to serialize access to shared DB2 data structures.
- The DB2 installation process was modified to ensure that the code installed was appropriate for the level of operating system (AIX Version 3.2.4, 3.2.5, or 4.1.1) and hardware (uniprocessor or SMP).

DB2 for AIX Version 2.1

DB2 for AIX Version 2, with improved features and performance enhancements, is accompanied by a rich suite of tools for database administration and performance tuning. At the time this article is being written (January 1995), DB2 for AIX Version 2.1 is in beta testing with customers worldwide. DB2 for AIX Version 2 comes as a single binary that supports all levels of AIX—from 3.2.4 to 4.1.1 and beyond—ensuring that all users can benefit equally from the DB2 for AIX Version 2 performance and functional enhancements.

Highlights of Version 2 are described in the following sections.

Innovative Multimedia and Object-Oriented Applications

Extensions to the relational capability of DB2 enable multimedia objects to be manipulated and object-oriented programming techniques to be used to increase programmer productivity and data integrity. These extensions include the following.

- User-Defined Functions (UDFs), User-Defined Triggers (UDTs), check constraints, and Large Objects (LOBs)
- The ability to write recursive SQL queries so that a single SQL statement can replace complex application logic in applications such as bills-of-material processing and path expression queries

Flexible Management of Large Databases

Managing very large databases (limited only by the size of physical storage) is easier when a database can be partitioned into separately managed parts called *tablespaces*. DB2 Version 2 has enhancements for large database administration and improved database availability, including the following:

- Ability to place tables, indexes, and LOBs into different tablespaces, and flexibility in assigning storage devices to tablespaces, enabling administrators to maximize database availability and performance
- Support for online or offline backup and recovery of all or part of the database using tablespaces
- Support for backup and recovery utilities using several devices in parallel, reducing the time for these activities
- A high-speed load utility to perform bulk loading of data into tables

A key advantage of IBM's SMP architecture is the direct portability of most software developed for uniprocessors.

Distributed Applications and Data Replication

With DB2 for AIX Version 2.1 and accompanying products, the application developer and database administrator have great flexibility in distributing data and applications across various workstation and host platforms. Enhancements in this area include the following:

- Support for Distributed Relational Database Architecture™ Application Server (DRDA™ AS) to provide access to DB2 databases on AIX and OS/2 from MVS, VM, and OS/400 applications
- Support for data replication products such as Data Propagator™ Non-Relational, Data Propagator Relational, and DataRefresher™, enabling extensive data replication facilities from DB2 for MVS, DB2/400™, IMS, and VSAM to AIX and OS/2 DB2 databases
- Distributed Unit of Work (DUOW), also known as two-phase commit support

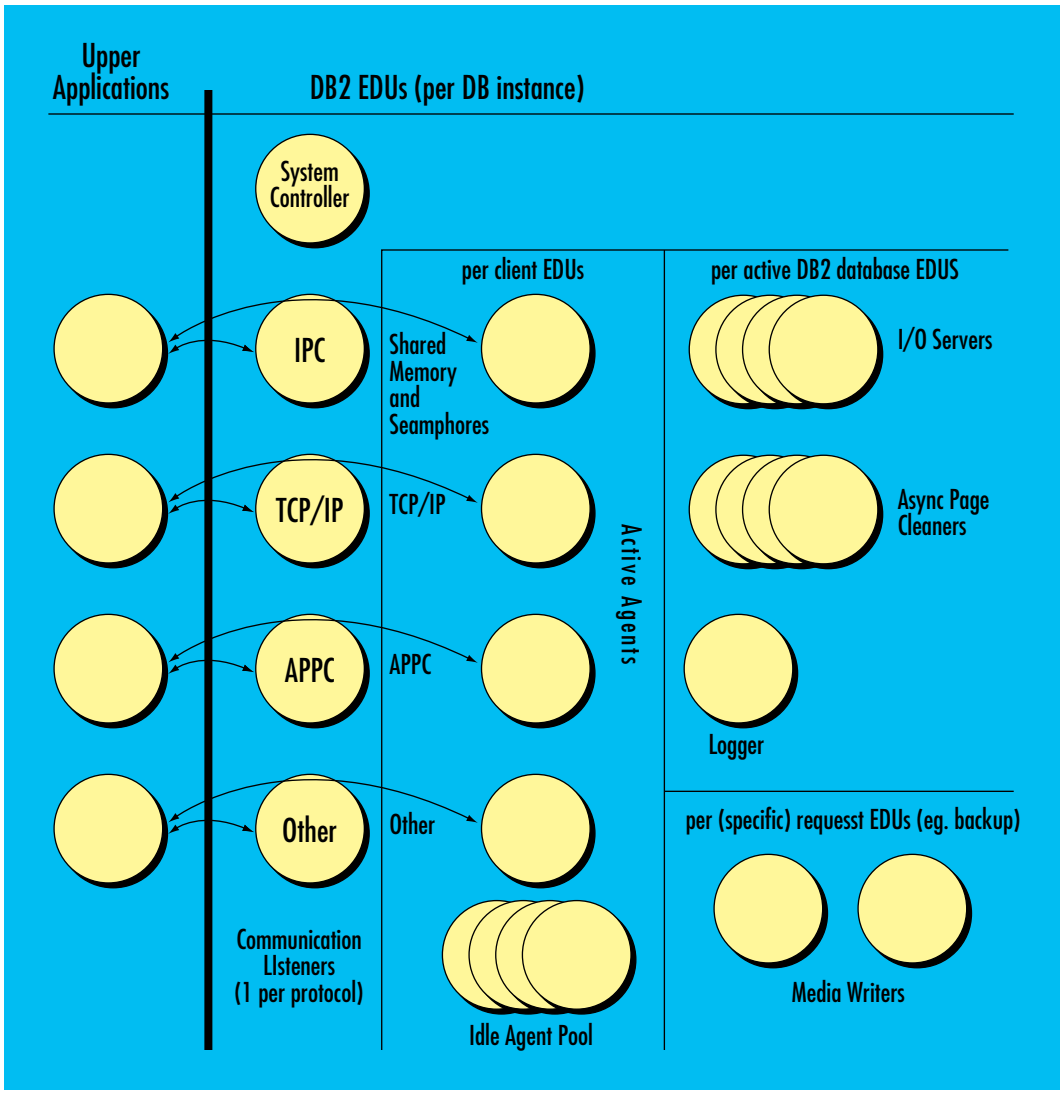


Figure 1. The DB2 process model architecture

Performance

The new SQL optimizer architecture uses sophisticated techniques to transform complex queries into their most efficient form and to accurately determine the best access path to data. Support for I/O prefetch and parallel I/O further improves the performance of Online Transaction Processing (OLTP) applications, decision-support applications, and database utilities.

DB2 Version 2 Architecture

The architecture of DB2 Version 2 incorporates Engine Dispatchable Units (EDUs), the fundamental dispatching unit for all asynchronous activities performed by the DB2 database engine.

EDUs are a logical concept. Physically, they are implemented as either processes or threads depending on the platform. In the OS/2 environment, EDUs are implemented as threads within a single process. In the AIX environment, EDUs are implemented as AIX processes. Although AIX 4.1 provides thread support, the need to support DB2 under AIX 3.2 (which does not have thread support) dictates the use of processes in DB2 for AIX Version 2. Threads will be used in the future on thread-efficient platforms.

The DB2 engine creates different types of EDUs for specific tasks.

- **System controller**: The master EDU controls the starting and stopping of DB2 and owns the shared resources.
- **Agents**: An agent EDU is assigned to every connected client application. A pool of idle agents can be defined. When a new database connection is made, one of these idle agents is dispatched to handle the connection. Using idle agents reduces the connect time of applications to the database.
- **Asynchronous page cleaners**: EDUs off-load buffer page writing operations from the agent EDU executing the SQL query. This eliminates the need for the agent EDU to wait synchronously until modified pages are written from the buffer to disk to make room for query data. Using asynchronous page cleaners is particularly advantageous for user environments in which the arrival of database requests is irregular. The page cleaners take advantage of lulls in database activity to flush buffer pages to disk in preparation for future requests.
- **I/O servers**: These EDUs perform parallel I/O and big-block reads, which can dramatically improve performance in decision-support environments. For example, if DB2 is performing a table scan, I/O server EDUs will be used to prefetch multiple pages in a single request (sequential prefetch). I/O server EDUs are also invoked by the DB2 index manager to prefetch data based on an index sequence (index prefetch). The big-block read capability (reading several disk pages using a single I/O request) can reduce the CPU overhead and improve query response time.
- **Media writers**: The capability to perform backup and recovery using multiple devices in parallel is built into the DB2 engine. DB2 dispatches media writer EDUs to perform the backup and recovery operations in parallel, dramatically reducing the elapsed time to perform these functions. The high-speed load utility also uses media writer EDUs to store data directly onto disk, bypassing all the processing normally associated with SQL `INSERT` statements.
- **Communication listeners**: One communication listener EDU is assigned to each type of communications protocol supported by DB2, such as Interprocess Communications (IPC) for local clients, and Advanced Program-to-Program Communications (APPC), TCP/IP, and others for remote clients.
- **Database logger**: The logger EDU writes a log of the changes to the database data performed by agents for user applications. There is one logger EDU for each DB2 database controlled by the DB2 instance.

DB2 has no concept of checkpoints. Unlike other open systems databases, DB2 does not have performance degradations at periodic intervals to flush buffer pages to disk and to commit the changes to the log. All committed transactions are guaranteed to be recoverable, preserving the Atomicity, Consistency, Isolation, and Durability (ACID) transaction properties of the DB2 database.

Figure 1 illustrates the DB2 process model architecture. The circles represent EDUs (processes under AIX). A DB2 instance is a single system that controls one or more DB2 databases. The System Controller is responsible for creating all DB2 EDUs, including communication listeners, media writers, and idle agents.

EDUs are the fundamental dispatching unit for all asynchronous activities performed by the DB2 database engine.

When a user application attempts to connect to a DB2 database, an appropriate communication listener will detect the conversation attempt, and assign one of the idle agents (or create a new agent) to own the connection and perform database work.

The database is inactive (no runtime resources assigned to it) until at least one user is connected to it. The System Controller activates a database at the time of the first connect to it by creating the EDUs associated with a database and assigning resources to it.

Each DB2 database has a logger EDU and a configurable number of I/O servers and asynchronous page cleaners. All database EDUs and allocated memory are released when the last application disconnects from the database.

Media writer EDUs are created when a specific task needs to be done, such as backup or recovery, which differs from other DB2 EDUs.

DB2 Performance Results

In January 1995, IBM announced the completion of industry-standard TPC-C benchmark testing of DB2 for AIX Version 2.1.

A RISC System/6000 POWERserver™ R24, which has a single POWER2 processor and DB2 for AIX Version 2.1 achieved 1,470.06 transactions per minute (tpmC) at $896.22/tpmC on the TPC-C benchmark. The transaction rate of 1,470 tpmC

tops the previous best-reported uniprocessor result by 31%, which is on IBM's RISC System/6000 POWERserver 59H with Sybase SQL Server Version 10.0.1

On the TPC-C benchmark, DB2's new result is the highest of any uniprocessor using either open-system or proprietary databases, and tops many of the multiprocessor systems.

The IBM solution of DB2 and POWERserver R24 achieves 22% better tpmC than an eight-processor Sun SPARCserver® 1000E (1,204 tpmC at $890/tpmC) using Sybase® SQL Server 10, 13% better tpmC than a four-processor AT&T® 3555/4 (1,296 tpmC at $1,040/tpmC) using INFORMIX® On-Line 5.03, and 5% better tpmC than a two-processor HP™ Model H70 (1,403 tpmC at $758/tpmC) using SQL Server 10.0.1.

DB2 Version 2 features can increase the performance of transactional and decision-support workloads in both SMP and uniprocessor environments.

These TPC-C results show the excellent transaction processing performance of DB2 for AIX Version 2.1. The test demonstrates sustained rapid response time under a heavy workload from 1,280 users accessing a large database configured with 124 gigabytes of disk. DB2 demonstrates that it can take on the challenge of managing data in very demanding environments.

DB2 for AIX Version 2.1 has been designed to exploit the capabilities of both SMP and uniprocessor architectures to deliver excellent performance and sustained scalability as more SMP processors are added to the system. (**Note**: Measurements of DB2 performance under AIX SMP show outstanding performance; however, they were not completed at press time. Therefore, official numbers were not available for this article. For additional information, contact one of the authors of this article or your IBM marketing rep.)

The AIX Journaled File System (JFS) was used to manage DB2 data on disk during the TPC-C benchmark. The excellent performance of AIX JFS in the heavy workload TPC-C scenario was instrumental in achieving the outstanding TPC-C performance results above.

For ultimate performance and flexibility, DB2 for AIX Version 2.1 users can choose between three mechanisms for managing data on disk: AIX JFS, logical volumes managed by the AIX Logical Volume Manager (LVM), or physical devices managed directly by the DB2 database.

SQL Optimizer Technology

DB2 Version 2 features can increase the performance of transactional and decision-support workloads in both SMP and uniprocessor environments. DB2 Version 2 benefits do not stop there. For a database to perform well on complex decision-support workloads, it is not enough to merely work harder by parallelizing operations, off-loading processing to ancillary processors, and relying on faster CPUs and disks. Working smarter is also essential, and this is exactly what the DB2 SQL optimizer technology is designed to do.

All of IBM's relational database offerings on all platforms have always had a cost-based SQL optimizer. The cost-based optimizer eliminates the need for tailoring the query to the operational environment. Without a cost-based optimizer, poor response times and significant unnecessary work could result.

The increasing importance of decision-support applications is resulting in more complex database queries. These queries may have been written by end users, generated by automatic tools, or produced by the many point-and-click application interfaces popular in DOS, Windows, OS/2, and UNIX environments.

The DB2 optimizer incorporates a sophisticated query rewrite phase that automatically transforms a complex query into a different query that is analyzed to generate the best possible access plan to the data. As a result, the end user does not need to be concerned with coding the optimal SQL query.

The Version 2 optimizer also looks at many alternatives in its search for the best query execution plan. In addition, it engages more sophisticated techniques of modeling the cost of different ways to fetch data from disk. All these techniques can result in an order-of-magnitude performance improvement for complex queries as compared to existing SQL optimizer technology.

The power of choosing the best access plan is not free—it takes time. To allow users the flexibility of balancing the time spent in generating the access plan versus the execution time, users can specify the class of optimization techniques that the DB2 optimizer should apply during access plan generation. Users can also specify the optimization class for the whole application or for an individual SQL statement.

Tools for Managing Performance

To assist database administrators and developers in obtaining the best possible performance from the database under a variety of operating conditions, IBM has developed two new tools: DB2 Visual Explain and DB2 Performance Monitor.

DB2 Visual Explain

An easy-to-use, graphical tool, DB2 Visual Explain provides details about the access plan

chosen by the DBMS optimizer to access data. DB2 Visual Explain supports Version 2 of DB2 for AIX with these capabilities:

- **Graphical presentation of Explain output.** Explain output, which reveals the SQL access plan chosen by the optimizer, is usually textual or tabular in nature. For most SQL statements, this output is difficult to interpret.

  DB2 Visual Explain presents this output in an easy-to-understand graphical format. Relationships between database objects such as tables and indices are instantly clear, as are various operations—such as table scans and sorts—that the optimizer has chosen for accessing the data. With this information, users can tune the structure of their databases for maximum efficiency and performance.

- **Detailed optimizer information.** DB2 Visual Explain provides optimizer information to inform the user of the cost of executing the SQL statements. This information includes I/O and CPU cost estimates for each operation, bind-time, and current table statistics.

  An administrator or developer can easily identify the most expensive operation for a given SQL statement and focus on tuning that operation. For example, tuning might consist of creating an index that would avoid the expense of a full table scan.

- **What-if modeling capability for SQL statements**. With DB2 Visual Explain, users can model the effect of various changes in the database environment on SQL statements. For example, a user can determine the estimated time needed to execute a query in a production environment with a million rows—all without the need to add data to a test environment of 100 rows.

Figure 2 shows a graphical representation of an SQL query that is optimized by DB2 to produce an access plan. The rectangles represent database tables, while the ovals represent database operations such as table scans, index scans, joins, or sorts. The graphical presentation of the access plan makes it easy to identify possible SQL problems. For example, creating an index would eliminate the requirement to do a full table scan. Users can click on any access plan object (in this case, SCAN 42) and view the access plan details associated with the operation.

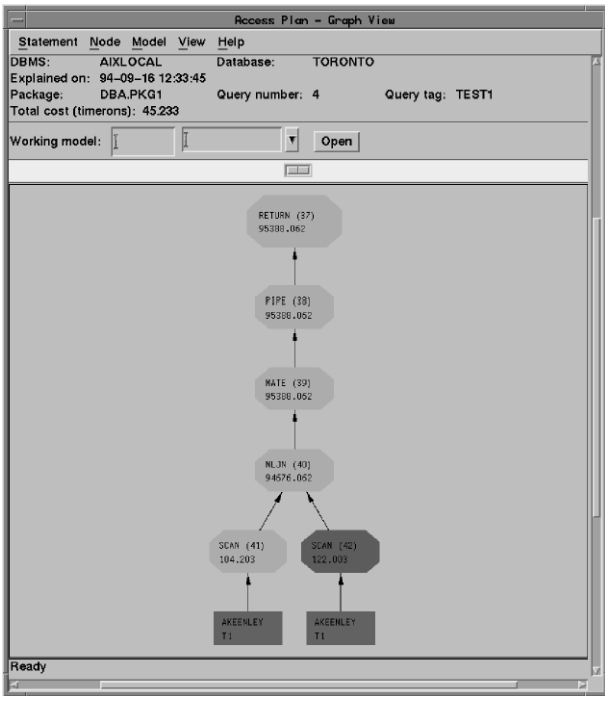


Figure 2. DB2 Visual Explain

DB2 Performance Monitor for AIX

The Performance Monitor provides comprehensive performance data collection, viewing, reporting, analysis, and alerting capabilities for Version 2 of DB2 for AIX.

- **Comprehensive, flexible data collection.** DB2 Performance Monitor for AIX supports more than 200 performance attributes including buffer pool, lock and deadlock, sorting, communication, agent, and logging information. Data is shown for database managers, databases, tablespaces, tables, connections, transactions, and statements.

  Support for two types of performance data, snapshot and event, also provides flexibility. Snapshot data measures performance characteristics at times, while event data summarizes performance attributes for a certain duration (for example, from the time a statement begins executing to the time it completes).

  For a statement event, for example, the DB2 Performance Monitor can monitor the start and stop time, total CPU time, total number and CPU time of sorts, rows read and written, and the SQLCA for the statement. For dynamic statements, the SQL statement text is also captured.

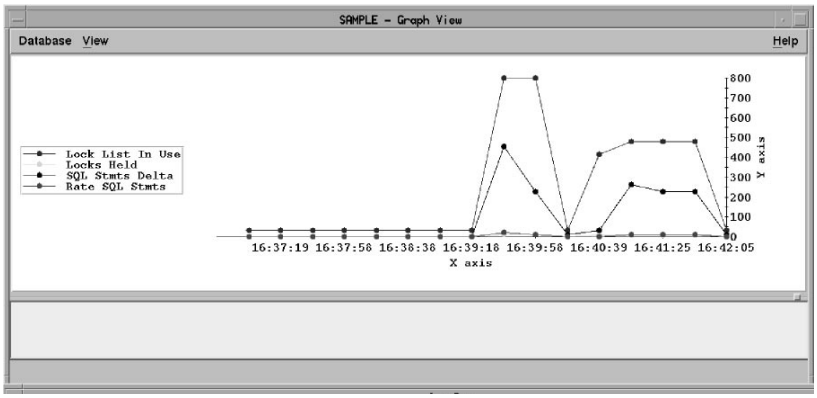


Figure 3. Performance Monitor

- **Easy-to-use, intuitive viewing and reporting.** Collected data can be viewed in real time and recorded for later playback and analysis. Recorded data can also be loaded into an SQL performance database for further querying and reporting.

  When viewing data, users have the option of graphical or textual presentation. Graphical User Interface (GUI) views help users interpret complex data, while textual output provides information for reporting and archival purposes.

- **Powerful data analysis capabilities.** With DB2 Performance Monitor for AIX, users can customize measurements with spreadsheet-like formulas. For example, rather than monitoring an absolute measurement, a user can monitor a ratio calculated from two related measurements. A variety of formulas are supported, including functions such as average, minimum, and maximum.

  DB2 Performance Monitor for AIX has a starter set of commonly used measurements so users can perform useful monitoring immediately. Data can be analyzed to filter out only those records applicable to certain database objects or to a certain period. For example, users tracing a performance problem can quickly focus on event records applicable to a given database.

  The data produced by the DB2 Performance Monitor can be stored in a DB2 database, providing an opportunity for additional analysis. For example, event records showing I/O cost for every statement in a given application can be sorted in descending order, quickly identifying the most expensive statements.

- **Robust alerting capabilities.** For any performance measurement, users can define exception conditions by specifying a threshold value. When that value is reached, the user can specify any or all of the following actions:

  -Notification through a window or audible alarm
  -Logging of a record in a log file
  -Execution of a command or program
  -Notification to a management product such as NetView®

With a comprehensive, flexible, yet easy-to-use array of monitoring options, administrators and developers have a powerful tool to aid in performance tuning and problem determination.

Figure 3 shows, in graph form, the level of database activity over the last five minutes using selected indicators of database performance.

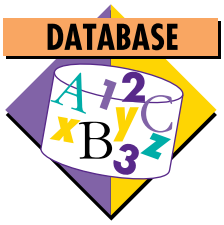
Summary

DB2/6000 has the following benefits in both uniprocessor and SMP environments:

- **Solid architecture**. DB2 can support many users in client-server OLTP and decision-support environments without compromising the efficient use of system resources.
- **Excellent performance**. DB2 for AIX can handle sustained rapid response time under a heavy workload from hundreds of users accessing a large database.
- **Good scalability**. As the number of processors increase, DB2 for AIX takes advantage of them, resulting in greater throughput.

Gene Kligerman, IBM Canada, 1150 Eglinton Avenue East, Toronto, Ontario, Canada, M3C 1H7. Internet: genie@vnet.ibm.com. Mr. Kligerman is currently a planner for the DB2 for AIX Version 2.1 server. He has a BSc and an MSc in Computer Science from the University of Toronto.

Ming Wu, IBM Canada, 1150 Eglinton Avenue East, Toronto, Ontario, Canada, M3C 1H7. Internet: mingwu@torolab2.vnet.ibm.com. Ms. Wu is a member of the workstation DB2 performance group. She has an MASc in Electrical Engineering and a BASc in Engineering Science from the University of Toronto.



INFORMIX On-Line Dynamic Server

By David Clay

The INFORMIX On-Line Dynamic Server 7.1 on AIX 4.1 was released in December 1994. This server represents the culmination of Informix's efforts to build a new server architecture aimed at maximizing DBMS performance and utility on Symmetric Multiprocessor (SMP) machines. On-Line Dynamic Server 7.1 features Dynamic Server Architecture with Parallel Data Query (PDQ). Version 7.1 functionality includes parallel scans, joins, aggregates, index builds, backup, restore, and recovery. It also includes a mechanism for horizontal table partitioning and global resource control for decision-support queries.

The Informix SMP capability is not just an extension to an old architecture: it is a fundamental redesign of the server. A look at the evolution of Informix servers, beginning with Version 5.0, provides a better understanding of the advantages of the INFORMIX On-Line Dynamic Server 7.1 architecture.

On-Line 5.0 Server Architecture

The On-Line 5.0 engine is SMP-enabled to benefit many Online Transaction Processing (OLTP) applications. Version 5.0 architecture has one operating system process for each connected client, shown in Figure 1. The SMP operating system can schedule several of these server processes simultaneously, thereby achieving some level of parallelism. The engine processes coordinate their access to shared resources, such as the buffer cache and logging services, using a high-performance mutual-exclusion mechanism. This works well, as shown by the large number of published TPC benchmark results using INFORMIX On-Line 5.0. For example, in October 1993, INFORMIX On-Line 5.0 achieved a TPC-C rate of 726.13 tpm on a RISC System/6000 (RS/6000™) Model 590 running AIX.

Of course, some drawbacks to this architecture exist. One is that the number of operating system processes increases as the number of clients increases. This results in an increase in process context switches. A second drawback is that a single-user session cannot use more than one processor to work on a large query.

Dynamic Server Multithreaded Architecture

Version 6.0 of On-Line Dynamic Server addresses the first drawback. Dynamic Server Architecture (DSA) is a multithreaded architecture in which many threads are multiplexed over a few server processes. The number of server processes is typically smaller than the number of physical processors on the SMP machine, as shown in Figure 2. Each user session gets a thread that can be scheduled on one of the Informix server processes rather than getting its own dedicated process.

A thread has a much smaller context than a process, and thread switches can be done without entering the operating system kernel. Thus, a thread-context switch time is a small fraction of a process-context switch time. This is advantageous because Database Management Systems (DBMSs) spend much of their time context switching.

This time savings represents only part of the advantage of multithreading. More significantly, the server can schedule threads according to the best use of the server. For example, suppose 1,000 clients are each supported by one thread in the server. If one thread obtains an exclusive lock on a critical centralized resource such as the log buffer, a DSA server process can guarantee that the thread will not be context-switched while holding the critical resource.

This guarantee cannot be made in a DBMS which implements this using 1,000 server processes. From time to time, the operating

David Clay

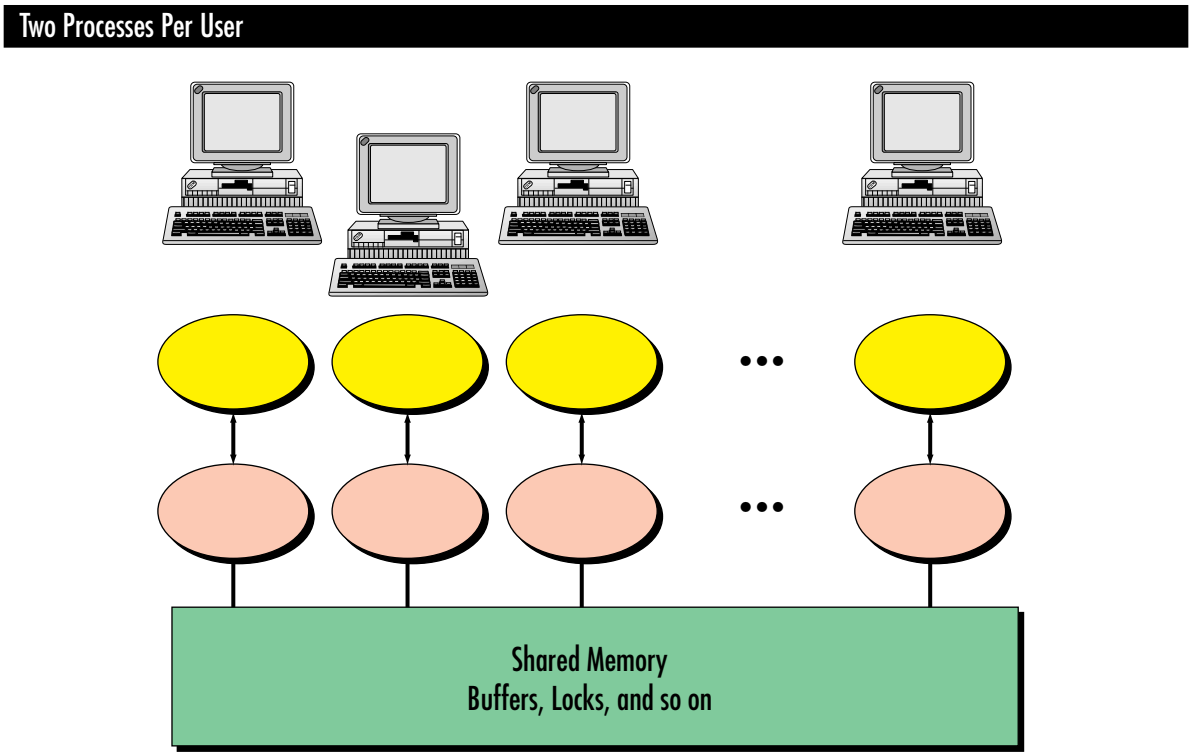


Figure 1. Two processes per user

system will time-switch a process holding a critical lock. The processes allowed to run in its place will simply note that they cannot obtain the lock and give up the processor to the next process, which does the same, and so on. As the number of user sessions increases, the problem becomes worse. Some, but not all, of these problems can be alleviated using a Transaction Processing (TP) monitor in a process-based system. This requires a considerable investment in TP software, additional application development time and complexity, and additional administrative costs.

Figure 3 illustrates the performance of On-Line 5.0 (a process-based server) versus On-Line Dynamic Server 6.0 (a thread-based server) as the number of user sessions increases. With On-Line Dynamic Server 6.0, the throughput remains stable as the number of clients increases.

Informix's Dynamic Architecture has been shown to be scalable for OLTP workloads; that is, physical processors can be added to handle larger workloads so that the processing time remains the same. This helps to fulfill one of the promises of SMP: when a system runs out of horsepower, capacity can easily be expanded by plugging in more processors. SMP scalability is difficult to achieve in a DBMS because of many internal concurrency problems that arise, particularly as the number of physical processors becomes large.

In addition to a radically new architecture, On-Line Dynamic Server 6.0 also included some new performance features: parallel backup, parallel recovery, parallel index build, configurable read ahead, and dictionary cache.

The query engine was completely rewritten for Version 7.1.

PDQ Architecture

The multithreading subsystem that made its debut in Version 6.0 is the ideal platform on which to build a parallel query capability. Thread creation, destruction, and synchronization is much more efficient than the corresponding functions for operating system processes, making it practical to quickly map several threads to a query. However, many other structural changes to queries are required before the thread mechanism can be used effectively on a single query. The iterator model and the exchange iterator are two such structural changes.

The Iterator Model

The query engine was completely rewritten for Version 7.1 to conform to an iterator model. *Iterators* are self-contained software objects that

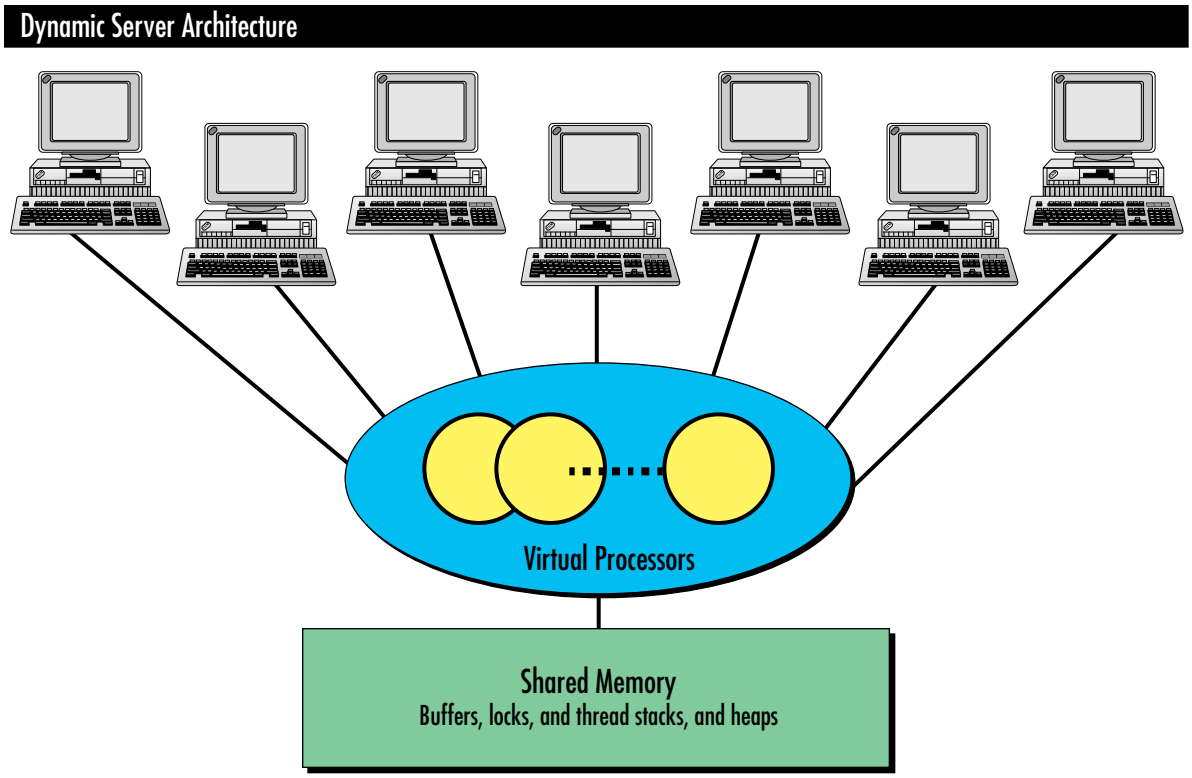


Figure 2. Multiprocess and multithreaded architecture

accept a stream of tuples from one or two anonymous sources, usually another iterator or iterator pair, and produce a transformed stream of tuples. Specific iterators are scan, nested loop join, sort merge join, hash join, union, sort, and aggregate. The iterator model allows complex queries to be structured by stringing together the appropriate iterators into a tree structure as shown in Figure 4. It also facilitates parallelism because iterators can be easily duplicated to make a new instance of the iterator. The OnPerf Graphical User Interface (GUI) monitoring utility that ships with the server actually draws the iterator tree and allows the user to view the flow of tuples through the tree as the query executes.

New iterator algorithms were also introduced in Version 7.1. Aggregation (count, sum, avg, min, max) is done using a unique hashing algorithm that is much faster than the previous algorithm. Hash joins have been slow to appear in commercial products. Hash joins are frequently much faster than sort merge joins, because they only require that a hash table be built on the smaller of the two inputs. The larger input does not need to be organized in a particular way. Sort merge joins require that *both* inputs be sorted, which can be very costly.

The scan iterator, used to read table data from the disk, has many improvements over earlier systems such as "light scan" functionality, which often allows full-table scans to bypass the buffer cache. Predicates are "pushed down" and evaluated directly in the disk buffer. Using these methods, tables can be sequentially scanned at maximum hardware disk rates, typically in the 2 MB to 5 MB per-second range. This is unusually fast for a Relational DBMS (RDBMS).

The Exchange Iterator

The mechanism for launching and managing parallel threads is the *exchange iterator*, first described as part of Goetz Graefe's experimental Volcano DBMS. A special exchange iterator can initiate new threads and copy iterator instances into the new threads. It can also perform pipe fitting between iterator instances and manage the data flow using buffering and back pressure to equalize the speed between producer and consumer threads. This pipe fitting includes a repartitioning mechanism described below and shown in Figure 5.

Data Partitioning

To parallelize queries, tables and intermediate results must be divided into partitions. The DBA partitions permanent tables using extensions to the `create table` statement. Internal temporary tables are partitioned by the DBMS. The Informix terminology for horizontal partitioning of tables across many disk devices is *fragmentation.* Tables can be fragmented by round robin, a range of key values, or an arbitrary series of expressions. Indexes can be fragmented in the same way as tables. The SQL syntax for creating tables and indexes and altering tables is extended to allow for establishing and changing fragmentation schemes. Figure 6 shows examples of the syntax.

Table or index scans are parallelized by starting a scan thread for each fragment. Another advantage to table and index fragmentation includes logical partitioning by range or expression. This allows the query engine to eliminate certain fragments based on the `where` clause of a query. For example, if a table is partitioned by entry date and a query searches for a range of entry dates, not all of the table's disks will need to be scanned.

A fragment is an ideal unit of backup—another advantage of fragmentation. When a disk crashes, only the affected fragment must be restored, not the entire table. The unaffected fragments remain structurally intact, and queries may proceed against the table even though part of the table is unavailable. SQL syntax is provided to skip unavailable fragments on read-only queries.

Fragmentation can be used to accommodate very large tables. It is not practical to store a single table of several hundred gigabytes as a single structural entity, because it sometimes forces the DBMS to read, update, or restore the table as a unit.

Partitioning of Intermediate Results

Depending on the requirements of an individual iterator, tuples may need to be repartitioned on-the-fly. The exchange iterator does this. For example, when a hash join is parallelized, there will be several instances of the hash join iterator, each with two parallelized inputs. In the simplest case, these inputs will be table scans. The results of the parallel scans should be divided among the join threads so that each unique join key value always goes to the same join thread. This is done by a highly efficient hashing algorithm that is built into the exchange iterator. Exchange takes tuples from the scan thread, then hashes on the join key to

OLTP Scalability

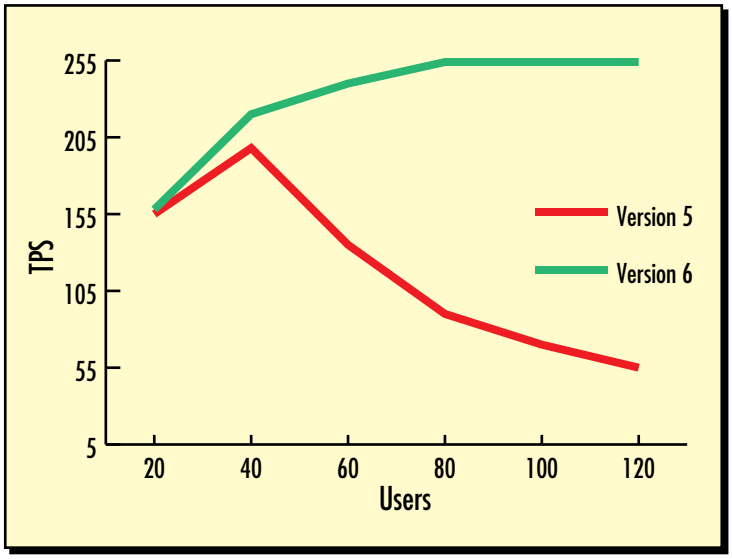


Figure 3. OLTP scalability

Sample Iterator Tree

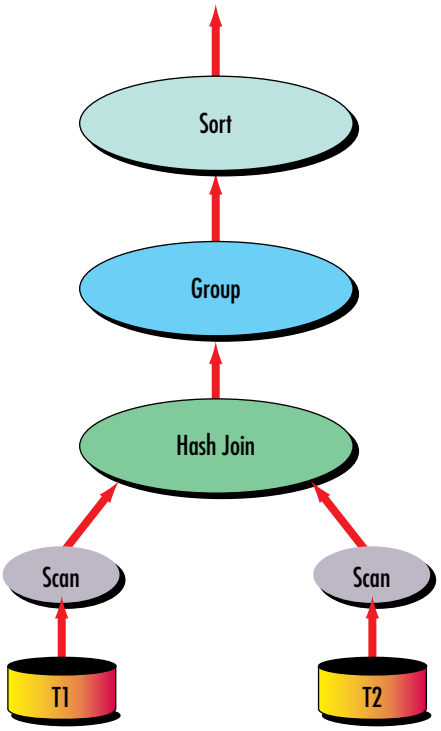


Figure 4. Sample iterator tree

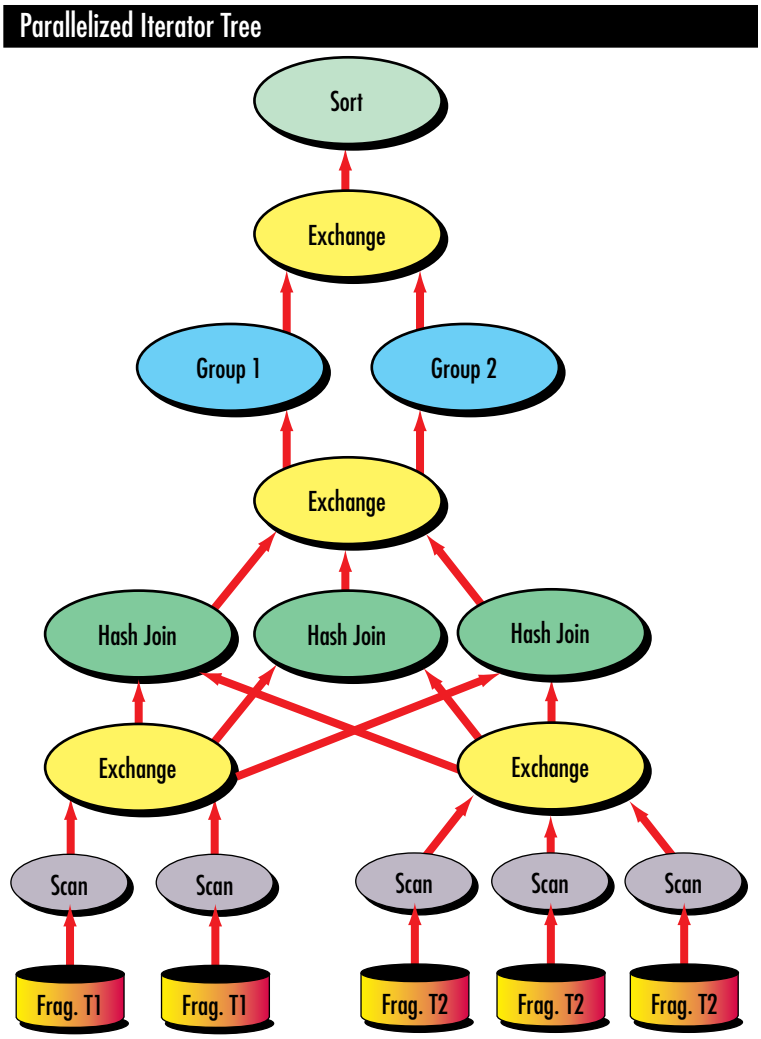


Figure 5. Parallelized iterator tree showing exchange pipelines

```
Create table account fragment by expression
      (column specifications. . .>
       fragment by expression
              account_num <100,000 in account_space1,
              account_num <200,000 in account_space2,
              remainder in account_space_remainder;

Create index account_index on account (account_num)
       fragment by expression
              account_num <50,000 in index_space1,
              account_num <200,000 in index_space2,
              remainder in index_space_remainder;
}
```

Figure 6. Example of syntax for establishing fragmentation

determine which join instance will receive the tuple. Aggregation and sort merge join iterators also require hash partitioning.

Partitioning in this way can cause data skew, which occurs when values of a join or group by key are not uniformly distributed. For example, to count inhabitants of the United States by state, the thread handling New York would be much busier than the thread handling Alaska. The PDQ engine has several unique strategies for dealing with this situation and can adjust to it dynamically.

Resource Management

With PDQ, queries can be assigned a query priority ranging from 0 to 100 to indicate the percent of DBMS resources (such as CPU, memory, and scan bandwidth) that should be allocated to the query. CPU resources are controlled by limiting the number of threads assigned to a query. The number of threads varies with the query priority and the number of server processes, which represents the maximum degree of parallelism.

Performing queries that contain sorts, joins, and aggregates is highly dependent on memory availability. Since most commercial DBMSs (including On-Line 5.0 and On-Line Dynamic Server 6.0) have no way to allot memory among queries, they take the conservative approach of never using too much. Although this allows more queries to make adequate progress without impacting each other, it does not facilitate running large queries that could benefit from large memory allocations. On-Line Dynamic Server 7.1 has mechanisms for adjusting the size of the memory pool used for large queries, the maximum number of concurrent large queries, and the relative priority of the queries, so that the memory can be apportioned among them. Since these mechanisms are dynamic, the availability of resources to large decision-support queries can be dialed down during periods when the OLTP load is high.

Performance

The ultimate measure of a parallel DBMS implementation is its performance. Figures 7 and 8 show speed-up curves for two simple queries. Speed-up is measured by using more CPUs without varying the amount of data. A perfect speed-up curve is a hyperbola (not a straight line) showing that execution time is reduced to $1/n$ x t when resources are increased n times.

Scan Performance

Figure 7 shows the speed-up curve for a parallel select varying the number of disks. The difference between the top On-Line 5.0 line and the lower On-Line Dynamic Server 7.1 line is due to the algorithmic improvements described above. The On-Line 5.0 line does not show speed up because 5.0 was not parallelized for scans. The "Best Possible" line was obtained by running a small UNIX program that performed raw reads in a tight loop. The disks in this experiment could deliver about 2 MB per second, so the scan speeds shown were limited by the disk hardware, not the DBMS software.

Hash Join Performance

Figure 8 shows the speed-up curve for a parallel hash join varying the number of disks and CPUs. Again, speed-up is excellent, and the performance of On-Line Dynamic Server 7.1 surpasses On-Line 5.0. The rightmost point indicates that a speed-up of approximately seven is obtained with eight times the resources.

Future Products

A follow-on product scheduled for release in early 1996 will provide parallel load and unload utilities. These utilities will be capable of I/O to multiple external devices, and come with a GUI interface and built-in conversion routines for handling data from many sources. Numerous server performance enhancements will permit the loading and unloading of data from the database at disk speeds.

Informix's Dynamic Server Architecture is currently being extended to encompass loosely coupled systems such as IBM's SP2™. Many applications will never need a platform this large, but for those that do, DSA provides a growth path from the SMP platform. The new server product, On-Line Dynamic Server 8.0, is planned to be available in mid-1995.

David Clay, Informix Software, Inc., 921 S.W. Washington Avenue, Portland, OR 97205. Internet: davec@informix.com. Mr. Clay, an engineering manager in the Servers and Connectivity Division, was the project leader of the Informix parallel query project. He is currently extending the Informix engine to massively parallel, shared-nothing architectures such as IBM's SP2. Mr. Clay has a BA in Mathematics from Michigan State University and an MS in Computer Science from Cleveland State University in Ohio.

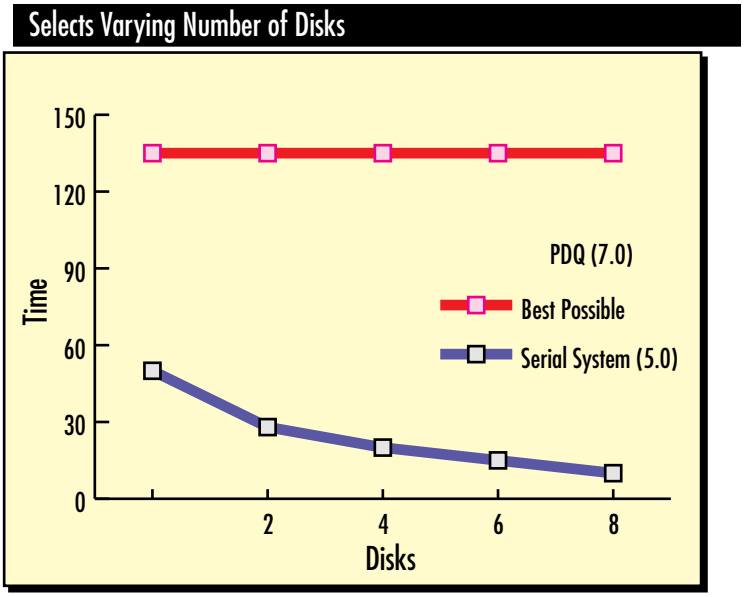


Figure 7. Selects varying number of disks

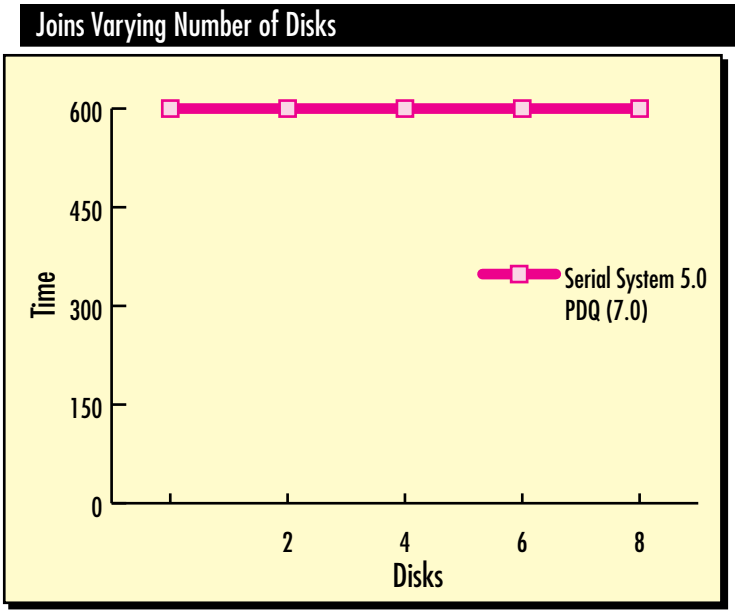


Figure 8. Joins varying number of disks

The Oracle Data Warehouse

By Sandra Lee

This article describes how parallel processing software such as Oracle7, in combination with multiprocessing hardware such as the IBM RISC System/6000 Symmetric Multiprocessor (SMP), provides a Decision-Support Systems (DSS) solution that will help companies predict future trends and growth opportunities that will give them a competitive edge in the marketplace.

In an ever-changing marketplace, collected data about a corporation becomes a very valuable asset. The effect of decisions based on this data can have a significant effect on the organization and can be a key to gaining and keeping a competitive edge in the marketplace. In recent years, a market for Decision Support Systems (DSS) has emerged to help companies use their stored data.

Most companies have an Information Systems (IS) organization made up of Operational Systems and DSS that deals with corporate data. An *Operational System* performs Online Transaction Processing (OLTP) functions such as order entry, work scheduling, and financial systems. OLTP gathers and stores organizational data from the day-to-day operations part of running the business. *DSS applications*, on the other hand, are responsible for providing consolidated data for making decisions about the day-to-day and long-term workings of an organization. Although such decisions influence the business, they are not directly involved with the actual running of the business.

In the past, many DSS analysts inefficiently gathered data. For example, a marketing director from a retail clothing company who wanted to do a targeted marketing campaign would ask the DSS analyst for the results of a complex query: "Who are all the people who spent more than $300 using credit cards after the interest rate increase in the four geographical regions of the United States?"

The DSS analyst would then turn to four regional Operational IS groups (or machines) to get a snapshot of the requested data. The snapshots obtained from the Operational IS groups would invariably be taken at different points in time, and sometimes from different data models. For example, the Western Region might track customers who used credit but not dates of purchases, while the Northern Region may track customer purchase dates but not method of payment. Additional research would be necessary to provide requested information not stored in that region's database. In short, this process of getting results from multiple sources with redundant and overlapping extracts would be very time-consuming with a high rate of errors.

Extracting inconsistent data from multiple databases is only one problem faced by DSS. The intensive CPU usage required by DSS to do complex read-only transactions of massive amounts of data is another challenge. Furthermore, OLTP and DSS use logically separate data. To address these issues, some companies began providing expensive proprietary query database computers, pioneering the technique of separating OLTP from DSS. Data from the OLTP computers was separated and consolidated on a query database computer. All DSS transactions would be run against the DSS consolidated query database. The objective of separating OLTP and DSS onto separate computers was to speed transaction processing as well as ease the painful searching for data from several different sources.

Nevertheless, OLTP and DSS remain intimately linked. Consolidating data from company-wide OLTP systems, and perhaps from external sources, constitutes the core of DSS. The term *data warehousing* describes the collection, transformation, and distribution of data useful to

decision-makers. With the right tools, company decision-makers can exploit massive amounts of diverse data to respond quickly and flexibly to rapidly changing customer needs and competitive environments.

In the retail clothing company example above, it would have been more efficient for the DSS analyst to run a DSS application against a consolidated data warehouse that contained information from the four geographical regions plus credit card information loaded from external sources. The resulting data would have been integrated and organized into a consistent form, providing a much faster query result with a high degree of accuracy.

Consolidating data from company-wide OLTP systems, and perhaps from external sources, constitutes the core of DSS.

Parallel Processing in the Data Warehouse

Hardware and software for a data warehouse must support several, often conflicting, requirements.

Hardware requirements for a data warehouse vary depending on the size of the warehouse that needs to be created. Considerations include the volume of data to be kept online in primary storage, the size of the storage subsystem supported by the hardware architecture, and the memory and processor requirements for running batch jobs or long-running complex queries. Access to external storage media or machines may also be important in considering the time needed to load data into the warehouse.

The data warehouse software must support both complex preplanned queries and small ad hoc requests. Easy information addition and updating, and fast query performance are essential.

Both hardware and software technology developed in recent years are beginning to address these DSS requirements.

Server Architectures

Until recently, data warehouse applications were limited to only those projects that could justify the high cost of implementing a data warehouse. Existing server technology generally did not provide cost-effective access to enterprise data because of architectural limitations.

Lack of client-server functionality for users to perform ad hoc data access and analysis has limited the usefulness of mainframes for data warehousing. In addition, mainframe performance has not kept up with the processing power needed to mine enterprise data.

Specialized approaches, such as proprietary query-processing systems, have provided improved query performance, but at a price/performance ratio that is often expensive. For example, the cost per gigabyte of disk storage on a proprietary query processor can be up to four times that of an open system. Lack of openness also deterred many companies from implementing these systems.

Open systems have provided dramatic improvements in price/performance for OLTP applications and offer the best support for client-server data access. However, with uniprocessors, response time for queries has been limited by the speed of a single CPU, and the performance of complex queries in an online or batch environment cannot meet application requirements.

With parallel processing machines, the limitations of uniprocessors have been significantly overcome for data warehousing applications. By providing multiple processors, performance and response times for complex queries, data loads, and index creation against large data sets are reduced. The price/performance of open systems has improved by 400% during the last ten years alone. Open multiprocessor systems are the only server platforms that can keep pace with users' processing demands and provide adequate interoperability, data access tools, and data distribution.

IBM's RISC System/6000 Symmetric Multiprocessors (RS/6000 SMPs) fill a large market need in DSS applications. RS/6000 SMPs, initially offered with four CPUs and scaling up to eight, provide multiple processors at a reasonable price and respectable performance on high-end machines. When IBM incorporates the PowerPC™ 604 and the 620 chipsets into the RS/6000 family, performance should increase even more. If the data warehousing capacity provided by these RS/6000 SMPs becomes insufficient, IBM will also provide HACMP/6000™ SMPs—loosely clustered SMPs similar to the HACMP/6000 uniprocessors—and massively parallel SP/2s.

Parallel Software

Although open systems, particularly multiprocessors, have made excellent gains in price/performance, the primary architectural limitation of the DBMS software—until recently—was the inability to take advantage of multiple processors in performing typical DSS operations. Another weakness of traditional Database Management Systems (DBMSs) was an inability to use multiple memories not shared by CPUs.

In recent years, DBMSs have made great strides in parallel processing. Parallel processing allows a task to be broken into many smaller

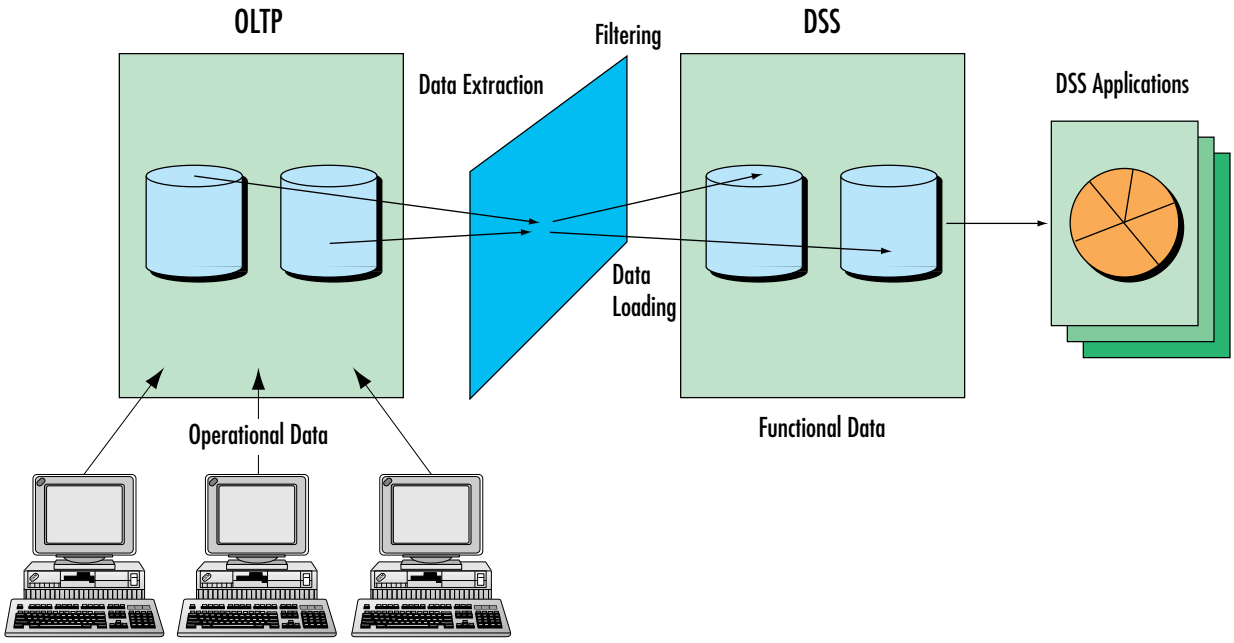


Figure 1. The data warehousing process

tasks that are performed simultaneously. This is unlike traditional serial processing in which a single CPU executes instructions from a single shared memory, one task at a time. Because tasks are smaller and shared among several CPUs in parallel processing, a particular task is finished much more quickly, improving response time and increasing the amount of processing that can be done in a given amount of time.

Although parallel processing has been used for CPU-intensive scientific and analytic applications, it has not been widely used in commercial applications because of the difficulty in parallelizing applications. The difficulty lies in structuring tasks so that certain steps can be executed in parallel, while ensuring that the sequencing is preserved for steps that must be executed serially.

Oracle7

Traditional DBMSs, such as Oracle7, now provide parallel processing capabilities that are appropriate in a commercial DSS environment with SMP and Massively Parallel Processing (MPP) architectures.

The Oracle7 database has a successful history on SMP systems. Oracle7's Parallel Query Option (PQO), Parallel Server, and interoperability enable Oracle7 to support application integration, distributed operations, and mixed application workloads—all part of DSS.

Since it offers excellent price/performance on open systems hardware and is architected to serve a wide variety of enterprise computing needs, Oracle7 PQO is an attractive alternative to specialized DBMSs for data warehousing. Oracle7's PQO processes database requests in parallel, reducing response time for complex queries and for queries against large databases with data warehouses.

The Oracle® Parallel Server enables the Oracle7 DBMS to use multiple nodes on a uniprocessor such as the RS/6000, as well as clusters of multiprocessors such as IBM's upcoming HACMP SMP cluster. Cluster support provides high availability, thereby allowing access to data as needed with reduced database downtime. IBM's HACMP clustering technology permits nodes to be taken down and serviced without affecting the entire system.

Oracle7 runs on many platforms, including the full range of IBM AIX offerings. With Oracle's Gateway technology, Oracle can also access legacy data from other databases and flat files on a variety of different platforms while supporting OLTP and DSS on hardware best suited for a company's particular needs.

Creating the Oracle Data Warehouse

There are several steps in creating a data warehouse, which will be more than just a collection

of OLTP data. Careful design, maintenance, and good data retrieval tools are essential for a data warehouse to be a valuable component of DSS.

Figure 1 provides an overview of the processing of operational data into functional data for decision support. Day-to-day operational data entered into a traditional OLTP database is extracted, filtered, and then loaded into a DSS system. From there, the data is extracted by applications for decision-support use.

Designing the Warehouse

Although good DSS software will improve the performance and usefulness of a data warehouse, careful database design is crucial to its effectiveness. A good data warehouse must support both preplanned decision-support applications and ad hoc queries relating items that may have little in common, such as linking payment methods with a particular zip code.

In one test on a 16-processor system, scaling of more than 95% was measured for the Oracle7 Direct Path Loader.

The data warehouse model must also address the conflicting goals of system flexibility and data delivery performance. Flexibility is required to be able to add tables and attributes of primary data and summaries. Quick query response is also required since it determines the number of analyses that can be performed and the amount of data that can be fed to a presentation package for dynamic data representations.

Although these issues are common to all relational databases, the primarily read-only nature of a data warehouse permits some specialized optimization techniques, such as generating summarized data tables for frequent query paths. Information in data warehouses can be summarized within the database as well. Such summarized data can be used for simple, ad hoc queries while the more detailed information can be used in complex, prewritten queries.

An IS organization designing a data warehouse must decide what type of data should go into the warehouse based on the type of information most often used, the type of queries that will be run, what information should be summarized, and so on.

Loading and Maintaining the Warehouse

Data that will eventually be stored in the data warehouse must first be extracted from the OLTP database. The data is then filtered to remove unwanted information and possibly combined with data from other databases or files. At times, additional data such as a date or location may be added, or the data may be converted to allow comparisons with similar data, such as converting all revenues to U.S. dollars.

The example retail clothing company's data warehouse would contain information from the four IS regions that has been massaged into the same form and contains the same information. For example, the data warehouse would include purchase date and method of payment for all customers in all geographic regions. Additional credit information from external sources, such as a credit card company, may also be added to the data warehouse.

Once OLTP data is extracted and filtered, it must be loaded into the DSS database. The time required to load data into a data warehouse can be a critical factor in the success of a data warehouse system, especially for maintenance. Maintenance can require periodic incremental updates of the data in the warehouse: adding or changing data that has been changed since the last update of the warehouse. Periodically, a complete refresh of the data in the warehouse may also be needed.

Oracle7's parallel data-loading capability speeds the creation and maintenance of the data warehouse, particularly when adding or updating information. Oracle7's PQO provides a parallel load function that uses the already efficient direct path of SQL*Loader. Oracle7 Direct Path Loader bypasses SQL processing to load data directly into database tables. Rows of a single table can be loaded simultaneously by different processors. Multiple sessions running SQL*Loader use the direct path capability to load data simultaneously into the same table. Each SQL*Loader session acts independently, producing near-linear scaling of performance and reducing the time required for data loading. In one test on a 16-processor system, scaling of more than 95% was measured for the Oracle7 Direct Path Loader. While this result is spectacular, it must be remembered that highly parallel systems are highly tunable. The data warehouse must be carefully designed and tuned for best results.

Parallel indexes, which can be created either during the data loading process or just after loading, are often needed in a large database such as a data warehouse. Indexing speeds creation and reconfiguration of a data warehouse. Oracle7's PQO includes a parallel index feature. It automatically executes a `CREATE INDEX` command in parallel across multiple CPUs by parallelizing the table scan and sort associated with index creation. Oracle7's parallel index function provides good scaling, but it is important to remember that

index creation performance is heavily dependent on the amount of sort area available for each of the index build processes.

Data in the data warehouse constantly changes. New information must be added, old data updated, and the data warehouse itself may have to be rearranged for better performance on changed query needs. To do this, *data replication*—the maintenance of several consistent copies of data across different databases—is necessary. The data warehouse must be synchronized with data in the OLTP database, and summarized data within the data warehouse must be kept consistent.

Oracle7's data replication capability helps data warehouse maintenance. Oracle7 can transparently replicate data to and from all databases used to store data, including OLTP databases, DSS databases, and combined OLTP/DSS databases. Using the Oracle Gateway product, Oracle7 can access data stored in non-Oracle databases or flat files.

Using the Warehouse

Retrieval and presentation of contents by DSS applications will determine the worth of the data warehouse. The design of the data warehouse is key to efficiently using the indexes, summary data, and other tuning techniques that ensure rapid retrieval of desired data. The data is more valuable if both anticipated and unanticipated data requests can be rapidly completed. As non-operational data is added to the data warehouse, there will be many unanticipated queries.

Oracle7's PQO is essential to obtaining full value from the data warehouse. It speeds queries by splitting a complex query into smaller parts that can be processed in serial, as shown in Figure 2. Oracle7 also ensures contention-free queries whether or not the queries are parallel, so users attempting to write data do not block users attempting to read data, and vice versa. For more information about Oracle7's PQO, see the May 1994 *AIXpert* ("Oracle Parallel Technology Empowers AIX Systems," page 37).

Data retrieval also presents query results from the data warehouse. To create customized DSS applications for retrieving data, Oracle provides CASE and the Cooperative Development Environment (CDE) tools. Tools such as Oracle Forms and Oracle Reports enable application developers to create interfaces to retrieve the results of prewritten complex queries into a particular format. Oracle also provides end-user query tools, such as Oracle Data Browser or Oracle Data Query, for ad hoc queries without the need for users to know underlying database structures. Oracle supports hundreds of third-party tools, giving companies a wide range of tools choices to fit their requirements.

Today, computer technology is fueling the emergence of data warehousing as an essential component of successful businesses. Parallel processing software such as Oracle7 in combination with multiprocessing hardware like the RS/6000 SMP, provide a DSS solution that will help companies predict future trends and growth opportunities, and give them a competitive edge in the marketplace.

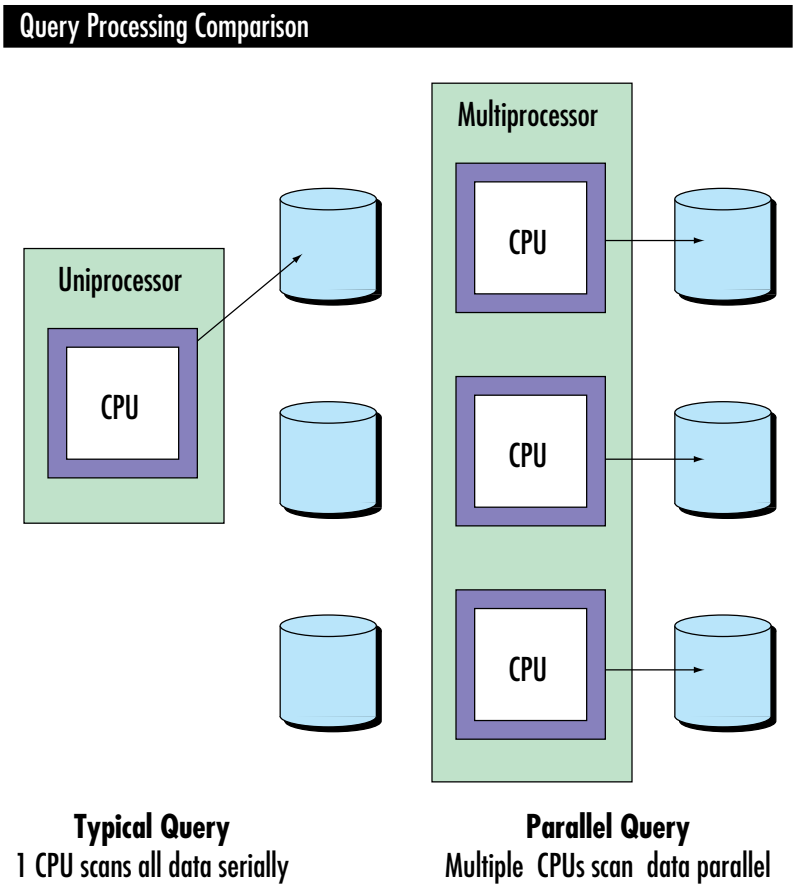
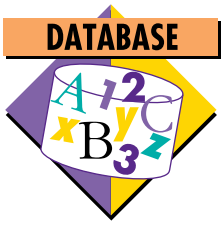


Figure 2. Query processing on uniprocessor versus multiprocessor

Sandra Lee, Oracle Corporation, 500 Oracle Parkway, Box 659406, Redwood Shores, CA 94065. Internet: shlee@us.oracle.com. Ms. Lee is the product line marketing manager for the AIX platform at Oracle. She has BA degrees in Computer Science and English from the University of California at Berkeley and a graduate degree from Boston University.



SYBASE for IBM RS/6000 SMP Servers

By Josh Bersin

This article discusses Sybase's SQL Server 10 architecture for the IBM RISC System/6000 PowerPC-based SMP servers. It describes the scalable high-performance SYBASE architecture for AIX and how it can be used in the development of large-scale, mission-critical client-server applications.

SYBASE SQL Server 10 is the latest generation of Sybase's client-server Relational Database Management System (RDBMS) servers. SQL Server has been available since 1987, and through a joint development and marketing relationship with Microsoft®, it has been widely adopted in the industry on over 150,000 RISC and PC server systems. Symmetric Multiprocessor (SMP) support became available with Release 4.8, which shipped in 1990.

Sybase's view of the client-server market is still unique in the industry. The key to high performance client-server applications is the ability to deliver multithreaded high performance in the server with advanced capabilities for transaction integrity. Stored procedures, triggers, and database Remote Procedure Calls (RPCs) have made this possible.

Stored procedures and triggers are server-based applications that encompass shared business logic similar to CICS™ transactions. They are compiled and resident in shared memory for maximum performance. They implement SQL and other functions, and protect access to the data from across the network.

In the SYBASE architecture, all database users are viewed as clients. Clients can execute SQL or RPCs to access data directly or to invoke stored procedures. Clients can run on PCs, UNIX workstations, Macintoshes®, or even IBM mainframes.

This powerful architecture, coupled with Sybase's unique multithreaded RDBMS server, has created an entire development environment for new applications. The RDBMS environment offers much more than just relational access—it offers scheduling, multithreading, memory management, online backup, built-in networking, and a variety of security functions. On a system such as the RISC System/6000 (RS/6000) PowerPC SMP, SQL Server provides a total client-server programming and deployment environment.

Josh Bersin

SMP Hardware Architectures

SMP systems implement multiple processors sharing a single memory. To deliver a single systems image, they have *cache coherence* features incorporated—ensuring that one processor does not write over the memory being used by another. The key challenge in the design of these systems is scalability. The hardware and operating system must be designed to minimize contention and locking as additional processors are added. Most UNIX SMP systems today scale well up to four processors, but then tail off significantly after that, as shown in Figure 1.

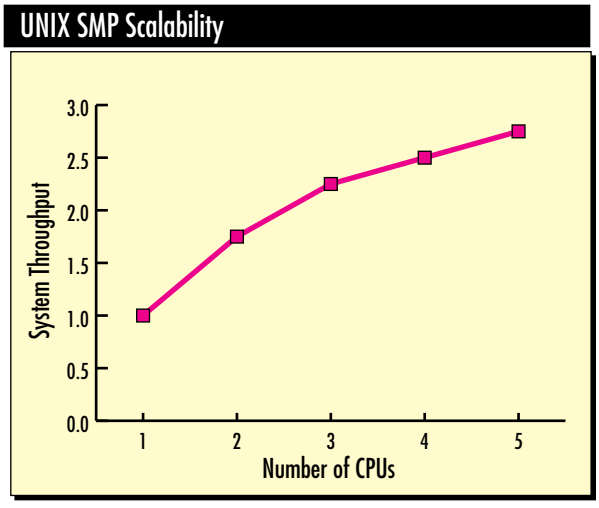


Figure 1. Typical UNIX SMP scalability—operating system throughput

Most measurements of scalability measure operating system-level tasks. The key features in client-server applications are scalability of transactions-per-second and number of users, while maintaining response time at a subsecond level. Sybase, realizing the demands of SMP architectures, specifically designed an implementation of SQL Server for optimum scalability in the Online Transaction Processing (OLTP) environment.

SYBASE Architecture

The key innovation in SQL Server is the single-process multithreaded architecture (see Figure 2). SQL Server uses a single database process per CPU with an optimized thread model within the process. The SQL Server itself handles network connections, scheduling, and memory management within the database process.

Scalability

This single-process multithreaded architecture results in an RDBMS server that scales to thousands of concurrent users on RS/6000 systems. As workload is added, SQL Server achieves uniform subsecond response time with almost no degradation.

Alternative database architectures implement multiple processes, producing much higher system overhead and greater resource utilization.

The net result is a performance characteristic that provides predictable, scalable throughput, and subsecond response time as user load is added, shown in Figure 3.

High-Performance OLTP on PowerPC SMP

Based on benchmarks being performed at the time this article was written, SYBASE is already achieving very high performance and throughput on the new IBM SMP systems (Model J30 4-Way SMP with TPC-C transactions), as shown in Figure 4.

Support for Many Online Users

SYBASE's architecture, coupled with compiled and memory-resident stored procedures, allows very large numbers of users to be supported with predictable subsecond response time. A key feature making this possible is Sybase's internal threading architecture. The SQL Server's internal threads allow individual users to run within the RDBMS process, using less than 60 KB of memory per user.

The SQL Server can support hundreds to thousands of users in an online environment on the RS/6000 SMP products. Based on expected performance of the PowerPC family, we have esti-

Multithreaded Architecture Versus Traditional RDBMS Design

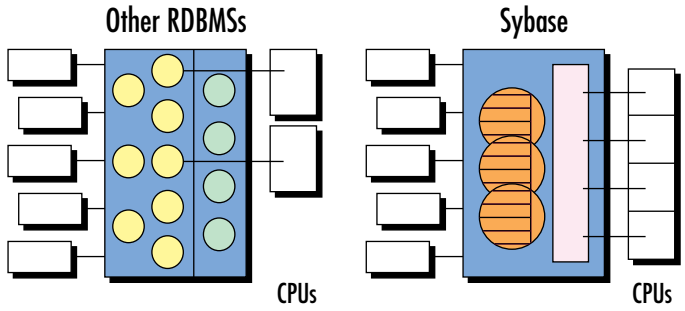


Figure 2. SYBASE multithreaded architecture versus traditional RDBMS design on uniprocessors and SMP

SYBASE Performance Characteristics

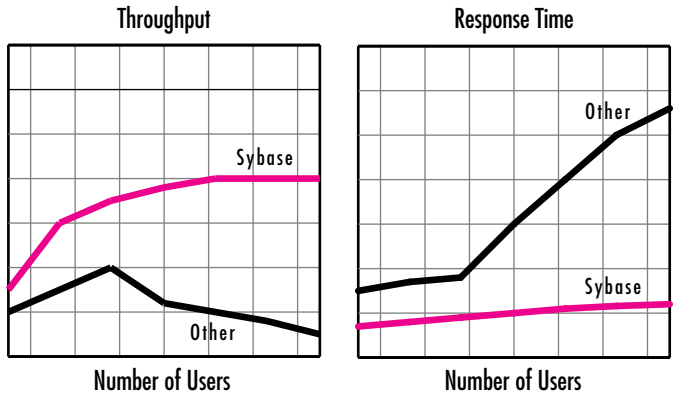


Figure 3. SYBASE performance characteristics on RS/6000 PowerPC SMP

TPC-C Workload

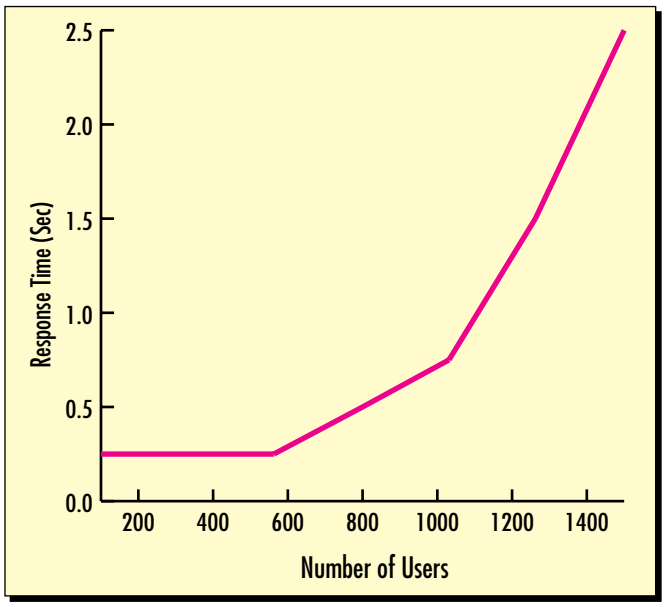


Figure 4. RS/6000 with SYBASE System 10 running TPC-C workload

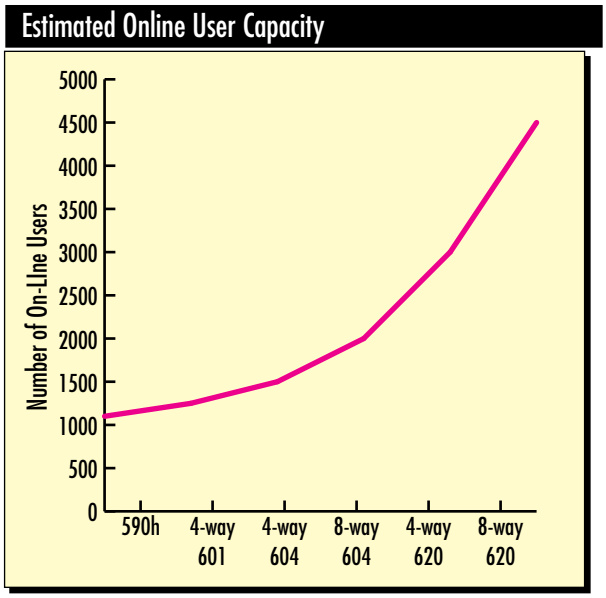


Figure 5. Estimated online user capacity RS/6000 SMP servers running the TPC-C workload (1.5 sec response time)

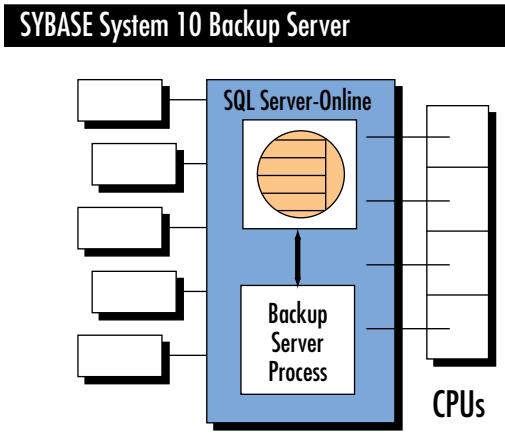


Figure 6. SYBASE System 10 Backup Server architecture

mated the online performance of SQL Server on the PowerPC to scale to thousands of users, as shown in Figure 5.

Support for Very Large Databases (VLDBs)

One major problem facing customers as their applications grow is the support of large databases (tens to hundreds of gigabytes online). The critical issues to support such databases include high-speed online backup, support for mixed workloads (OLTP and queries), and complex queries that access large amounts of data.

Over the last ten years, Sybase has developed features and capabilities to support VLDBs. Most important is the SYBASE Backup Server, which is shipped with every copy of SQL Server. The SYBASE Backup Server is a separate UNIX process that can run on any SMP processor. It provides online parallel database backup to a variety of tape devices. Backup Server runs while the SQL Server is operating, backing up over 90% of the transactions that execute during the back-up process. Backup Server has been benchmarked on SMP systems backing up online databases at over 40 GB per hour. Figure 6 shows the Backup Server architecture.

Support for Complex Queries and Mixed Workloads

As databases grow larger, it becomes increasingly important to support mixed workload: high-volume OLTP and growing loads of queries and batch reports. This environment requires the RDBMS to interact efficiently with the operating system so the SMP capabilities of the system can be efficiently utilized to balance the workload. SQL Server's Virtual Server Architecture (VSA) efficiently accommodates this requirement.

Sybase Interactive Query Accelerator

A different strategy is often needed for complex queries. When a database reaches tens of gigabytes in size, random queries can often monopolize an entire CPU and I/O channel for extended lengths of time. This naturally contends with system needs for online applications.

Sybase recently announced the availability of the SYBASE Interactive Query Accelerator (Expressway Technologies), which provides a fully indexed copy of the SQL Server database for direct query access at very high speeds. The Query Accelerator allows queries to access indexes against the database on any field, regardless of query, to minimize both CPU and disk access for queries. It exploits the large memory and SMP capacity of the PowerPC server to maximize throughput—and provides speed-up from 400 to more than 1,000 times normal for SQL queries against the database.

SQL Server VSA for SMP

In the RS/6000 SMP, the SQL Server VSA enables the customer to load multiple servers (up to the number of processors) that function as a single server. Each process can send threads to any CPU, allowing the system to maximize use of the computing resources. Since the customer can allocate as many database engines as needed, the

system can be partitioned between RDBMS processing and other system use.

This architecture has many advantages:

- Maximizes use of the system by dispatching low-overhead threads to any available CPU.
- Allows customers to tune the system. If only two CPUs are needed for SQL Server, customers only create two engines. As more are needed, the system can be reconfigured to add capacity.
- Allows SQL Server to efficiently manage mixed workloads of OLTP and queries by dispatching tasks to separate CPUs and minimizing contention.

Compared to other RDBMS architectures, only a minimum number of AIX processes are used, allowing excellent performance and scalablity.

Managing SYBASE System 10 on the IBM SMP

In the SYBASE SQL Server VSA architecture (see Figure 7), the system can be tailored for any particular workload. The client systems and administrator see the VSA as a single SQL Server accessing a single database. When the SQL Server is brought up, the administrator determines the number of engines to be loaded. Each engine is an SQL Server process, sharing single thread and run queues.

There should be no more engines than CPUs, and the number of engines should be increased until optimum performance is reached. The engines dispatch client tasks to their appropriate processors using a single queue and shared memory, presenting a single virtual server. As additional engines are added, system performance will improve—increasing demand on I/O and other system resources. As the number of engines is increased, the system should have enough balanced disk to handle the increased workload.

In this symmetric architecture, each engine can run on any CPU. If a CPU is busy, available tasks will be dispatched to any available CPU. If the number of SQL Server engines is less than the total number of CPUs (usually recommended), there will always be available CPUs for additional applications on the system.

SQL Server Task Management in SMP

The SYBASE SQL Server documentation describes the actual task management in the SMP. Here is a simple description of the process:

1. A client application issues a login request. In response, SQL Server creates a user task to handle work from the client.

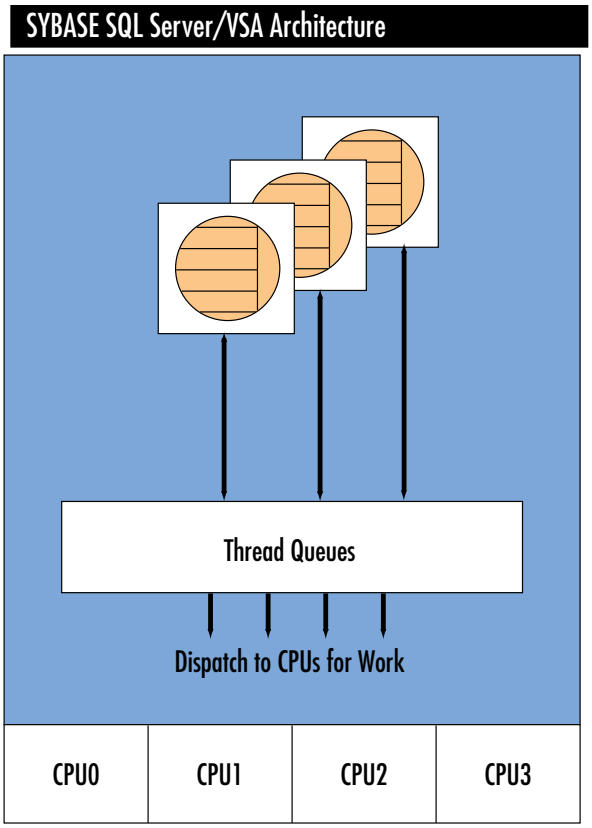


Figure 7. SYBASE SQL Server VSA architecture

2. The client presents SQL Server with work to do—a series of Transact-SQL commands.

3. SQL Server adds the client's user task to the runnable task queue. The server engines compete for the user task at the head of the task queue.

4. The engine executes each step until the task completes or blocks, while waiting for I/O or locking. When the task blocks, it yields the server engine to run other user tasks. Once the block is resolved, the user task is again added to the runnable task queue.

5. After the task blocks for the last time, it continues executing until it finishes. Then the user task yields the server engine and moves to the sleeping task queue until the client presents the server with more work.

Josh Bersin, Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608. Internet: joshua.bersin@sybase.com. Mr. Bersin manages Sybase's product and marketing strategy for the IBM product line. He has a BS from Cornell University, an MS from Stanford University, and an MBA from the University of California at Berkeley.

The NetView Association Invites Vendors to Participate

By Larry Kuntz and Donna Bowie-Conway

In September 1994, IBM announced a major leap forward in network management framework technology with new versions of NetView for AIX and NetView for OS/2. (A new version of NetView for OSF/1® was also announced for Digital Equipment Corporation's POLYCENTER® Manager.)

Multiple Database Support

One key strength of NetView for AIX is its support for multiple relational databases, including DB2, INFORMIX, Ingres®, Oracle, SYBASE, and Rdb. The new NetView for AIX also includes other enhancements:

- Continuous operator access—one NetView console can take over for another during outages
- A rich set of Application Programming Interfaces (APIs)
- An improved end-user interface
- New trap delivery facilities

For vendors that provide network management products and services, these new functions represent exciting opportunities. For example, network management applications can now write to, and read from, the most popular databases in the industry. Continuous operator access offers an opportunity for applications to perform automated operations and around-the-clock network monitoring.

Your network and systems management applications can exploit the richest set of APIs in the industry, including General Topology Manager (GTM), end-user interface, Simple Network Management Protocol (SNMP), event filtering, and the industry-standard X/Open™ Management Protocol (XMP) API.

Serving Third-Party Vendors

The NetView Association has experienced nearly a three-fold increase in the number of third-party applications made available by association members since October 1994. As the number continued to grow, *Network World* magazine noted that NetView now ranks first in the total number of applications—moving ahead of its two main competitors, Hewlett-Packard's® OpenView™ and Sun's SunNet Manager.

Continuing its steady growth, the NetView Association enrolled its 200th member in November 1994. Jointly sponsored by IBM and Digital Equipment Corporation as part of their unique development and marketing partnership, the association was established to enrich the management capabilities of the NetView product line. Its members provide the additional management functions that customers need to ensure their communications and telecommunications equipment and systems are effectively managed.

NetView Association members are third-party vendors who develop applications, vendors who develop SNMP-based communications products, and service providers such as consultants and systems integrators. Member services include the following:

- Technical support via telephone and E-mail
- Technical updates about IBM management products
- Not-for-sale software loans

- Referrals to classes about NetView programming
- Listing your product or service in the NetView Association Catalog, which is distributed to the IBM and Digital sales teams
- A subscription to *ViewPoints* magazine, which contains news about the NetView products, the association, and its members
- Invitations to the NetView Association Annual Meeting
- Loans of RISC System/6000 and Alpha/AXP hardware
- Testing and distribution for Management Information Bases (MIBs)
- A certification program to verify that applications are integrated with NetView
- The opportunity for vendors to demonstrate their certified applications at trade shows

Membership is free of charge, and most of the services listed are provided at no cost.

The NetView Association Annual Meeting, held in the Spring in North Carolina, brings together association members from all over the world to air concerns, make suggestions, learn about IBM's and Digital's plans for NetView, and do business with each other.

Membership in NetView Association is open to all third-party vendors who develop management applications software or tools, or provide services related to network or systems management. IBM and Digital see your membership as a "win-win" proposition: you receive assistance in developing and marketing your product, and IBM and Digital benefit from having the number-one network management platform as measured by the breadth and quality of vendor applications.

To join the NetView Association or for more information, send a note to Larry Kunz at ldkunz@vnet.ibm.com or call the association's hotline at 919-543-2939.

Larry Kuntz and **Donna Bowie-Conway** are development managers with the NetView Association. IBM Corporation, Box 12195, Research Triangle Park, NC 27709.

POWER NOTES

AIX 4.1 Packaging

Filesets
AIX 4.1 has approximately 2,200 packages called filesets that can be separately installed. For example, the `bos.rte.up` fileset contains the uniprocessor kernel and the `bos.rte.archive` fileset contains archive commands such as `tar`, `cpio`, and `backup`. This new packaging allows installation of only what is necessary, generally requiring less disk space than AIX Version 3. Since each fileset can be serviced separately, fixes for AIX 4.1 are smaller and more localized. Fixes are delivered in fileset packages and are cumulative; that is, each new level of a fileset contains all previous changes.

VRMF
In AIX 3.2, Program Temporary Fix (PTF) numbers were used for revision control, making it difficult to determine the level of a system component. In AIX 4.1, Version Release Modification Fix (VRMF) levels are used for revision control. For example, the AIX 4.1.1 `bos.rte.up` fileset had a VRMF of 4.1.1.0, indicating Version 4, Release 1, Modification Level 1, Fix Level 0.

Version number: Incremented to indicate a new product. For example, AIX Version 4 is a different product from AIX Version 3. New versions contain major functional enhancements and typically come two or more years apart.

Release number: Incremented to show a product enhancement; we expect new releases of AIX Version 4 will come approximately one year apart.

Modification level: Incremented for two reasons: accumulation of maintenance and support of new processors or devices. Neither the maintenance nor the new support enhances the behavior of the product on existing systems. When the modification level is adjusted, the fix level is reset to zero. We expect modification levels of AIX Version 4 three to six months apart.

Continued on next page

Fix level: Incremented when a fix is added to a fileset. When the modification level is adjusted, the fix level is reset to zero. Fixes for AIX Version 4 are produced on customer demand.

Maintenance and fix levels for AIX Version 4 do not change application interfaces; applications that are written to documented interfaces should function identically on different maintenance and fix levels.

Costs

Customers are charged when upgrading to a new version or release. There is no charge for maintenance or fixes, although there may be a media charge unless the fix is obtained electronically through FixDist.

When to Update to the Next Level

Here are guidelines for updating to the next level:

- Update to the next fix level only when requiring a fix for a particular problem.
- Update to the next maintenance level when you need support for new hardware or want to apply preventive maintenance
- Update to the next version or release to take advantage of new or enhanced function

Overview of AIX 3.2.5 Common Mode

When an application is compiled on AIX, libraries are bound statically within the application. In AIX 3.2.5 and prior releases, these static libraries were compiled in Power Mode. When the application is run on a PowerPC, the Power instructions in those static libraries will be emulated in software. The application will run successfully, but the emulation may decrease performance slightly.

Compiling in *Common Mode* generates common binaries between the POWER, POWER2, and PowerPC processors. By building an application using this feature, the static libraries that are bound in the application will be compiled in full Common Mode. Since the application will have been built on AIX 3.2.5, it will be fully compatible with AIX 3.2.5. Also, since future releases of AIX will preserve the upward binary compatibility of executables wherever possible, the applications built using this feature will run well on future releases of AIX, including AIX 4.1.1.

We recommend that AIX Version 3.2.5 plus the above feature be the development platform for applications intended to run on AIX Version 3.2.x and AIX Version 4.1.1. In AIX Version 3.2.5, the IBM XL C Version 1.3, C Set ++ Version 2.1, XL Fortran Version 3.1, and XL Pascal Version 1.1.3 compilers support the common mode of compilation.

Important Note: This toolkit of libraries will provide full common mode if your application is compiled in common mode or if you do not bind shared libraries statically (override the shared attribute).

For more information, refer to the document *All About AIX 4.1*. Information is also available through IBM's World Wide Web server on the Internet. To access, open the following URL:

`http://www.austin.ibm.com/services/vendors/aix41/aix41.html`.

To receive the AIX 3.2.5 Common Mode Static Libraries, send a note to ibmspsc@austin.ibm.com or call the SPSC Support Line at 1-800-445-3440.

Optimizing Databases for SMP

By Stephen Horn and D. Britton Johnston

This article highlights the advantages of SMP in a transaction processing, database environment. It also takes an in-depth look at techniques for optimizing database scalability and enhancing database performance in an SMP environment.

IBM's combination of the PowerPC Architecture™ and the AIX 4.1 Symmetric Multiprocessing (SMP) capability is an example of an industry-wide commitment to support this new style of computing with complementary open hardware and system software offerings. Unlike past technology trends, vendors are trying to exploit the advantages of SMP and limit the disruptions of technology change. User organizations and software vendors can also facilitate this transition. By becoming more knowledgeable about SMP, they can leverage the technology to achieve its maximum benefit. Nowhere is SMP's leverage potentially greater than in database management.

SMP Advantages

Although SMP originated in scientific computing, its true advantages are in transaction processing and databases. The greatest advantage is the ability of different CPUs to handle multiple tasks simultaneously. Opportunities for parallelism exist throughout database management. Multiple disks, users, records, and requests can often be accessed, serviced, updated, and satisfied at the same (or nearly the same) time. SMP can deliver the greatest performance boost for this type of application. The big win occurs with databases because transaction loads and data structures can usually be partitioned so that more processing can be handled simultaneously.

The challenge of SMP is to effectively deliver the power of multiple CPUs working in parallel. Increasing the total number of processors increases the total processing power of a system—but not necessarily its throughput. All SMP systems follow a curve of diminishing returns, in which the extra throughput achieved by adding more CPUs becomes smaller with the addition of each CPU, as shown in Figure 1. This fall-off is due to the extra overhead of coordinating tasks among CPUs.

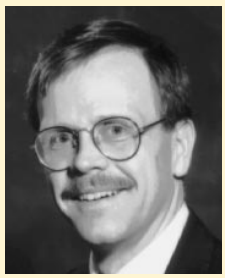
A system is considered scalable if its throughput increases in direct proportion to the number of CPUs added (as indicated by the 45° dotted line in Figure 1). The goal of optimization is to maintain the scalability curve near the 45° optimum (or 1:1 scalability) for as long as possible (Figure 2) using techniques at the system, database, and application levels.

Techniques for Optimizing SMP Performance

In an SMP machine, multiple CPUs execute multiple program flows called *threads* through a common memory state, as shown in Figure 3. Shared memory implies that more than one CPU can execute instructions from the same in-memory binary image and can access data from the same memory-resident structure.

The operating system controls the regions of memory that are accessed, when they are accessed, and which applications can access them. Most operating systems, however, grant limited memory allocation rights to applications. In fact, one benefit of an SMP-enabled Database Management System (DBMS) should be to hide the details of SMP so that applications can move unaltered from a uniprocessor machine to an SMP machine.

The key variables for optimizing database scalability in an SMP system are techniques for deferring or avoiding I/O, reducing memory contention, reducing locking overhead, and performing work in parallel.



Stephen Horn

D. Britton Johnston

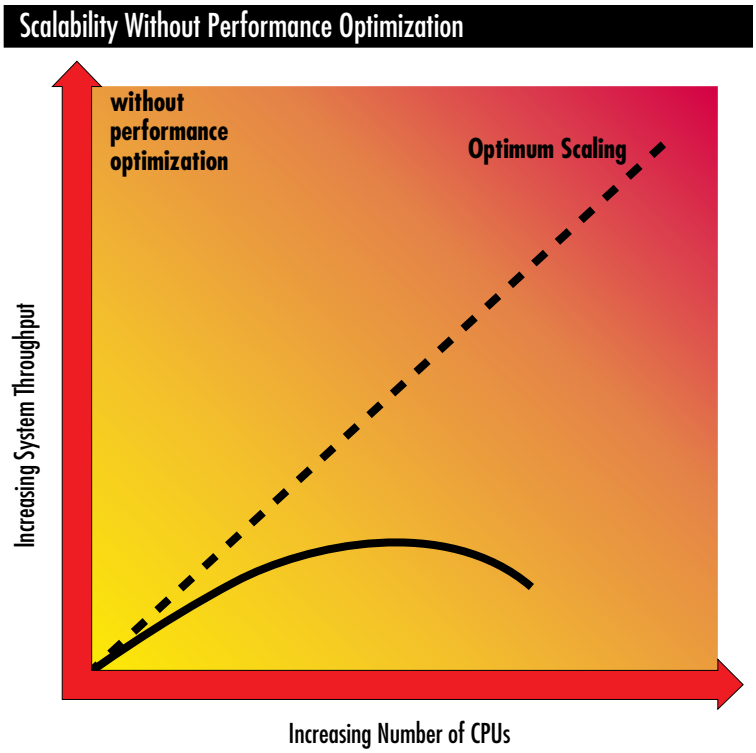


Figure 1. SMP scalability without performance optimization

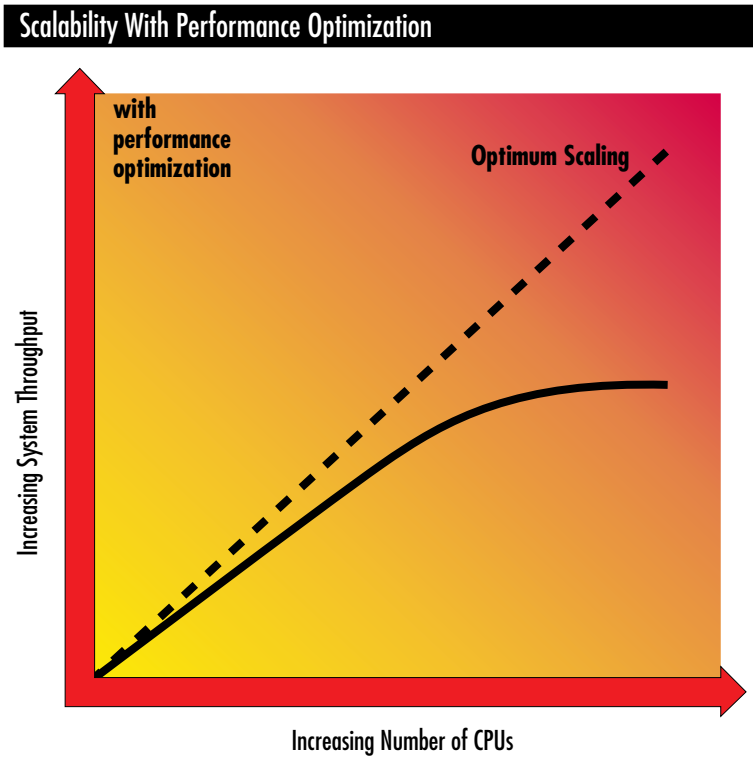


Figure 2. SMP scalability with performance optimization

Most DBMSs have features that manage (and allow users to manage) one or more of these performance parameters. The next section describes ways that applications can enhance database performance.

Enhancing Performance

The best way to optimize performance on a multi-CPU system is to start with a database that is already optimized for a single-CPU system. Many benefits automatically transfer to SMP by affecting one or more of the variables listed above.

Fine-Grain Locking

When one transaction locks a portion of memory, other transactions that depend on that portion of memory cannot complete. A fine-grain locking strategy limits the amount of memory and the length of time that memory is locked—thereby reducing its potential contention with other transactions. Three techniques to increase locking granularity are described below.

- **Multiple shared-memory locks**. Each data structure in shared memory can be controlled by its own lock rather than by a common lock shared with other data structures. Therefore, a client attempting to update the lock table to lock a record does not need to wait for another client process to finish updating the transaction table. Both processes can proceed in parallel. The resulting decrease in contention for shared memory can have a significant impact on throughput and response times, particularly on SMP machines with many users accessing the databases concurrently.
- **Granular transactions**. Only the parts of a transaction that write data require exclusive access to in-memory resources. By locking the data for only part of the transaction (that is, by making transactions more granular), the lock time can be dramatically reduced, and more transactions can use the same collection of in-memory structures at the same time.

  IBM's AIX 4.1 provides a special feature that supports granular transactions, an operating system feature that database system vendors should exploit. The AIX SMP architecture requires software-assisted cache coherency. By design, a database retains sufficient information to know when to resynchronize memory. By keeping this feature turned off until it is specifically required, the DBMS can further

reduce overhead in concurrent transaction processing on an SMP machine.

- **Record-level locking**. DBMSs that lock at the page or block level encounter serious contention problems when a transaction touches more than a few records. Since they must lock an entire block, all records in the block are locked for the duration of the transaction, possibly including tens or thousands of records not affected by the transactions. As more transactions begin, the frequency of contention increases exponentially, resulting in a massive gridlock of transactions waiting for needed records. This can devastate response time and throughput. If a transaction only locks a particular record, the frequency of contention is dramatically reduced.

Multiserver Operation

The ability of a DBMS to operate across multiple copies of itself against the same database is called *multiserver operation*. Organizations do not necessarily need an SMP machine to run multiple servers for the same database. In a distributed database, for example, separate physical databases stored at different locations in the network can be automatically linked together into a single logical database accessible to all tools and applications. However, the SMP machine with its multiple CPUs provides an ideal environment in which to run multiple servers. Users can achieve the parallelism of multiple servers without extra network overhead.

Multithreaded Operation

Multithreaded operation enables a server to accept multiple instruction streams from several clients simultaneously. Just as they have used multiple servers, organizations have run multithreaded DBMSs prior to SMP. The technique reduces CPU idle time (such as when a CPU waits for a disk I/O to complete) and serves multiple users concurrently. The architecture comes into its own, however, when threads have more than one CPU on which to execute, and can do so at the same time.

Spin Locks

Processes typically employ an operating system structure called a *semaphore* to request service from each other. Semaphores manage a queue of processes waiting for shared resources. When the resource is freed, the next process in the queue is granted access. In an SMP environment, semaphores can mean inefficient communication.

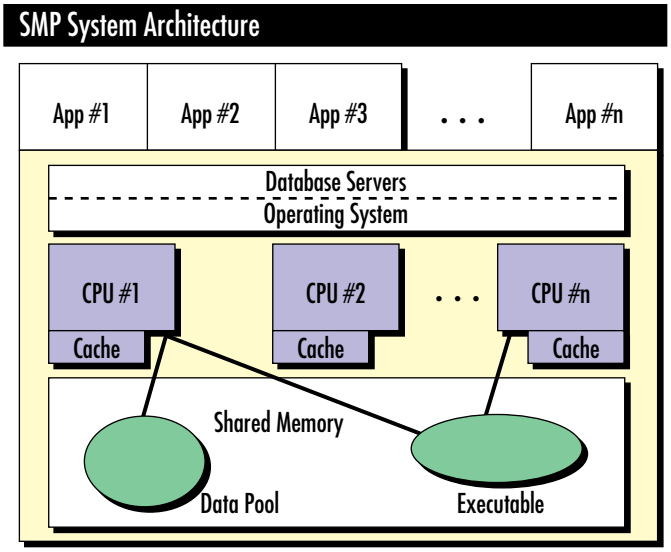


Figure 3. SMP system architecture

They generally involve the operating system and may require that one process sleep until the resource is ready for it (when it can then be reactivated by the operating system).

A spin lock differs from a semaphore. With *spin locks*, each process performs its own test for a requested resource. If the resource is being used, the process goes through a series of retries (the "spin"), checking for availability of the resource until it meets a preset (tunable) threshold, after which it resorts to a semaphore. The requesting process never has to relinquish its place in the run queue, go through the operating system, or change states (sleep/wake)—saving considerable time.

Asynchronous Page Writers (APWs)

This strategy also reduces the time that application processes spend waiting for resources such as disks. APWs are independent processes that periodically write updated database and log-file buffers to disk. If a server process needs to reuse a buffer, there is a high probability that a page writer has already written it to the disk. This means that the server process almost never has to perform its own I/O. APWs can be self-tuning to adjust the number of APWs that are active, the frequency in which they scan the buffer pool for modified pages, and the frequency in which they write dirty pages to disk.

Opportunistic Record Buffering

Another way to reduce disk I/O is to pack records from multiple transactions into a single network message to the client. This technique

also dramatically reduces the amount of network traffic, thereby speeding up disk accesses that are needed.

Larger Buffer Pool

The larger the memory buffer available to the DBMS, the more of the database that can be kept in memory. With more memory available, the probability is reduced that a particular part of memory will be required by two processes at the same time, thereby reducing the frequency of contention. A larger buffer pool also increases the likelihood that data will be held in memory, rather than on disk, so that the frequency of contentions caused by disk I/Os is also reduced.

What Programmers Can Do

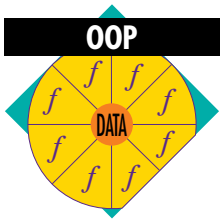
With database-centered applications, application designers can always do things to maximize performance. One is to eliminate *hot spots*—those places within the database that the application constantly targets for retrieval or update purposes. If every CPU constantly searches for the same data, all but one CPU will be left waiting in line—regardless of the strategies employed by system designers to avoid or reduce those waits.

The most significant aspect under a programmer's control is the selection of the machine, operating system, and DBMSs themselves. Know the SMP scalability curve for the particular configuration of hardware and software you are considering. Ask the vendors if they provide special features to extend that curve and if those features play off each other. SMP is a natural for database applications. Be certain that your DBMS is a natural for SMP.

Stephen Horn, Progress Software Corporation, 14 Oak Park, Bedford, MA 01730. Internet: stephen.horn@progress.com. Mr. Horn is the database product manager for Progress® Software. He has over 20 years of experience in developing applications and systems software in databases and transaction processing. He has a BA in Operations Research from the University of Massachusetts.

D. Britton Johnston, Progress Software Corporation, 14 Oak Park, Bedford, MA 01730. Internet: britt.johnston@progress.com. Mr. Johnston is the database development manager for Progress Software. He has worked on design and implementation of various RDBMSs for ten years. Mr. Johnston has a BS in Computer Engineering from the University of Connecticut.

Building an OO Development Environment on AIX



By Michael J. Branson and Eric N. Herness

This article describes an Object-Oriented (OO) development tools environment for a large OO project in IBM's programming lab in Rochester, Minnesota. It outlines the findings and outcomes, the types of tools assembled, the source of those tools, customizations made, and feedback received from users. It describes what did and did not work, and the requirements that led to the tools environment described.

This article provides a behind-the-scenes view of the tools environment necessary to support a large OO development project. A complete Object-Oriented Programming (OOP) environment includes a methodology supported by processes, tools, and an OOP language. Constructing a tools environment for a large-scale OOP project requires a variety of resources and a unique set of skills. Effectively deploying object technology requires timely and effective delivery of scalable tools environments.

The OO Development Strategy

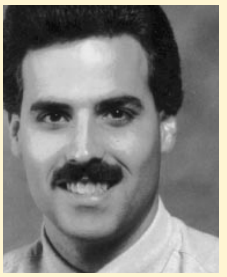
The OO development process must be integrated with a toolset that enables developers to be productive. The proper use of OO development tools is not as obvious as traditional tools because many are new, and there has not been a lot of experience with their use. Like traditional programming, developers depend on a process to guide them in using the tools.

Tools used in OO development must relate to the development methods and the development process used within a project. Tools by themselves do not replace a methodology or a

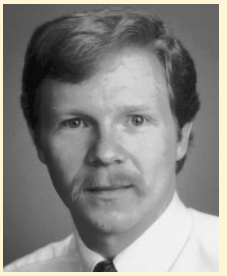
Project Description

The project that drove the development of this toolset was a large operating systems development endeavor. Because of the nature of this project, not all of our findings may be relevant to other projects. Here are the key characteristics that made our project unique.

- C++ was the implementation language for this project. Some tools are optimized to C++ and some problems may be unique to this development environment.
- It is a large-scale, object-oriented development project—150 direct programmers and dozens of affected programmers and supporting staff. The new C++ code written has more than 7,000 classes and more than one million lines of source code. About 12,000 compiles occur on the network each day.
- In addition, other smaller projects use the development environment, which adds hundreds of other users—making scalability a top priority.
- A legacy system is involved, so new C++ code had to interact with the old procedural code.
- There is no Graphical User Interface (GUI) component.



Michael J. Branson



Eric N. Herness

process. They are used to practice a methodology and to complete the deliverables of the development process.

Environment

The *development environment* includes the base hardware and software configurations used for development. It also encompasses the target execution environments of the software being developed. It is common to use a variety of hardware platforms with various operating systems to develop software products.

The environment and the tools are interdependent. For example, if the development workstations are a given, the tools must run on them. If, on the other hand, you are purchasing new equipment and base platform software, you should be influenced by where the OO development tools run best. The target execution environment also affects the requirements for compilers and build tools.

The development environment includes the base hardware and software configurations used for development.

Other Factors in the Transition to OO

Other elements essential to successful object-oriented development include education and training, the development process, and the development culture. Fundamental changes must occur in how software development is done when transitioning to object-orientation.

The development culture reflects changes experienced by the developers in the transition to OOP. Some workstation-based tools that accompany the OO development process are the most leading-edge tools available today—often motivating developers. On the other hand, tools used for OO development are so different from traditional tools that they require new thinking by developers. Discovering that you not only need new tools, but also a new toolbox, can be very frightening.

Integrating Tools with OO Development

There are many different OO methods. [1,2,3,4] Although each method takes a different approach to development, there are similarities between them. To define a toolset that supports the methodology choices of a diverse set of developers, a method-independent development process must be considered. A *method framework* lists the steps performed during development. [5,6] The following steps characterize the assumptions made by our team about the OO development process.

- **Derive candidate classes and objects**. This step encompasses domain analysis to produce candidate classes and objects. Refining and building a design model is more productive when candidate classes and objects are plentiful. Getting candidate classes and objects means understanding the problem domain and the requirements. It means carefully communicating with users of the system being built and with others having domain experience.
- **Build and refine a design model of those classes and objects**. The specific notation selected affects the exact name for this model; we call it the design model.
- **Build additional validation models to verify the evolving design**. Proceed with this step after a reasonable model of classes and objects has been formulated. These methodology-dependent models deal with more dynamic and functional elements of the design than the static model of the previous step. These models view the design model differently and help to validate and refine it.
- **Leverage existing classes, patterns, and frameworks**. Available class libraries are leveraged to describe the design and enable implementation. As details are added to the design model and other models are generated, the influx of components from the reusable class libraries continues.
- **Develop the class interfaces**. Creating the code-level interfaces to the classes can be an

[1]Booch, Grady. *Object-Oriented Analysis and Design with Applications*. Redwood City, California: Benjamin/Cummings Publishing Company, Inc. 1994.

[2]Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; and Lorensen, William. *Object-Oriented Modeling and Design*. Englewood Cliffs, New Jersey: Prentice-Hall. 1991.

[3]Jacobson, Ivar; Christerson, Magnus; Jonsson, Patrik; Overgaard, Gunnar. *Object-Oriented Software Engineering—A Use Case Driven Approach*. Wokingham, England: Addison-Wesley Publishing Company. 1992.

[4]Wirfs-Brock, Rebecca; Wilkerson, Brian; and Wiener, Lauren. *Designing Object-Oriented Software*. Englewood Cliffs, New Jersey: Prentice Hall. 1990.

[5]Branson, Michael and Herness, Eric. "The Object-Oriented Development Process." *Object Magazine*, 3(4), pp. 66-70. Nov-Dec 1993.

[6]Herness, Eric. "Object-Oriented Analysis and Design." *AIXpert*, pp. 40-44. May 1992.

additional verification step that may stimulate changes to the design done in previous steps.

- **Implement the classes**. Finally, probe the design decisions that have been encapsulated and left in the implementation. As implementations are written, additional small abstractions will be extracted. As additional uses for the component library are found, they should be reflected back into the design model.
- **Iterate through the steps until the design is robust**. Concurrency and iteration are inherent in the OO process. Each step can begin after a reasonable pass at the previous step has been made. A truly incremental and iterative process will take multiple passes through each step, producing workable code at the end of each pass. Second and succeeding passes will require less time in the earlier steps and more time in the latter, until a stable working system is attained.

The nature of the activities in an OO method framework require support from nontraditional tools. Figure 1 illustrates these activities and lists the kinds of tools that would be used to perform them. All of these tools must be available and integrated to support a large-scale object-oriented development project.

OO Tools

Many types of tools are needed to support OO development. Often, stand-alone products only work in isolation; many are not built to effectively support anything but small undertakings.

The following sections describe our tool choices and the customizations necessary to make the toolset both integrated and supportive of large-scale development.

CASE Tools

CASE tools are essential to deploy OO technology to a large project successfully. CASE tools that support analysis and design are helpful in the "Build Design Model" activity in Figure 1. They also support constructing additional models that validate the static model as shown in the "Build Validation Models" in Figure 1. Some CASE tools are optimized to one notation and method, while other tools can be used for several method and notation pairs.

The chosen CASE tool must not only support generating the appropriate models, but must also be integrated into the tools environment. Ideally, the CASE tool would accomplish the following objectives:

- Enable the developer to browse or edit the code that a class diagram models directly from the class diagram
- Load any code generated by the tool directly into the Configuration Management (CM) system
- Generate diagrams from the code in the CM system
- Enable multiple users to work on a set of design models simultaneously
- Control the design data by using the same CM and version control mechanism as the code

Besides integrating with the rest of the environment, the CASE tool should have a minimum set of additional characteristics:

- Scalable to support large projects in the thousands-of-classes range
- Have an underlying dictionary to allow the same entity to appear on multiple diagrams, but be defined only once
- Check for semantic inconsistencies between models and diagrams

CASE tools are essential to deploy OO technology to a large project successfully.

CASE tools must work with the documentation tools to easily generate the deliverables of development process activities. This may require some customization of the base CASE tool.

Variations on the Booch method were chosen for most of our development work. While methodology choice and application did vary, the Booch notation was agreed upon as a common documentation mechanism. When the project began in early 1992, no commercially available CASE tools supported Booch that met our requirements.

We used Cadre's® Teamwork® CASE tool that supported traditional development as an interim measure. Conventions were established and customizations were made to allow developers to build class diagrams and object diagrams using the traditional models available in Teamwork. When Rational ROSE™ became robust enough, developers moved to ROSE, which better supported object-orientation[7].

[7]Herness, Eric and Mitchell, Todd. "Using Cadre Teamwork for Booch Notation-based Design." 1993 Teamworkers Conference Proceedings (January 1993).

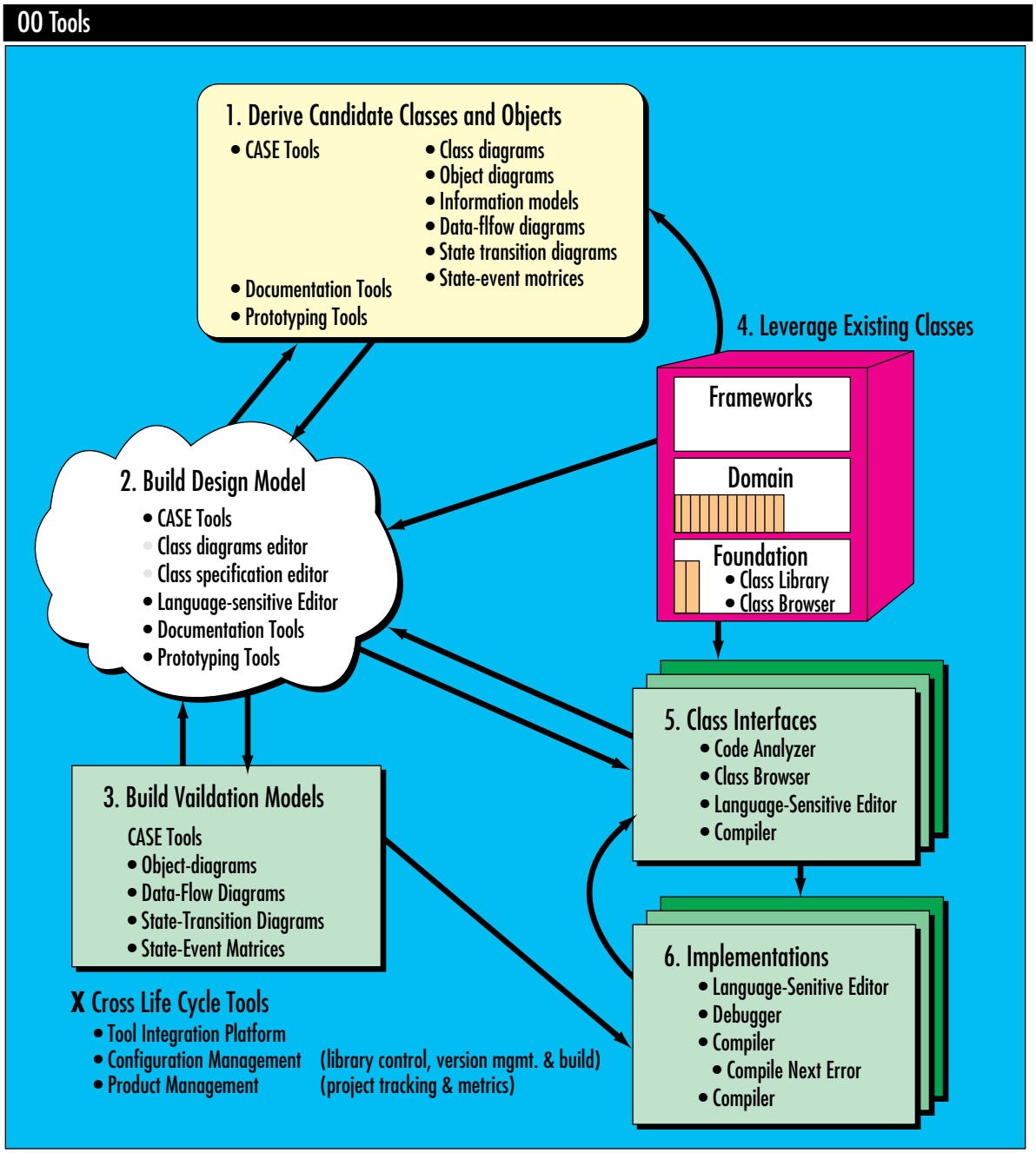


Figure 1. Object-oriented development—tool view

Our Customizations of CASE Tools

We made the following specific customizations to Teamwork:

- **Class specifications**. To tailor the Entity-Relationship Diagram (ERD) editor to our needs when working with class diagrams, we redefined the data dictionary to be used for Booch class specifications.
- **Information model**. We chose the ERD editor, also known as the Information Model (IM) editor to support class diagrams. Our users required an option for building Booch-like class diagrams. To make the data-dictionary editor more user-friendly, we added a pull-down menu that uses the SDE Workbench/6000-based editor to edit or add text to the data-dictionary entries.
- **Documentation**. To obtain documentation for a design, we added a pull-down menu to

retrieve all the notes and data-dictionary entries for a diagram, then send this information and the diagram to files that can be embedded into an inspection package. This enables the developer to obtain all documentation for a design with one menu command.

To translate class diagram information to C++ code files and to develop diagrams and code together in an iterative process, we added to both CASE tools the feature of bringing up an edit session on a file that corresponds to a class on the diagram. This is done by selecting a class on a class diagram, then selecting either Edit Source or Edit Header from a pull-down menu.

When this feature is integrated with SDE WorkBench/6000 and CMVC/6000, a search file (a special local customization made to the CMVC/6000 environment that will be described later) finds the file in CMVC/6000 when you select a class to edit its header or source code. The ability to check a file in and out of CMVC/6000 was added to the editor's pull-down menus.

Making the transition from Teamwork to ROSE was simple since we could reuse much of the customization. Developers demanded more support for the Booch notation and the semantic checking provided by Rational ROSE. To customize ROSE, we ported the customizations for editing source and header files. Supporting documentation required additional work because the open interfaces to ROSE are not the same as in Teamwork.

When we moved to Rational ROSE, we needed a mechanism to allow multiple users to access the same set of design diagrams. Diagrams were locked at the individual level with Teamwork. Because ROSE does semantic checks across diagrams, it locks at the project level. Since this was not acceptable for our purposes, we integrated ROSE with our configuration management system, allowing users to check out portions of a project they are working on.

It was considered too large an investment for us to add to ROSE the capability of generating code from design models. Although customizing CASE tools is difficult, it is not impossible, and yields benefits to the developers. If CASE tools are too hard to use, they will not be used, and that leads to a whole different set of problems.

Class Browser

Reuse is key to the OO development process. A class browser is used to understand the class library and the system that is being constructed. The "Leverage Existing Classes" in Figure 1 highlights the key role of the class browser: to assist in the search for existing reusable components and to analyze other classes for possible inclusion into the reuse library.

Integrating the class browser into the CM system is essential to finding reusable classes that emerge as the project proceeds.

Initially, we began using the C Set++ for AIX: Source Code Browser. Users rejected this browser after many attempts to adopt it for the following reasons:

- This browser requires classes to be compilable. It also requires a program to be written that uses any classes that will be browsed, because the compiler generates browser information. Although this requirement originally sounded harmless, we found that it made browsing code under development or browsing new code from an outside source very difficult.

- Since this browser offers a wealth of information and our project is large, a single browser session on all classes in the project is impossible to get running. Once it begins running, the performance is unacceptable.

- Our users did not like the GUI to this browser, so we developed our own browser. We produced a tool that did not exhibit the same problems as described above. The new browser was developed with considerable input from our user community. One key reason for the success of this browser is that it does not try to do everything that the C Set++ for AIX: Source Code Browser attempted to do. This yields much better performance and requires less from the user.

We may still use the C Set++ for AIX: Source Code Browser after the project is completed and in maintenance mode. It provides valuable information about the overall program structure, the use of classes, methods, and so on. This browser has also matured significantly since our project began.

Integrating the class browser into the CM system is essential to finding reusable classes that emerge as the project proceeds.

Code Analyzer

The code analyzer tool analyzes the interface and implementations written in the target language. It typically looks for inappropriate or high-risk use of language features. It should be tailored to enforce local coding conventions. The code analyzer must be able to do the following:

- Robustly parse and semantically understand standard C++ code
- Issue messages about source code that violates coding conventions
- Issue messages about source code that violate the rules of good semantic C++ programming, and warn about traps and pitfalls of the language
- Issue messages about syntax errors
- Tailor analysis to your project's standards and conventions
- Allow messages to be tailored
- Turn off messages about included files
- Suspend analysis of certain messages directly in the source code

Together with the CASE tools, the code analyzer is among the tools most closely integrated with the development process. Our OO development process shows that output from code analysis is required in addition to reviewing the models that are output from the CASE tools and the actual code. The code analyzer, therefore, must easily analyze a series of interfaces separately, or a series of interfaces and associated implementations. This output must be easily integrated with the documentation tools that are used to build the various design packages.

Flexibility is another required characteristic of the code analyzer. As coding conventions evolve and additional rules of analysis are determined, the analyzer must be easily modified to accommodate them.

Our project used a C++ code analyzer that we developed locally to enforce minor conventions such as naming, program structure, and so on. We also used the code analyzer to help programmers avoid some of the traps and pitfalls of C++. The analyzer was used to promote the use of C++ as an OOP language and to address considerations of the execution environment of the code being developed.

The code analyzer is a good educational tool for less-experienced C++ programmers. But some developers reject the "opinions" of such a tool until they are burned in a significant enough way to illustrate the importance of the analyzer.

Documentation Tools

Documentation tools must work together with the CASE and CM tools. Depending on the vehicle used to distribute and review the evolving object-oriented design, separate tools may be necessary. The key is simple design packages, versioning of design information, and flexibility in providing the necessary outputs. It is essential that developers focus on OO development rather than document creation.

Another consideration is the possibility of online verifications. Tools that make this easy will encourage additional inspection of material and can have a positive impact on the overall project.

Our project used traditional mainframe-based documentation tools and augmented them with diagrams from the CASE tools. Since generating review packages is still a manual task in our environment, each developer performed this task differently, often leveraging some of the CASE tool customizations that were described earlier. Although some online design and code inspections occurred, traditional review meetings with hardcopy material are still the norm.

Project Management

Project management tools assist in tracking inspections and collecting data related to the quality and productivity of the development project. Tight integration to the development process is required.

Dependency management capabilities are essential in project management tools. The incremental nature of the development process produces many dependencies. Understanding these dependencies is key to managing OO software development projects. The tools must provide management with the necessary information.

Some tools were effective when used as always, but others often required creativity in adapting them to the new methodology. We continue to use traditional project management tools because we considered supporting developers a higher priority than improved project management tools.

Prototyping Tools

Prototyping tools include tools for creating user interfaces, requirements validation, and design exploration.

Prototyping tools include tools for creating user interfaces, requirements validation, or design exploration. These tools should use the same language and class libraries as the final product, when possible.

No specific prototyping tool was available to our developers, but AIX (the target execution platform) has been used for prototypes. The software being developed will ultimately execute on another target, but AIX has been more convenient during prototypical stages. AIX has better debugging support than the project's execution

platform, and the tasks of linking and loading are much simpler.

Language-Sensitive Editors

A language-sensitive editor can provide assistance in reading and writing code and should be integrated with the CM, debugger, compiler, and CASE tools.

Our project uses the ez editor and related tools from the Andrew Toolkit. This editor provides a source view of C++ code that is knowledgeable of the language and its syntax. The editor formats different C++ constructs according to how the developer chooses to view the source. The source code editor is common between the CASE tools, CM tool, code analyzer, compiler, browser, and the debugger.

Class Library

A basic tool in the OO development environment is a foundation class library that contains types or classes that support basic data structures such as lists, queues, and stacks. The library should be optimized to the execution environments that are being targeted. Many foundation class libraries are available in the market today.

A GUI class library may also be necessary, depending on the domain of the system being built.

Our project has a foundation class library that was retargeted for the project's execution environment. The library supports the same interfaces for our prototyping and execution platforms. Applicable frameworks were not available at the start of the project. The development environment is currently being extended to better support development with frameworks.

Compiler

The compiler must be integrated into the CM and the build system. Integration with the editor and build system should support *compile/next-error*, a feature that places the user in the correct position within the source code when a compiler error message is selected.

Translators from C++ to C were used in the early stages of the project because no C++ compilers were available to support the execution target. Today, we use a common compiler front-end with support for multiple execution targets (to allow for prototyping).

Debugger

An integrated source-level debugger is essential for integrating the compiler, editor, and CM. The debugger must support object-orientation, not just offer what traditional debuggers have offered for years.

Our project used locally built debuggers to support our execution target. No commercially available debuggers support this target. Currently, the debugger on our prototyping platform is better than the debugger on the final execution platform because of the graphical way that the debugger on the prototyping platform displays objects. As debugger work continues, this difference in debuggers should diminish.

Most projects should be able to leverage commercially available debuggers, provided that they are open and can be integrated with the rest of the toolset.

Tool Integration Platform

An integrated source-level debugger is essential for integrating the compiler, editor, and CM.

The tool integration platform enables the integration of the rest of the tools described in this section. Tool-to-tool communication allows different tools to work together. Developers may look to a single source for all the tools at their disposal. The IBM AIX SDE WorkBench/6000 is an example of this type of tool, which becomes necessary because of the large number of tools and developers involved in a large-scale OO development project.

We integrated many tools into a single tool integration platform. Some tools required much work, while others offered integration with our platform as a feature.

We chose the IBM AIX SDE Workbench/6000 (Workbench/6000) to integrate tools. Each tool in the environment is registered with WorkBench/6000, which facilitates tool-to-tool communication and keeps developers focusing on the tasks of software development rather than the underlying tools that make up the environment. Developers do tasks necessary for development. Tools are launched by the system that support the tasks.

Three types of tools are involved in the WorkBench/6000 environment:

- **Tools shipped with SDE WorkBench/6000**: The basic tools shipped with the Workbench/6000 environment were modified to meet our requirements for function and our local preferences. First, we developed a replacement for the Workbench/6000 Development Manager (DM). This DM has more function than the product version and is more tightly coupled to our CM tool. Next, we upgraded our local editor to support the

necessary Editor Broadcast Message Server (BMS) messages. This gave our developers integrated support for the same editor that they were using before the release of the Workbench/6000-based environment.

- **Tools already integrated into SDE WorkBench/6000**: Integrating this type of tool is straightforward. Installation scripts normally accompany the tool and update the control files necessary to run the tool in an integrated mode. For example, control information found in the `.softinit` file and `softypes` directory is usually updated by these scripts. We use the Andrew File System (AFS®) and have a global version of these files that contain default values. We were often forced to modify these scripts to accommodate our distributed environment.
- **Tools locally or vendor-developed that are not integrated:** These tools required the most integration work. Vendor tools that were not yet integrated were a problem. We were usually unable to support all of the BMS messages required for a simple encapsulation because we did not have the source code for the tools. We defined a separate tool class for the class browser and Teamwork CASE to support starting the tool from either the Tool Manager or Development Manager.

Object-oriented projects put pressure on the CM system to support iteration and the structure of an object-oriented system.

It was easier to integrate locally owned tools because we controlled the source code. We integrated these to provide a consistent view of the tools development environment. Many developers were familiar with a consistent, integrated toolset on a mainframe they had used in previous projects. Having many tools with different user interfaces and usage semantics is very confusing. By integrating them, we could eliminate some confusion. Educating developers about these tools makes the transition easier.

Configuration Management

The CM system required for OO development is very important. Many previously described tools leverage the CM system to provide versioning and check-in/checkout capability—both essential in a large software project.

Object-oriented projects put pressure on the CM system to support iteration and the structure of an object-oriented system. This is necessary to effectively manage the source code, design the artifacts of the development process, and produce builds or drivers on a timely basis.

Our project initially used a Workbench/6000 integrated version of Revision Control System (RCS) called *softrcs*, but we quickly needed more capability than this entry-level tool provided. Migration to a customized CMVC/6000 gave us more functions and enabled us to control access to files and to structure our project so we could continue to support a growing number of developers. We combined CMVC/6000 with AFS and much add-on code to formulate our CM solution.

Before explaining the customization, a brief overview of the file tree is necessary. Our project worked with the hierarchical directory structure and used directory names to provide partitioning of the large amounts of data inherent in a project of this size. A strong relationship exists between directory names and the family—release and component qualifiers required for CMVC/6000. We architected the structure of the source file tree to match the high-level architecture of the system being constructed. This relationship enables developers to navigate, and easily, or at least methodically, find interfaces or implementations of the various classes that make up the system. There are architected places for interfaces that are exported from categories, highly reusable components, and local scoped classes. This facilitates reuse and provides a structure to understand the system.

Beyond this structure, the incremental development process requires that various levels of source code be worked on simultaneously. One driver can be in a test phase, while another is being coded, while yet a third may be in the design phase. All will have active code. Reducing developer confusion in this area and providing flexible compilation options were key goals.

Locating the right file (header file or implementation) is a complicated task that must be done efficiently. The compilation process and the CASE tools require this searching capability.

Once the basic structure was in place, and tools supported the movement and location of code within this structure, the `make`-based compilation model was modified to support the hierarchical structure and the various versions of source code that might be required for doing builds.

A complex set of new tools was written so developers did not have to deal directly with `make` files, while also providing fast and intelligent build capabilities that leveraged our distributed computing resources. Automated dependency analysis also eliminated build failures due to inaccurate information that might be supplied by developers in `make` files. This required that we add support to

Tool Evolution				
Tool Category	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**
Editor	ATK's ez	ATK's ez	ATK's ez	ATK's ez
Compiler	Internal tool (Prototype platform)	Internal tool (Prototype platform) (Beta for target)	Internal tool (Prototype platform) (Target platform)	Internal tool (Prototype platform) (Target platform)
CASE	Teamwork (Not integrated)	Teamwork	Teamwork ROSE	Teamwork ROSE
Library Control	softrcs	CMVC/6000	CMVC/6000	CMVC/6000
Debugger		Internal tool (Prototype platform)	Internal tool (Prototype platform) (Beta for target)	Internal tool (Prototype platform) (Beta for target)
Class Library		Internal tool (Prototype platform)	Internal tool (Prototype platform) (Target platform)	Internal tool (Prototype platform) (Target platform)
Browser		C Set++ for AIX: Source Code Browser	C Set++ for AIX: Source Code Browser	C Set++ for AIX: Source Code Browser Internal browser
Code Analyzer		Internal tool (Not integrated)	Internal tool	Internal tool

Figure 2. Phases of tool evolution

the basic CMVC/6000 product. Once completed, we had a complicated, yet usable environment that supports many developers.

Customization was critical to allow us to structure the files for developers to use while also supporting large-scale compilation throughput requirements necessary for doing full-driver and user-oriented builds.

The Development Platform

The choice of the development platform—where these tools run—is a key part of the OO development paradigm. This choice is influenced by the execution environments for the system being built and by the availability of the various tools on particular platforms. Our experience shows that a high-powered workstation on each developer's desk supported by a high-speed LAN is a critical part of the infrastructure necessary for the OO development process to be successfully utilized. This environment and the associated tools needed to support the OO development process must be understood, planned for, budgeted for, and appropriately integrated.

The next section describes which tools were available as the toolset moved from a loosely bound set of independent tools to an integrated OO development environment.

Tools Environment Evolution

The development environment evolved over time. The resources to develop, acquire, and integrate these tools were limited, so tools support was offered on a just-in-time basis. The iterative nature of the process, however, required that the toolset become completely available very quickly. Figure 2 shows the development phases of the toolset.

Phase 1. This phase allowed developers to design and code in a flexible environment. Some developers were apprehensive about moving to object-orientation and could not begin work until there was some sign of a real compiler on the execution platform. We delivered this to help the early adopters of the project with their design and prototyping. They also needed help in getting more developers accustomed to OOP.

Phase 2. The second phase was based on experiences and feedback from users of the first toolset. Although more complete, this toolset still lacked robustness, integration, and target support. A compiler that worked on the final target platform was necessary to show that the toolset could support this project adequately.

Moving to CMVC/6000 was also key in this phase. It was important to reach the final code repository as quickly as possible to reduce the future costs of migration. The code analyzer, class library, and class browser gave developers a complete toolset. Although integration was still lacking, getting these types of tools to developers was important so we could assess their value to the project and make our support and customization investments wisely. Since this was our first very large C++ project, some tools were not demanded by the developers, probably because they were unaware of them. It was our responsibility to make them aware of as many tools as possible.

Phase 3. This phase delivered the Rational ROSE CASE tool. Teamwork, with conventions for Booch-notation diagrams, no longer met our needs. As more developers moved from analysis to design and code, we needed the additional notation and function provided by ROSE.

The rest of the toolset was enhanced and integrated based on problems we encountered as the toolset was used. Simplifying the workflow associated with the development activity was driving enhancements in this phase.

Phase 4. This is the toolset we use today. We are still working on the compiler's optimization capability. We have made performance improvements, but the number of developers is also increasing. Because of this, performance work continues.

The object-oriented paradigm has great potential for building high-quality software systems when it is effectively complemented with an industrial-strength tools environment.

Future Directions

Use of the environment results in many suggestions for improvement. As the focus shifts from creating components to reusing them, the tools environment will also have to evolve.

Incremental compilation. Technology exists for incremental compilation and the requirements for it are usually obvious. In an environment like ours, it is a challenge to manage the information model and data storage requirements for the incremental compile scenario. We believe this requires fine-grained object management capabilities that are not yet available in OO tools environments that support hundreds of users.

Frameworks. The use of frameworks is also increasing. Framework development places added requirements on many tools, such as browsers, and requires additional types of tools that are not in our existing environment.

CASE technology. A round-trip forward and reverse engineering capability, properly integrated into the CM environment, can give more power to developers.

Formal measurements and tools. The metrics required for OO development are still not crisply defined. Good progress, however, is being made in this area, and the tools challenge will soon be to gather and present these measurements effectively.

Tighter integration between design tools and the CM system. With the necessary tool interfaces, the overall design of the system as described in the design tools can be the basis for organizing data in the CM system. The CM tool, on the other hand, can also be the repository for the design data. Our future goal is to keep all relevant development data (design, code, metrics, and others) in the same logical location in the CM system. All must be versioned, found, reused, and fed into the system build process.

Conclusion

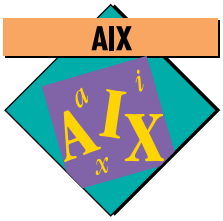
Constructing and successfully deploying a development tools environment for large-scale OO development is a major undertaking, and requires many different organizations to cooperate in order to build the appropriate infrastructure.

Object-oriented technology can effectively solve many small to medium-sized programming problems. The concepts of object-orientation can also be applied, with careful planning and investment, to the large-scale problems being solved by the large software development organizations prevalent in business today. The object-oriented paradigm has great potential for building high-quality software systems when it is effectively complemented with an industrial-strength tools environment. Customization for an organization's development needs will always be necessary, but hopefully products will evolve to allow for easy customization and integration.

Michael J. Branson, IBM Corporation, AS/400 Division, 3605 Highway 52 N., Rochester, MN 55901. Internet: mjb@rchvmw3.vnet.ibm.com. Mr. Branson is an advisory programmer, tools strategist, and object-oriented consultant to internal projects in Rochester. He has a BS in Computer Science from Purdue University.

Eric N. Herness, IBM Corporation, Personal Software Products Division, 3605 Highway 52 N., Rochester, MN 55901. Internet: herness@vnet.ibm.com. Mr. Herness is a senior programmer working in object-oriented systems development. He has a BS in Business Administration from the University of Wisconsin at Eau Claire and an MBA from the Carlson School of Management at the University of Minnesota.

POWER2 Server Performance Analysis



By Maurice T. Franklin, Jacob Thomas, and Sohel R. Saiyed

This article compares the relative performance strengths of two IBM POWER2 Architecture servers so that our customers can make more informed comparisons between these systems.

This article compares the performance of two POWER2 Architecture™ implementations: the RISC System/6000 (RS/6000) 590 and RS/6000-59H. Although the two servers operate at the same clock frequency, their designs are different. Figure 1 shows the key characteristics of these servers.

Both servers (590 and 59H) are identical at the processor level, making them completely binary compatible. Both can execute up to six instructions (one branch, one conditional register, two fixed-point, and two floating-point) per cycle and have the same Instruction cache (I-cache) size (32 KB). The key differences between the two are in memory hierarchy: Data cache (D-cache) size, D-cache line length, presence of an L2 (secondary) cache, and memory bus bandwidth (the 590 D-cache is configured for a 256 KB and 8-word memory interface only if a system has at least four memory cards; otherwise, it will operate with a 128 KB cache and a 4-word memory interface). Another difference between the two systems is cost; the 59H has a higher list price than the 590.

POWER2 Performance Monitor

Every system based on the POWER2 Architecture includes an embedded hardware performance monitor. The monitor consists of 22 hardware counters that can count particular events as they occur on the POWER2 processor, and a control register that allows users to select which events to monitor at any given time. The monitor provides many useful performance measurements, including the number of elapsed cycles and executed instructions, counts of cache and TLB misses, and utilizations of execution elements. The counters can be accessed using a special AIX kernel extension to obtain a precise view of the system behavior when a program is executing—while having a negligible impact on the performance of the program.

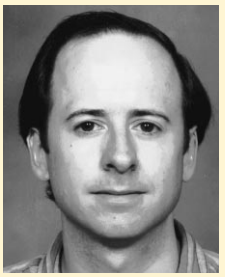
Benchmarks Measured

We analyzed the performance of three classes of workloads: floating point, integer, and Online Transaction Processing (OLTP). Three industry-standard benchmarks—SPECfp92™, SPECint92™, and the TPC-C benchmark—were selected to represent these three workloads. We believe these benchmarks represent the real workloads found in major segments of the server market worldwide.

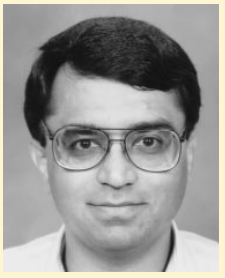
The SPECfp92 benchmark suite, published by the Standard Performance Evaluation Corporation (SPEC™), has five single-precision (32-bit) and nine double-precision (64-bit) floating-point programs. The suite represents the floating-point intensive workloads frequently encountered in workstation environments. Most codes in the suite consist of highly repetitive loops that operate on large sets of data accessed systematically.

A well-designed memory subsystem is key to the performance of this workload on the superscalar POWER2 processor, which can execute up to four floating-point operations per cycle. The fixed-point units, which also handle floating-point memory loads and stores, can perform two operations per cycle. The double-precision codes put an additional load on the memory subsystem since fewer of the 8-byte operands can be held in the D-cache, possibly requiring more frequent memory accesses.

SPECint92, an integer benchmark published by SPEC, consists of six integer workloads chosen from applications such as compilers and text pro-


Maurice T. Franklin


Jacob Thomas


Sohel R. Saiyed

Characteristic	RS/6000 Model 590	RS/6000 Model 59H
Clock frequency	66.7 MHz	66.7 MHz
Processor interface	8 Words	8 Words
Instruction cache size	32 KB	32 KB
Data cache/line size	256 KB/256 B*	128 KB/128 B
L2 cache	N/A	1 MB
Memory interface	8 Words*	4 Words

* Assumes four or eight memory cards

Figure 1. Key characteristics of the RS/6000 models measured

The design differences between the two POWER2 servers center on the memory hierarchy.

cessing. Unlike the SPECfp92 workload, in which the combination of floating-point and fixed-point units can rapidly consume the data, SPECint92 workloads are less sensitive to memory design since the two fixed-point units alone must execute all the arithmetic and memory load/store instructions, limiting the rate of memory references. Also, since integer data is only four bytes wide, the stress on the memory subsystem is less than for double-precision floating-point data.

The TPC-C benchmark simulates OLTP workloads. TPC-C, like the commercial workloads it represents, consists primarily of integer operations (such as pointer manipulation) with few repetitive loops and many branches to parts of the code not recently referenced. The OLTP executables are large (typically greater than 1 MB) and complex (two to three million instructions per transaction). As a result, the set of unique instruction addresses that are referenced with every transaction (the instruction footprint) is large. The combination of branching behavior and code path length makes the I-cache design a significant performance factor when executing the TPC-C workload.

These benchmark codes have a compilation aspect to their performance, which for many codes can often be improved by code optimization. Customers can use the RS/6000 compilers (C Set++ for AIX/6000 and AIX XL Fortran Compiler/6000) to perform advanced optimizations. Preprocessors such as KAP™ (Kuck and Associates, Inc.) and VAST™ (from Pacific-Sierra Research Corporation) are also available to optimize source programs. We used these products to increase the performance of the benchmarks within the rules specified by SPEC and the TPC-C council.

Processor Design Impact on Performance

The design differences between the two POWER2 servers center on the memory hierarchy. The Model 590 has a high-capacity Level-1 (L1) D-cache and an 8-word interface to memory, but no L2 cache. Model 59H has a large L2 cache (1 MB) with a smaller L1 D-cache and a 4-word interface to the memory. The L2 cache on the 59H is a combined instruction and data cache.

One key aspect of processor performance is the cache miss rate and miss penalty. Cache miss rates for a particular cache design (set associativity, cache size, and line size) vary between workloads depending on their memory reference pattern (for both data and instructions). For example, frequent non-localized branches increase the I-cache miss rate, while noncontiguous data references increase the D-cache miss rate. The cache miss penalty is the number of cycles to service a cache miss. The penalty depends on whether the missing item (instruction or data) is fetched from the L2 cache (requiring only a few cycles) or memory (a dozen or more cycles).

Another key aspect of processor performance is the degree to which a processor can perform instructions in a few cycles with as much parallelism as possible. This aspect of performance depends on the sequence of instructions encountered and the processor's superscalar design. The memory hierarchy and superscalar design aspects determine the average number of Instructions Per Cycle (IPC) completed by the processor during the execution of a workload. Since the 590 and 59H are identical at the processor level, only the changes in the memory hierarchy design affect the IPC. Lower miss rates or reduced penalties (or both) increase the IPC. Since the 590 and 59H have the same clock frequency, a higher IPC results in faster execution of the workload.

Performance Analysis

This section analyzes the relative cache miss rates and IPC of the three workloads gathered using the performance monitor. To highlight the performance differences, we show the relative increase/decrease in cache miss rates and IPCs.

Figure 2 compares the relative D-cache miss rates for the 59H and 590 for each of the SPECfp92 programs. The first column lists the name of the benchmark program. The second column shows the 59H to 590 L1 D-cache miss ratio: values greater than 1.0 indicate a 59H L1 D-cache miss rate higher than the 590 L1 D-cache miss rate. Column 3 shows the 59H L2 to 59H L1

D-cache miss ratio. Although the 59H L2 cache is a combined instruction and data cache, only the data miss rate is included in the ratio presented in column 3 (the I-cache miss rate is insignificant, generally for SPECfp92). Those ratios that are close to 1.0 in column 3 imply high 59H L2 data miss rates, with most of the misses from the L1 also missing in the L2 and going to memory. The IPC ratios of the two servers are presented in the fourth column. For benchmarks with IPC ratios less than 1.0, the 590 is better than the 59H.

Figure 2 clearly shows that the smaller D-cache on the 59H generally results in much higher miss rates than on the 590 for this workload. The high ratios in column 3 indicate that the 1 MB L2 on the 59H is not large enough for many of these benchmarks. Since the penalty to resolve references from the L2 is several cycles (compared to one cycle from the L1), a low miss rate from the L2 (low ratios in column 3) may not completely offset the high L1 miss rate of the 59H. Considering the IPC comparisons, the 59H can almost match and even occasionally exceed the 590's performance on these programs. Nonetheless, the L2 only marginally compensates for the smaller L1; in fact, the 590 is 4% faster on the overall SPECfp92 than the 59H. As a result, Model 590 has a slight performance and a significant price/performance advantage for this type of workload.

Figure 3 shows similar data for SPECint92 benchmarks. SPECint92 is moderately dependent on the L1 D-cache for high performance, but to a lesser extent than SPECfp92.

For the integer workloads, the 59H to 590 data cache miss ratios are smaller than some of the very large ratios (16.00 or 19.50) seen in some floating-point benchmarks (Figure 2). The L2 miss ratios are small for all SPECint92 benchmarks; therefore, most 59H L1 misses are L2 cache hits. For this type of workload, the caching of D-cache lines by the L2 effectively compensates for the smaller L1 D-cache on the 59H system. The overall performance of the two models is very comparable on this type of workload (the 59H is less than 1% faster), but Model 590 clearly leads in price/performance.

Figure 4 shows the same comparative data as the preceding figures for the TPC-C benchmark, with one significant addition. To highlight the importance of the I-cache for this type of workload, column 4 gives the L1 I-cache to L2 Instruction miss ratio on the 59H. The L1 I-cache miss rates of the 59H and the 590 are roughly the same since both servers have the same size (32 KB) I-cache, and so they are not compared. Since the 59H L2 cache is a combined cache, the 59H I-cache misses may be L2 hits. On the 590 system, all I-cache misses are resolved by accessing main memory.

As column 4 in Figure 4 indicates, most I-cache misses on the 59H are resolved by the L2. This implies the 59H design is highly effective in reducing the I-cache miss penalty, thus improving this workload's IPC. Since memory delays dominate TPC-C performance, these improvements significantly increase the IPC and overall performance of this type of workload. The resulting 16% improvement in performance seen in Figure 4 is dramatically higher than the overall through-

	Workload	Data Miss Ratio 59H L1/590 L1	Data Miss Ratio 59H L2/59H L1	IPC Ratio 59H/590
Single Precision	wave5	2.69	0.44	0.96
	alvinn	0.71	0.12	1.15
	ear	1.14	0.59	1.00
	mdljsp2	19.50	0.06	0.99
	swm256	1.91	0.90	0.96
Double Precision	spice	2.47	0.21	0.97
	doduc	16.00	0.14	1.03
	mdljdp2	3.00	0.07	0.99
	tomcatv	2.00	0.83	0.96
	ora	7.60	0.65	1.00
	su2cor	3.18	0.29	0.96
	hydro2d	5.06	0.18	0.97
	nasa7	1.57	0.38	1.13
	fpppp	3.20	0.13	1.01

Figure 2. Floating-point relative cache miss rates and IPC (SPECfp92)

Workload	Data Miss Ratio 59H L1/590 L1	Data Miss Ratio 59H L2/59H L1	IPC Ratio 59H/590
espresso	3.00	0.32	1.02
li	2.50	0.10	1.01
eqntott	1.92	0.22	1.05
compress	2.29	0.20	0.99
sc	4.66	0.23	1.01
gcc	3.00	0.35	1.07

Figure 3. Integer relative cache miss rates and IPCs (SPECint92)

Workload	Data Miss Ratio 59H L1/	Data Miss Ratio 59H L2/ 590 L1	Instruction Miss Ratio 59H L2/ 59H L1	Instructions Per Cycle 59H/590 59H L1
TPC-C	1.96	0.62	0.15	1.16

Figure 4. Transaction processing relative cache miss rates and IPCs

put changes of the other two workloads. When comparing configured system costs for workloads such as TPC-C, the 59H also has a clear price/performance edge over the 590. Therefore, the TPC-C workload—better than either SPECfp92 or SPECint92—illustrates the benefits of the design trade-offs made to the POWER2 implementation in the Model 59H to enhance its commercial performance.

Conclusion

All POWER2 Architecture implementations are identical at the processor level, differing only in memory hierarchy and bandwidth. These differences have varying impacts on the performance, depending on the characteristics of the workloads. The SPECfp92 workload type relies on a large D-cache and high memory bandwidth, and is handled slightly better by the Model 590. Generally, Model 590 exceeds the performance of the new 59H on traditional floating-point intensive workloads at a considerable cost advantage.

The middle ground for the memory design trade-off is seen in the SPECint92 workloads, in which the D-cache size is less critical, and the addition of an L2 can consistently, though marginally, improve performance. For such workloads, the two systems often have similar performance. For a transaction processing workload, the benefits of adding an L2 cache to the Model 59H are substantial and outweigh the loss due to the smaller D-cache and narrower bandwidth. For TPC-C and similar workloads characterized by large and complex integer manipulation code, the POWER2 Model 59H is clearly the best choice for a high-performance server, delivering higher performance and price/performance.

Together, these two POWER2 Architecture servers—both based on the same superscalar RISC technology and each with memory designs highly tuned to particular customer workload requirements—deliver world-class performance at competitive prices in their respective market segments.

References

- White, S. W. and Dhawan, S. "POWER2: Next Generation of the RISC System/6000 Family," *IBM Journal of Research and Development* 38 (1994): 493-502.
- Saiyed, Sohel R. and Thomas, Jacob. "The IBM POWER2 Architecture Implementations," *AIXpert* (August 1994): 55-57.
- Welbon, E. H.; Chan-Nui, C. C.; Shippy, D. J.; and D. A. Hicks. "The POWER2 Performance Monitor," *IBM Journal of Research and Development* 38 (1994): 545-554.
- Saiyed, Sohel R.; O'Connor, J. Michael; and Franklin, Maurice T. "POWER2: CPU-Intensive Workload Performance," *PowerPC and POWER2: Technical Aspects of the New IBM RISC System/6000* (SA23-2737): 129-136.
- Standard Performance Evaluation Corporation. "CINT92 & CFP92 Benchmark Descriptions," *SPEC Newsletter* 4, No. 4 (December 1992): 9.
- Transaction Processing Performance Council. *TPC Benchmark C Standard Specification Revision 2.0.* October 1993.
- Franklin, M. T.; Alexander, W. P; Jauhari, R.; Maynard, A. M. G; and Olszewski, B. R. "Commercial Workload Performance in the IBM POWER2 RISC System/6000 Processor," *IBM Journal of Research and Development* 38 (1994): 555-561.

Maurice T. Franklin, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: maurice@austin.ibm.com. Mr. Franklin is a staff programmer in the AIX Software Performance group, currently developing software tools for performance tuning of the Workplace operating system. He has a BA in Computer Science from the University of Texas at Austin.

Jacob Thomas, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: thomasj@austin.ibm.com. Mr. Thomas is an advisory programmer in processor and system performance in the RISC System/6000 Division. He has an MS in Statistics from Michigan State University and an MS in Physics from Birla Institute of Technology and Science in Pilani, India.

Sohel R. Saiyed, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: saiyed@hwperform.austin.ibm.com. Mr. Saiyed is an advisory engineer in the RISC System/6000 Division Systems Architecture and Performance group. He has an MS in Computer Engineering from Clemson University and a BTech degree in Electrical Engineering from the Indian Institute of Technology in Kharagpur, India.

TTY Subsystem in SMP

By Eddie Ho, Lance Russell, James Partridge, and Derwin Gavin

The AIX 4.1 multi-user terminal I/O subsystem has been reengineered to operate in a Symmetric Multiprocessor (SMP) environment. This includes the overall subsystem migration from BSD to System V Release 4 using the Streams framework. The new Streams-based TTY subsystem conforms to the new interface and standard and offers additional flexibility to meet today's multi-user system requirements.

AIX is a multi-user operating system that can be accessed by hundreds of users. The TTY subsystem is the part of kernel responsible for the data path between the running application and all I/O devices. The originating devices for AIX that use this subsystem can be grouped into three classes:

- **Networked/virtual devices** including `aixterm`, `xterm`, or applications such as `telnet/rlogin`
- **Physical devices** attached directly or through a multiport asynchronous adapter (Physical devices include ASCII terminals, serial printers, plotters, modems, and data collecting devices—any device with an RS-232 interface.)
- **System console** with low-function terminal (lft) emulation, primarily used as a terminal console for booting the system and for initial administration use

AIX 4.1 is SMP-enabled, providing serialized access to common data structures and efficiently using Streams and MP resources. As a result, protocol modules or device drivers have scalable performance on any processor. Other highlights of AIX 4.1 TTY subsystem include the following:

- Supports a single open-system terminal discipline (POSIX™) instead of three different flavors (BSD, AT&T, and POSIX), thereby eliminating confusion for administrators selecting a discipline
- Supports ISA bus-based native ports in the widely used RS/6000 Model 40P
- Supports the three native ports in the SMP hardware with console mirroring
- Supports native serial ports, the 8-port adapter, the 16-port adapter, and the 128-port asynchronous controller that is currently available in AIX 3.2
- Supports both BSD and AT&T pseudo-TTYs and their naming conventions, enabling applications to fully use the AT&T TTY functionality
- Supports the following communication protocols:
  - Terminals, printers, and plotters
  - Modems
  - Communication applications such as Serial Line Internet Protocol (SLIP), Asynchronous Terminal Emulation (ATE), and Communication Utility (CU)

Most existing application interfaces are compatible with AIX 4.1 except those using the IBM PC-RT® or IBM PC-specific `ioctls`. Equivalent functionality is provided using the standard interfaces, which allows your AIX 3.2 terminal I/O application to be migrated to AIX 4.1 without recompiling.

Figure 1 summarizes the AIX multi-user environment.

Streams Overview

Streams is a flexible set of tools for developing UNIX system communication services. It defines a generic message-driven queuing interface and provides a framework and tools for implementing communication services, such as a network pro-

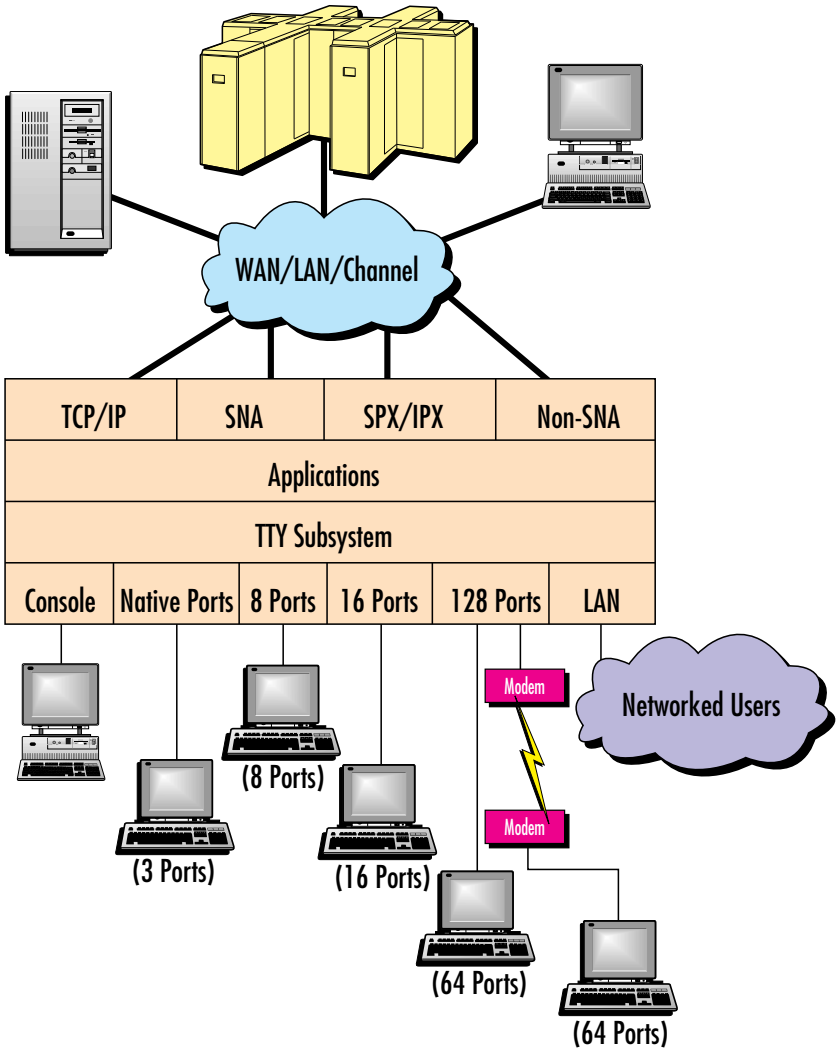


Figure 1. AIX multi-user environment

tocol stack. Streams does not impose any specific network architecture—it is simply a framework. The framework includes the following components:

- **Stream Head**: Part of the stream that interfaces to the user process
- **Stream Modules**: Kernel-level routines that implement the behavior of the supported protocol suite
- **Stream Device Driver**: The driver that handles the data between an external I/O or pseudo device and the AIX kernel

Figure 2 illustrates a simple transport model using Streams.

Benefits of Streams

Streams is a standard interface for developing modular, portable, networking layers that reuse code from different communications protocols. It facilitates the rapid growth of communications services. The Streams framework on AIX 4.1 conforms to OSF/1 Version 1.2. There are several benefits to Streams-based communication:

- The user application can customize the protocol stack in real-time by making the code path more efficient and direct for the application. For example, during printer configuration sessions, the printer discipline module is pushed rather than the terminal discipline module.
- Various protocol layers and device drivers can be shared and mixed without predetermined

restrictions based on the application requirements. For example, all terminal devices share the terminal discipline module, including the lft emulation.

- Runtime protocols can be substituted by the user application. For example, in the remote TCP/IP (SLIP) environment, the discipline module replaces the SLIP module in order to meet the protocol requirement.

TTY Subsystem Infrastructure

The TTY subsystem has many modules that conform to the Streams architecture. Applications define the stream stack based on a set of port requirements during the special file-open phase. Conversely, the stack is destroyed when the special file is closed, such as when the device is disconnected. The stack structure consists of the following:

- **Stream head**: Required for all stream stacks, it processes user requests; it is common to all Streams devices regardless of the nature of the implementation
- **Line discipline module**: Implements the session functions and supports the following types of protocols:
  - Terminal discipline (ldterm)—used by all terminal sessions, including lft emulation
  - Serial printer discipline (sptr)—used by all printer sessions except the parallel printer support
  - TCP/IP discipline for serial lines (SLIP)
- **Streams end**: Represents hardware-specific device drivers, including both Micro Channel® and ISA boards, and pseudo device drivers such as pseudo-TTY

The stream stack can contain other modules:

- **Ioctl processing** provides transparent `ioctl` processing before passing the message downstream.
- **Console mirroring** echos data from one serial port to another serial port (only applicable on SMP hardware with three serial ports).
- **Data mapping** allows I/O data to be remapped to a different value when a mismatch between the terminal hardware and the application's code page occurs. Typical environments include non-ISO-8859-1 ASCII terminals that are incompatible with the application

Streams Model

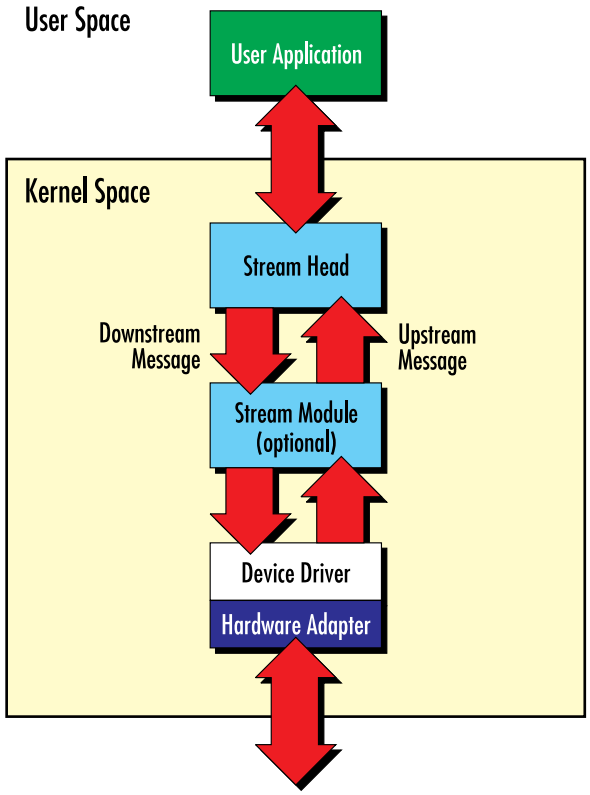


Figure 2. Streams model

or IBM Code Page 850 applications that are being accessed from an ISO 8859-1 terminal.

- **Code Page 932 converter** converts input data from Code Page 932 to IBM EUC-JP and vice versa; these modules will be pushed on the stack whenever the system locale is set to IBM 932 in Japan.

Figure 3 summarizes the construction of each protocol stack during runtime.

Changes to Commands in AIX 4.1

The AIX 4.1 `stty` command supports a single comprehensive discipline (POSIX) as defined by Open Software Foundation® (OSF®). This eliminates the need for AIX Version 3 subcommands, such as `stty-berk`, `stty-att`, and `stty-bsd`, which are therefore removed. Manipulation options (such as `add`, `del`, `disp`, and `get`), that accompany each subcommand are also removed.

An equivalent interface is provided via the Streams framework. For example, instead of using `stty get` to view the protocol stack, use the `strconf` command.

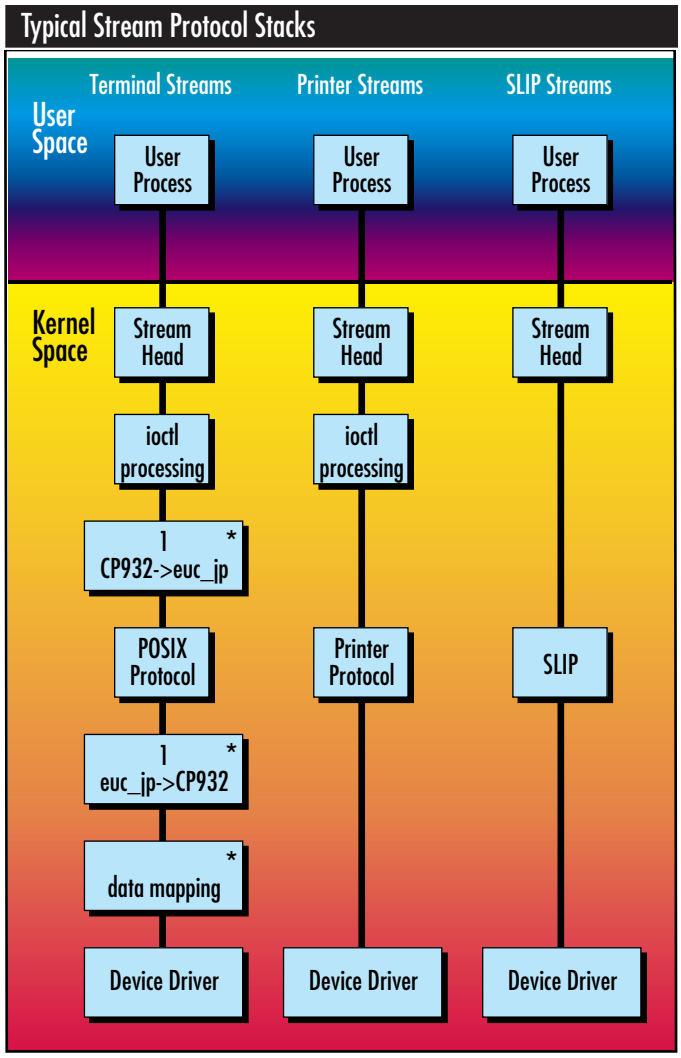


Figure 3. Typical stream protocol stacks for terminal, printer, and SLIP functions. (* *Optional modules*)

AIX 4.1 Header File

The AIX 4.1 header files are the same as in AIX Version 3:

- `ioctl.h` for BSD and AT&T programming interface
- `termios.h` for POSIX interface
- `termio.h` for AIX compatibility interface

An additional header file `termiox.h` is used for the new extensions to supplement the existing header files.

AIX 4.1 Programming Interface

The major programming interface changes are in two areas.

- Removes some PC-RT interfaces (`tcgcsmap`, `tcscsmap`, `txgbaud`, `txsbaud`) if there is an equivalent interface using the POSIX standard.
- Replaces `ioctls` that conflict with the Streams architecture (`txaddcd`, `txdelcd`, `txgetcd`, `txsetld`) with Streams `ioctls` such as `i_push`, `i_pop`, and `i_list`.

Maintaining Port State Information for a TTY

In AIX Version 3, TTY port attributes, such as line speed and character size, were persistent regardless of how the TTY driver specified those attributes. In AIX 4.1, TTY attributes specified using the `ioctl()` system call, the `tcsetattr()` routine, or the `stty` command persist only until the TTY driver processes a `close()` system call. The TTY attributes specified using the System Management Interface Tool (SMIT) or the `chdev` command set new TTY default values that are persistent until replaced by newer default values. The examples in Figure 4, which assume that `tty1` has been defined via SMIT to have a line speed of 38,400 bits per second, illustrate the behavior differences between AIX 3.2 and 4.1.

To summarize, SMIT and the `chdev` command set the default set of attributes, which persists across sessions and during reboots. The `stty` command `tcsetattr()` interface sets the runtime attributes for only as long as the device is open.

Differences in AIX 4.1 Pseudo-TTY (PTY)

Pseudo-TTY is the pseudo-driver used for network terminal communication, such as the `telnet` or `rlogin` command. There are two major differences in PTY usage for Version 4.1: TTY naming and usage conventions, and the configuration of AT&T PTYs.

Pseudo-TTY Naming and Usage Conventions

AIX Version 4.1 supports both BSD and AT&T PTY conventions. The key element of AIX 4.1 PTY is moving toward the OSF®standard and departing from the AIX Version 3 mixed BSD and AT&T naming standard. AIX 4.1 implements clone devices. The AIX Version 3 PTY implementation uses multiplexed files and allows a non-conventional mixing of BSD and AT&T naming.

AT&T Naming Convention and Calling Sequence: AT&T naming has one master—a clone driver—and *n* slaves. The PTY driver manages the connection between master and slave. When a user process opens a PTY line, it must follow this sequence:

1. Open the master (`/dev/ptc`).

Example 1: Changing Port Speed on Current Port

User Actions	Version	Results
1. Log in to `tty1`.	AIX.3.2.5	Line speed is 2400 bps.
2. Issue `stty 2400` from the terminal	AIX 4.1	Line speed defaults to the SMIT attribute—38,400 bps.
3. Log out.		
4. Log in again.		

Example 2: Changing Port Speed on Different Port

User Actions	Version	Results
1. The `tty1` is idle and there is no `getty` process running on it.	AIX 3.2.5	`tty1` is set to 2400 bps by step 2, resulting in the `cat` command displaying the data using 2400 bps.
2. Issue `stty 2400 < /dev/tty1` from User 1 at another terminal.	AIX 4.1	`tty1` is closed in step 2 by the `stty` command. `tty1` is returned to the SMIT default of 38,400 bps; thus, the `cat` command will attempt to display the data using 38,400 bps.
3. Issue `cat /etc/motd > /dev/tty1` from User 2 at another terminal.		

Figure 4. Examples of port state changes

2. Read the name of the slave associated, calling `ttyname()`for example, `/dev/pts/n (n=0,...,n-1` where `n` is a PTY-configurable parameter).
3. Open the slave.

Each time a user process calls `ttyname()`, it gets the first free slave (for example, if the first slave free is the slave number 4, it gets `/dev/pts/4`). The slave cannot be opened before a master because the user process in AT&T naming would not know the name of the slave before opening the master.

BSD Naming Convention: The master and slave have the same number in BSD naming. The connection between master and slave is determined by a naming convention. Since each master has its associated slave, a user process can first open either the master or the slave. The order is not important and there is no clone master.

AIX 4.1 PTY Naming and Usage: All AIX 4.1 applications should be using the AT&T naming convention—the OSF standard. Although AIX 4.1 supports the BSD naming convention to facilitate BSD migration, the BSD and AT&T naming conventions cannot be mixed, even though this is allowed in AIX Version 3.

AT&T PTY Configuration

AIX Version 3 system administrators do not have to control the number of AT&T PTYs from the SMIT PTY configuration panel; they are automatically generated based on demand. The maximum number of PTYs is based on system resources, such as the number of inodes. In AIX 4.1, the SMIT configuration controls the number of AT&T PTYs, with a default of 256. System administrators must allocate more PTYs if needed by the application.

Eddie Ho, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Ho is a senior programmer in the AIX Executive Briefing Center. He has a BS in Computer Science from the University of Wisconsin and an MS in Computer Science from North Dakota State University.

Lance Russell, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Russell works in AIX kernel development. He has a BS in Computer Science from the University of California at Berkeley and an MS in Computer Science from the University of Minnesota.

James Partridge, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Partridge works in AIX communications development. He has a BS in English from the University of Texas in Austin.

Derwin Gavin, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Gavin is a senior associate programmer in AIX communications development. He has a BS in Computer Science from Grambling State University in Louisiana.

SystemGuard Processors for Remote Operations

By Kanti C. Shah and David F. Rittinger

This article describes the operation of the SystemGuard processor shipped with every IBM SMP system. Using the SystemGuard processor, system administrators can manage the server operations locally or remotely—even when the network, operating system, or individual hardware components are not functioning.

IBM's Symmetric Multiprocessor (SMP) products are designed to be servers in commercial environments. Commercial servers, shared by many users, typically run critical applications in which data integrity and availability are very important. Since many servers operate in unattended or remote locations, providing remote diagnostics and service is important while, at the same time, protecting access to sensitive data.

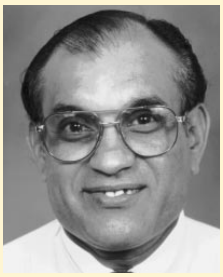
IBM's SMP family of servers includes a SystemGuard processor as a standard feature, making it possible to manage the server operations locally or remotely. In addition, it automatically alerts IBM service personnel to system problems and allows troubleshooting to be performed remotely.

Some reliability, availability, and serviceability features of IBM SMP products are as follows:

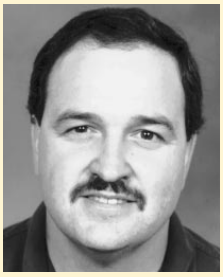
- **Data integrity**: Extensive parity checking is used on system cache memory and all internal and external buses. All IBM SMP systems use Error Checking and Correcting (ECC) memory subsystems.
- **Low downtime**: Hot pluggable/swappable disk drives are available on the Model J, which increases the uptime for the system.
- **Remote maintenance**: All facilities that can be accessed locally are accessible remotely. Menu-driven maintenance sessions aid in quick diagnosis and fault isolation.

  Critical problems are automatically reported to the technical support/service center for fast turnaround and service. The SMP system can operate with less than a full complement of resources (such as CPUs and memory), thereby increasing availability. Performance is reduced, but the system can remain operational until repairs are made.

  All remote accesses are password-protected at several levels. There is a quick disconnect feature in case of an attempted intrusion. A remote session can be mirrored at a local site or administration center.
- **Quick fault isolation**: Firmware-resident diagnostics are run when certain failures prevent the system from booting. Stand-alone diagnostics provide for quick and efficient disk and I/O fault isolation. Since online diagnostics can execute concurrently with applications and predict potential failures, preventive maintenance can be scheduled.
- **Configurability**: Permissible accesses are configurable via user settings in offline mode and from AIX service aids. An interface between internal firmware and the operating system makes these tasks transparent to users and fully executable from System Management Interface Tool (SMIT), service aids, or the command line.



Kanti C. Shah



David F. Rittinger

SystemGuard Processor

The SystemGuard processor controls the server functions listed in Figure 1.

Firmware in the SMP servers (independent of the operating system) interfaces with the SystemGuard processor to perform the following types of functions:

- Power system management
- Interface with operator panel
- Vital Product Data (VPD) management
- Management of test system during POST
- IPL process control
- Offline maintenance sessions
- Remote service access

SystemGuard Processor Access

The SystemGuard processor operates on separate power from the server, so administrators can work with the SystemGuard processor even when the server is off.

The SMP servers include three asynchronous serial ports. Two ports can attach consoles, either locally or remotely, to support operations and maintenance functions; the third port can attach an Uninterruptible Power Supply (UPS). These serial ports can also be used for other customer applications when they are not being used for the functions described above.

The SystemGuard processor can be accessed with an ASCII terminal (or PC with terminal emulation) that is locally or remotely connected to one of the serial ports. With a modem on two ports, IBM service personnel can remotely connect to one serial port while a system administrator monitors activity from the other serial port. Therefore, all remote console activity is mirrored so it can be monitored locally. Customers can control all access authorization using authorization flags, passwords, and mirroring of console sessions.

A system administrator can display and change the system operating mode (Normal, Secure, Service), issue a power-on command, and monitor the hardware boot sequence from an ASCII terminal, either locally or remotely. These functions remove the need for on-site personnel to perform operations such as turning keys, pushing buttons, and reading display codes at the operator panel.

The SystemGuard processor interfaces to the main system processor via interrupts, with polling aided by a mail box in NVRAM and a JTAG bus for initialization, and logging out machine internal states if a catastrophic hardware error halts the system operation.

Server initialization
Power system management
Power-On Self Tests (POSTs)
System configuration
Boot/Initial Program Load (IPL) process management
Diagnostics testing
Firmware updating
Electronic-mode switching (Normal, Secure, Service)
Error handling
Automatic server recovery
Remote operation
Remote maintenance
Automatic problem reporting
Security parameters

Figure 1. Server functions controlled by Support Processor

The interface to the operator panel, power system, and VPD EEPROMs is through the industry-standard I2C bus. A local, proprietary system-specific bus interfaces with programmed-code ROMs, flash EEPROM, NVRAM, and other direct and system I/Os, such as Time of Day (TOD), serial ports, and diskette drive.

Stand-by Mode

When the server is turned on, the SystemGuard processor executes built-in core tests to ensure all base components are functioning. By reading the VPD, it sets up a configuration table that is used to schedule Power-On Self Tests (POSTs).

In Standby mode, using menu-driven maintenance sessions, IBM service personnel—either locally or remotely, if remote access is authorized—can perform problem determination functions such as the following:

Electronic mode setting: If the machine's key is in normal position, the system can be placed in Service mode electronically from either a local or remote console. Maintenance menus are accessible only in Service mode. The Standby menu is displayed when a preset keyword is entered.

Read/display system configuration: Server configuration data can also be electronically read and displayed on the operator panel LCD by activating the reset button, or displayed on the attached console.

Set configuration: Processor and memory can be disabled or deconfigured under user control. The expansion units can be configured logically and physically.

Setting operational and test mode flags: Options can be set to bypass maintenance menus in Service mode and proceed with IPL, bypass POSTs, perform extended tests, and so on.

Other functions: Service personnel can also enable/disable remote service authorization, read/write NVRAM, read TOD, verify (read/write) operator panel display functions, check power system status, adjust voltage margins, and verify that power can be applied to the system unit and attached devices.

The following sections describe three phases completed by the SMP system before booting the operating system.

Power-On Phase

The power-on sequence can be started using one of three methods:

- Power-on push button on the operator panel
- Power-on command string from a local or remote console
- Preprogrammed TOD parameter

This last power-on feature is useful in saving energy during service-off hours, while allowing the system to be ready well before the service-on time begins.

The SystemGuard processor schedules and launches each POST on a selected system processor and gathers the results.

System Initialization Phase

When the power-on sequence begins, the SystemGuard processor starts the INIT phase. This initializes all system hardware and establishes the system configuration tables for various hardware components such as the CPU or Central Electronic Complex (CEC), memory boards, and Micro Channel adapters.

Firmware Update Phase

Normally, the system uses the code resident in the flash EEPROM. But if the SystemGuard processor determines that its checksum is not valid and cannot update the firmware (the key is not in service position or the correct diskette is not installed), the system will detour to the backup EPROM. Although the backup firmware has limited functionality, it allows the system to boot the operating system in uniprocessor mode. Then, the AIX filesystem and tools can restore or rebuild the firmware in the flash EEPROM. These operations are also possible from a remote center. In service mode, if a diskette is inserted in the drive, the SystemGuard processor will automatically update the flash EEPROM.

Power-On Self Tests

The integrity of each principal hardware component is tested by running POST diagnostics. If an error is detected by any of these quick-confidence POSTs, extended POSTs are run to further isolate the problem.

The SystemGuard processor or the main system processor runs these tests, but the test system is entirely managed by the SystemGuard processor. The SystemGuard processor schedules and launches each POST on a selected system processor and gathers the results. Any errors that occur are logged and the test sequence continues. The operator panel display and the consoles indicate test progress. The SystemGuard processor performs the self tests on directly attached I/O ports (not Micro Channel adapters or devices attached to I/O ports), the power system, and fan connections. The main processor complex executes the POST on I/O ports shared with the SystemGuard processor, cache, memory subsystem, interrupt system, SCSI adapters, and LAN adapters. Memory sharing, cache coherency, system I/O sharing, and arbitration are also verified in multiprocessor mode.

The failure of a non-critical server component, such as one of the processors or a memory segment, does not prevent the server from booting AIX. The SystemGuard processor deconfigures failing resources, so the IPL can continue using any available processor. This is a significant enhancement to system availability. Replacing a failed component can be done later.

During POST, all memory is tested, optimum memory address mapping and interleaving are determined and configured, and an IPL control block including memory map is built for the operating system.

Critical errors, such as failing access tests to minimum resources, no functioning processors, insufficient working memory, or a parity error on any bus may prevent the server from booting. Not only are such severe errors reported on the consoles attached, but a trouble call is also automatically placed to the remote service center (if the system is configured for call home and appropriate authorization flags are set).

Service Mode

If the power-on sequence was initiated with the system operating mode set to Service, the hardware boot sequence can display the maintenance menu or continue to IPL in Service mode. If the system boots in Service mode, the operator can

choose single-user mode, diagnostic mode, or the RAM filesystem. Once exited from a single-user mode, the INIT default allows the operator to enter the normal multi-user mode.

In Service mode, an offline maintenance menu provides two levels of access—general maintenance and customer-privileged access—to set passwords and remote authorization parameters.

From the offline maintenance menu, a system administrator or authorized IBM service representative (local or remote) can set hardware configuration data, display hardware error-log information, enable or disable the service console, perform a system reset, issue a power-off command, select the device from which the system should continue booting, run internal diagnostic tests, display or change system parameters, or select the language for maintenance menus (English, German, Spanish, French, Swedish, Norwegian, Belgium/Dutch, or Italian).

The menu-driven interface allows the administrator to define the command strings or passwords that must be entered to remotely power-on the system. They can set voltage margins, security access passwords, phone numbers, and service-line speeds. Special reserved commands modify VPD EEPROMs after a field upgrade. Administrators can also disable or enable remote power-on command strings.

Offline diagnostics are resident in the system—independent of the operating system—and can be performed even if the system fails to boot the operating system. Offline diagnostics allow the administrator to build a test suite consisting of a sequence of resident tests, and to select specific test parameters for stressing the system or isolating a fault. Extensive diagnostic tests can verify whether the server can boot AIX, whether all installed resources are functioning, and whether the resources can be shared by the server's processors. Tests to verify the wiring interconnects across printed wiring assemblies can be run.

These offline diagnostics tests can be run in a loop mode, or they can be halted on every error and continue upon user command. These tests can display detailed test status messages on the consoles. If the test results are accumulated, they can be analyzed later using maintenance aid procedures to identify a failing replaceable unit.

Automatic Problem Reporting

During the hardware boot sequence, the SystemGuard processor controls the running of Built-In Self Test (BIST) and POST routines to ensure that hardware components, such as CPUs, ECC memory, and data paths are operating properly. If the test routines encounter a failure that prevents the machine from booting, the SystemGuard processor automatically reports a problem to the IBM Service organization. A problem management record will automatically be opened in the IBM RETAIN system (an automated system for tracking problems and dispatching service personnel), which is monitored by the IBM Support Center. The IBM Support Center can log in remotely, perform maintenance, and run diagnostics.

Security

All remote operations, maintenance, and automatic problem-reporting functions of the SMP product family have access authorizations that the customer can control, including the following:

- Attachment of remote consoles
- Activation of automatic problem reporting
- The ability to dial-in to the server to perform remote operations or maintenance
- Customer-assigned command strings to enable power-on and power-off
- Customer and maintenance passwords to control function access
- Console mirroring that allows the activity on the remote service console to be visible on the customer console
- The ability to disable remote sessions immediately by resetting flags or changing system operating mode

The SMP system also maintains an event log of all call-out and login attempts.

The menu-driven interface allows the administrator to define the command strings or passwords that must be entered to remotely power-on the system.

Console Mirroring

During offline sessions before AIX is up, and during the online runtime phase in Service mode, the serial port device drivers allow mirroring of both consoles. Input is accepted from both of them, and output is sent to both.

SystemGuard Processor Interaction With AIX

The SystemGuard processor communicates with AIX using a specialized protocol. Interacting with AIX, the SystemGuard processor can control automatic server recovery, allow a technician to remove and replace hot-plug disk drives (on Model J), select server operating mode (Normal, Secure, or Service), configure CPUs (processor

start, stop, disable, and enable), alter the SystemGuard processor operating parameters, manage flash EEPROM update, log single-bit correctable memory errors and failing addresses, monitor status of server fans and temperature sensors, and control system shutdown, power off, reset, and restart.

If a severe hardware component failure is detected in a CPU or an unrecoverable memory error occurs, the SystemGuard processor will automatically scan the internal and boundary states of the processor and memory subsystem complex for later analysis. It will initiate server recovery by running internal tests, deconfiguring the failing resource, and then rebooting AIX. Using the `surv_m` command, the SystemGuard processor can monitor the AIX heartbeat. If the operating system hangs and cannot generate the heartbeat, the SystemGuard processor will detect this condition and automatically initiate a reboot of AIX, if the operator has enabled surveillance mode.

Using selections from the AIX Diagnostic Service Aids menus, a system administrator can display and alter numerous server parameters, such as server operating mode, CPU allocation and deallocation, security parameters (flag settings, passwords, console visibilities), and Support Processor dial-in and dial-out telephone numbers.

Summary

The SystemGuard processor ensures that system administrators and authorized service personnel have access and control of the server, even when the network, operating system, or individual hardware components are not functioning.

The capability of remote operations for power-on, power-off, reset, system operating mode, and display of operator panel messages eliminates the need for personnel on-site, eliminates travel to unattended locations, enhances system administrator productivity, and improves system availability.

The automatic recovery functions of the server reduce the time needed to identify problems and improve system availability.

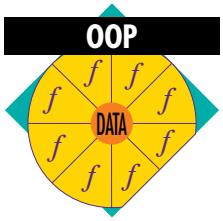
With hardware failures reported automatically—directly to the IBM Service organization—and the ability of IBM service personnel to remotely access configuration data, error logs, and run diagnostics for problem determination, server downtime and serviceability timeframes are reduced and system availability is improved.

Kanti C. Shah, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Shah, an advisory engineer, has been involved in various phases of design and testing of the SMP family of products. He has a BS in Electrical Engineering from Sardar Patel University in India and an MS in Electrical Engineering from the University of California at Berkeley.

David F. Rittinger, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Rittinger is a senior service planner involved in the remote support strategy and Reliability, Availability, and Serviceability (RAS) architecture testing of SMP. He has an MBA from Fairleigh Dickinson University in New Jersey.

Building Object-Oriented Frameworks

Part 1



By Deborah Adair

By using frameworks1, Taligent®, Inc.—an independent joint venture of Apple® Computer, IBM, and Hewlett-Packard—is realizing the full promise of object technology. A framework defines the behavior of a collection of objects, providing an innovative way to reuse software designs and code. Part 1 of this two-part series describes different types of frameworks and how they are used and addresses organizational concerns that impact framework development. Part 2 will outline a process for identifying and designing frameworks.

This article presents a process for developing frameworks and highlights four important guidelines:

- Derive frameworks from existing problems and solutions
- Develop small, focused frameworks
- Build frameworks using an iterative process driven by client[2] participation and prototyping
- Treat frameworks as products by providing documentation and support, and by planning for distribution and maintenance

While the process and techniques discussed are ideal for developing in the Taligent CommonPoint™ application system, they can also be applied to other object-oriented programming projects.

This article is intended primarily for software developers and designers in commercial, corporate, and higher education software development organizations. However, hardware designers, strategic technology planners, and technical managers might also find it useful.

Understanding Frameworks

A framework captures the programming expertise necessary to solve a particular class of problems. Programmers purchase or reuse frameworks to obtain such problem-solving expertise without having to develop it independently.

Framework Development

Developing a framework differs from developing a stand-alone application. A successful framework solves problems that appear different from the problem that justified its creation. The problem-solving expertise must be captured so that it is independent of both the original problem and the future solutions in which it is used; however,

Deborah Adair

[1]A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions. Taligent has designed frameworks for system software functions, such as networking, multimedia, and database access. With frameworks, software developers do not have to start from scratch each time they write an application. Frameworks are built from a collection of objects, so both the design and code of a framework can be reused.

[2]This article is written from the viewpoint of framework developers. We use the term "client" to refer to developers who might use a framework that you developed.

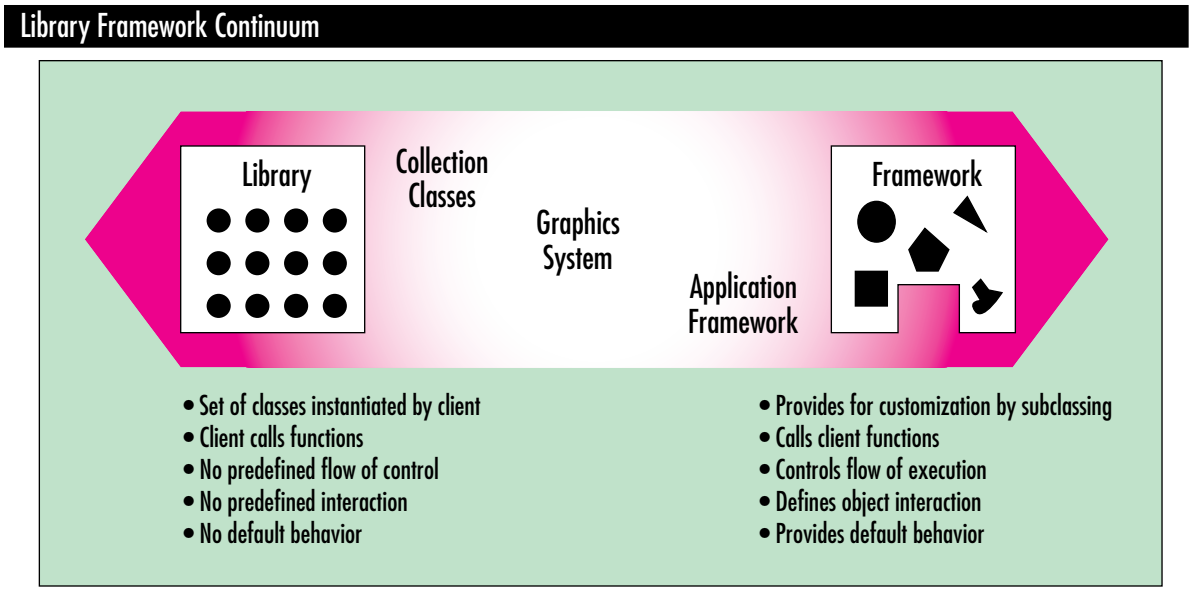


Figure 1. Library framework continuum

each program that uses the framework should appear to be the one for which it was designed.

Developers must clearly identify the class of problem that a framework addresses. For your clients to adapt the framework to new problems, they must understand the solution the framework provides and how to incorporate it into their programs. Because others have to understand how to use your frameworks, it is critical that you follow good software design practices.

Describing Frameworks

While clear differences exist between class libraries and frameworks, some libraries exhibit framework-like behavior, and some frameworks can be used like class libraries. As shown in Figure 1, this can be viewed as a continuum, with traditional class libraries at one end and sophisticated frameworks at the other.

Frameworks can be characterized by the problem domains that they address, as well as by their internal structure.

Framework Domains

The problem domain that a framework addresses can encompass application, domain, or support functions.

- **Application frameworks** encapsulate expertise applicable to many programs. These frameworks encompass a horizontal slice of functionality that can be applied across client domains. Current commercial Graphical User Interface (GUI) application frameworks, which support the standard functions required by all GUI applications, are one type of application framework. (The Apple MacApp™ system and Borland's OWL™ system are two such frameworks.)
- **Domain frameworks** encapsulate expertise in a particular problem domain. These frameworks encompass a vertical slice of functionality for a specific client domain. Examples of domain frameworks include a control systems framework for developing manufacturing control applications, a securities trading framework, a multimedia framework, or data access framework.
- **Support frameworks** provide system-level services, such as file access, distributed computing support, and device drivers. Application developers typically use support frameworks directly or after modification by systems providers. Support frameworks can be customized, such as developing a new file system or device driver.

The Taligent CommonPoint application frameworks extend across all three categories. Similarly, you can create frameworks that capture your own problem-domain expertise. Whether you are developing custom applications in a corporate setting or developing a suite of commercial applications as a software vendor, building and using frameworks can increase productivity. The frameworks you build will usually be domain or application frameworks.

Using the Framework

- Set box color
- Give box contents
- Draw box and contents


Drawing a default check box

```
TBox box;
box.SetColor(TRGBColor(0,1,0);
TCheckContents check;
box.SetContents(check);
box.Draw();
```

Extending the Framework

- Derive from contents class and override DrawContents


Extending the framework to draw an X instead of the default check

```
class TXMarkContent : public TBoxContent{
public:
   virtual void DrawContents(TGrafPort&port)
};
void
TxMarkContent::DrawContents(TGrafPort&port)
 {
   TLine line1(TGPoint(0,0),TGPoint(10,10))
   Tline line2(TGPoint(10,0),TGPoint (0,10))
   line1.Draw(port)
   line2.Draw(port);
}
```

Figure 2. Using and extending a framework

Framework Structures

Identifying the high-level structure of a framework makes it easier to describe the behavior of the framework and provides a starting point for designing framework interactions. For example, some frameworks are manager-driven—a single controlling function triggers most framework actions. You call the controlling function to start the framework, and the framework creates the necessary objects and calls the appropriate functions to perform a specific task.

An application framework generally uses a manager object to take input events from the user and distribute them to the other objects in the framework.

Another categorization, based on how a framework is used, is whether you derive new classes or instantiate and combine existing classes.

Architecture-driven frameworks rely on inheritance for customization. Clients customize the behavior of the framework by deriving new classes from the framework and overriding member functions.

Data-driven frameworks rely primarily on object composition for customization. Clients customize the behavior of the framework by using different combinations of objects. The objects that clients pass into the framework affect what the framework does, but the framework defines how the objects can be combined.

Frameworks that are heavily *architecture-driven* can be difficult to use because they require a substantial amount of code to be written. Purely data-driven frameworks are generally easy to use, but they can be limiting.

One approach for building frameworks that are both easy to use and extensible is to provide an architecture-driven base with a data-driven layer. Most frameworks provide ways for clients to use the built-in functionality and also modify that functionality. Typically, clients use a framework's built-in functionality by instantiating classes and calling their member functions. Clients extend and modify a framework's functionality by deriving new classes and overriding member functions.

Figure 2 shows a simple example of using and extending a framework. Suppose you have a simple framework that supports, among other things, drawing shapes. Clients might use it to draw a check box or extend it to draw other types of boxes.

Maximizing Framework Benefits

The key to maximizing the benefits of writing frameworks is to get as many developers as possible using them. Frameworks are a long-term investment; the benefits gained from developing frameworks are not necessarily immediate because framework designers need more time to create a framework than a procedural library, and clients need more time to learn a framework than a procedural library.

Managing Dependencies

Projects can often be divided into several separate frameworks and assigned to small teams. If it takes more than three or four programmers to produce a framework, it can probably be split into a set of smaller frameworks. Teams of two to four are usually more effective than teams of one, unless the single programmer is both an experienced framework developer and a domain expert.

Working with several small teams has its challenges:

- Programmers are focused on one aspect of a large project and might not understand all the interrelationships and client implications.
- Architectural consistency must be maintained across teams.
- Dependencies between frameworks can create bottlenecks.

There are several ways to alleviate these problems:

- Appoint a project architect who maintains the "big picture" and ensures that the frameworks ultimately work together.
- Follow standard design and coding guidelines.
- Decouple the frameworks by isolating the dependencies in intermediary classes.

Often when one framework requires the services of another, the connection can be implemented through an interface or server object. Then, only one object is dependent on the other framework. Until the other framework can support the necessary operations, the intermediary class provides stub code that allows the rest of the framework to be tested. Loosely coupled frameworks are generally more flexible from the client's perspective.

Designing Successful Frameworks

Successful frameworks have the following characteristics:

- **Complete**—Frameworks support features needed by clients and provide default implementations and built-in functionality where possible. Provide concrete derivations for the abstract classes in your frameworks and default member function implementations to make it easier for your clients to understand the framework and allow them to focus on the areas that they need to customize.
- **Flexible**—Abstractions can be used in different contexts.
- **Extensible**—Clients can easily add and modify functionality. Provide hooks so your clients can customize the behavior of the framework by deriving new classes.
- **Understandable**—Client interactions with the frameworks are clear, and the frameworks are well documented. Follow standard design and coding guidelines and provide sample applications that show the use of each framework. If the developers who build frameworks and those who use them follow the same guidelines, it facilitates reuse in both directions.

The most important consideration when designing frameworks is ease of use for the client programmer. From the client's perspective, an easy-to-use framework performs useful functions with no effort. The framework works with little or no client code, even if the default implementations are simply placeholders, and it supports small, incremental steps to get from the default behavior to sophisticated solutions. However, it is harder for clients to work around bad abstractions than invent their own. If you do not know how to solve a problem in a reusable way inside your framework, leave it out.

A framework does not necessarily have to meet all of these requirements to be successful; but if it does, it will be easier to convince other programmers to use the framework. When you design a framework, also look for ways to minimize the potential for client errors and enhance portability:

- Simplify clients' interactions with the framework to help prevent client errors. Make any client requirements as clear as possible in both your interfaces and documentation.
- Isolate platform-dependent code for easier portability of your framework. Designing for portability reduces the impact of porting on your clients.

This approach can also be used to isolate platform-dependent code or code that accesses a specific application's framework. To use different platforms or application frameworks, only one piece of the framework needs to be changed. Intermediaries can also be used to access legacy data or non-framework services.

Delivering Your Framework as a Product

Even if your framework will only be used internally, it must be treated like a product. Documenting your framework is an important step in the development cycle, but you must also plan how the finished product will be distributed and supported.

To use your frameworks, other developers need to know that they exist and how to access them. Unless a process is established for providing and distributing frameworks, it is difficult for other developers to use them. Reuse does not just happen; your organization must actively support and encourage it. One way to do this is to reward programmers for writing and distributing frameworks that others can use. Even more important, reward the programmers who use them.

Ideally, all frameworks are kept in a central repository, and a repository manager is responsible for notifying clients about new frameworks and updates to old ones. With a large enough repository, selecting the appropriate frameworks becomes an integral part of developing new program solutions.

To realize the benefits they can provide, your organization must commit resources to support frameworks. You must be able to assist your clients and respond to their problems and requests.

Over the lifetime of a framework, the cost of supporting it actually becomes a benefit—the support cost of one framework with three dependent applications will be less than the cost of supporting three independent applications with duplicate code. The more applications that use a framework, the larger the savings. Over the long term, using frameworks can reduce support and maintenance costs.

Early in its life, a framework will probably require routine maintenance to fix bugs and respond to client requests. Over time, even the best framework will probably need to be updated to support changing requirements.

Part 2, which will appear in the May 1995 issue, will focus on developing frameworks. For the latest information on Taligent, explore the World Wide Web page //http:www.taligent.com.

Taligent Standards

Early on, Taligent realized the need for common coding standards throughout the company. The chief architect began compiling a list of do's and don'ts that were required reading for all new engineers. This compilation has evolved into *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++*, published by Addison-Wesley. This book, which contains the guidelines followed by the Taligent engineers in the development of the Taligent CommonPoint application system, provides an excellent basis for your own coding standards.

Reuse does not just happen; your organization must actively support and encourage it.

Recommended Reading

The following references include standard object-oriented design references, new publications, and articles about frameworks from a variety of periodicals.

- Beck, Kent. "Patterns and Software Development." *Dr. Dobb's Journal* 19, no. 2 (February 1994): 18.
- Beck, Kent and Johnson, Ralph. "Patterns Generate Architectures," *European Conference on Object-Oriented Programming* (1994).
- Birrer, Andreas and Eggenschwiler, Thomas. "Frameworks in the Financial Engineering Domain: An Experience Report," European Conference on Object-Oriented Programming (1993): 21-35.
- Booch, Grady. "Designing an Application Framework," *Dr. Dobb's Journal* 19, no. 2 (February 1994): 24.
- Booch, Grady. *Object-Oriented Analysis and Design With Applications*. Redwood City, CA: Benjamin/Cummings, 1994.
- Campbell, Roy; Islam, Nayeem; Raila, David; and Madany, Peter. "Designing and Implementing 'CHOICES': an Object-Oriented System in C++," *Communications of the ACM* 36, no. 9 (September 1993): 117.
- Coad, Peter. "Object-Oriented Patterns," *Communications of the ACM* 35, no. 9 (1992): 152.

- Eggenschwiler, Thomas and Gamma, Erich. "ET++ SwapsManager: Using Object Technology in the Financial Engineering Domain," *OOPSLA '92 Conference Proceedings*, ACM SIG Notices 27, no. 10 (1992): 166.
- *Frameworks: The Journal of Software Development Using Object Technology*. Software Frameworks Association.
- Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. "Design Patterns: Abstraction and Reuse of Object-Oriented Design," *European Conference on Object-Oriented Programming* (1993): 406-431.
- Gamma, Erich; Helm, Richard; Johnson, Ralph; and Vlissades, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Forthcoming.
- Goldstein, Neal, and Jeff Alger. *Developing Object-Oriented Software for the Macintosh*. Reading, MA: Addison-Wesley, 1992.
- Johnson, Ralph. "How to Design Frameworks," *OOPSLA '93 Tutorial Notes*, 1993.
- Mallory, Jim. "TI Software Speeds Semiconductor Production," *Newsbytes NEW07200011* (July 1993).
- Nelson, Carl. "A Forum for Fitting the Task," *IEEE Computer* 27, no. 3 (March 1994): 104.
- "Semiconductor Industry: New Advanced C.M. software from Texas Instruments Expected to Revolutionize Semiconductor Manufacturing," *EDGE: Work-Group Computing Report* 4, no.166 (July 1993): 22.
- Shelton, Robert. "The Distributed Enterprise," *The Distributed Computing Monitor* 8, no. 10 (October 1993): 3.
- Stroustrup, Bjarne. *The C++ Programming Language*. 2d ed. Reading, MA: Addison-Wesley, 1991.
- *Taligent's Guide to Designing Programs: Well-Mannered Object-Oriented Design in C++*. Addison-Wesley, 1994.
- Wilson, Dave. "Designing Object-Oriented Frameworks." Personal Concepts, Palo Alto, CA 1994.
- Wong, William. *Plug & Play Programming, An Object-Oriented Construction Kit*. M&T Books, 1993.
- Taligent White Papers
  A Study of America's Top Corporate Innovators, Taligent, Inc., 1992.
  Lessons Learned from Early Adopters of Object Technology, Taligent, Inc., 1993.
  Driving Innovation with Technology: The Intelligent Use of Objects, Taligent, Inc., 1993.
  Leveraging Object-Oriented Frameworks, Taligent, Inc., 1993.

Deborah Adair, Taligent, Inc., 10201 North De Anza Boulevard, Cupertino, CA 95014-2233. 1-800-288-5545. Ms. Adair is a staff technical writer in the Taligent Technical Communications Department. She received her BS in Scientific and Technical Communications from the University of Washington.

Porting Fortran Between Cray and the IBM RS/6000

By Emad ElHamahmy and Luc Chamberland

XL Fortran Version 3 includes several extensions that enable developers to port their CRAY Fortran code with minimal recoding. Using the compiler options and routines described in this article will make your move from CRAY1 to the RISC System/6000 as easy as possible.

Most of the language extensions discussed in this article can be coded using Fortran 90, which is supported in XL Fortran Version 3. When you are writing new code, adhering to Fortran 90 is preferable because it ensures portability across any platform that supports Fortran 90. Of course, to port code that has already been written, using the extensions would likely require fewer changes than recoding your applications.

You can use the following XL Fortran compiler options and features to meet your porting needs.

Luc Chamberland

Emal ElHamahmy

The -qintsize Option

The `-qintsize` compiler option sets the default size of integer and logical data entities whose sizes are not explicitly declared. This option allows you to port code easily from 16-bit and 64-bit machines to 32-bit machines.

For example, the following declaration explicitly specifies that `x` has a size of 8 bytes:

```
integer(8) x
```

But the next declaration specifies that the default integer size be provided for `x`. For XL Fortran, the default size of integer/logical entities is four bytes, while for CRAY it is eight bytes.

```
integer x
```

If `-qintsize=8` is specified at compile time, `x` is declared as a 64-bit entity. This ensures that very large numbers (larger than the maximum `integer(4)` value) are not inadvertently truncated because of a different default size.

The `-qintsize` option affects the following:

- Integer/logical literals that do not specify `kind` type parameters
- Default integer/logical data objects and functions
- Intrinsic function results of type default integer/logical
- Typeless constants in integer contexts

Fortran 90 alternative to the `-qintsize` option: The Fortran 90 `KIND` intrinsic function returns the `kind` type parameter of an argument. It specifies the representation method for the argument. You can then use the `kind` type parameter when declaring entities. For example,

```
integer (kind=kind(0)) x
```

indicates that the size of `x` is the default integer size for the compiler. Compiled by a CRAY compiler, `x` is 8 bytes long; compiled by XL Fortran, `x` is 4 bytes long.

To ensure that entities have similar value ranges without specifying platform-specific information during type declaration (such as a `kind` type parameter), use the Fortran 90 `SELECTED_INT_KIND` intrinsic function. This function returns a `kind` type parameter value of an

[1]All references to CRAY in this article refer to the Y-MP™ hardware line and the CF77™ and CF90™ Fortran compilers.

integer data type that represents all integer values n in the range $-10^R < n < 10^R$. For example:

```
integer (kind=selected_int_kind(18)) x
   ! XL Fortran kind type parameter: 8
   ! CRAY kind type parameter: 8
```

The -qrealsize Option

Similar to -qintsize, the -qrealsize compiler option sets the default size of real and complex entities whose sizes are not explicitly declared. The default real size in XL Fortran is 4 bytes, while the default on the CRAY products is 8 bytes.

Keep in mind that the precision of entities of type DOUBLE PRECISION is twice that of the default real size. Thus, if you specify -qrealsize=8, DOUBLE PRECISION entities have a default size of 16 bytes. Similarly, the size of complex and double complex entities is affected because they are formed from parts of type real.

Fortran 90 alternative to the -qrealsize option: As for integer entities, the KIND intrinsic function is an alternative way to ensure portability for real entities:

```
real (kind=kind(0.0)) r
```

KIND(0.0) always returns the kind type parameter of the processor's default real size.

Similarly, the SELECTED_REAL_KIND intrinsic function returns the kind type parameter value of a real data type with a minimal decimal precision and exponent range.

The -qautodbl Option

Use the -qautodbl compiler option to convert single-precision entities to double-precision entities, and double-precision entities to extended-precision entities. Figure 1 shows suboptions.

To help maintain proper storage relationships when code is ported, some suboptions will pad as well as promote objects, shown in Figure 2.

NOTE: The -qrealsize is disregarded if you compile with both the -qautodbl and -qrealsize options.

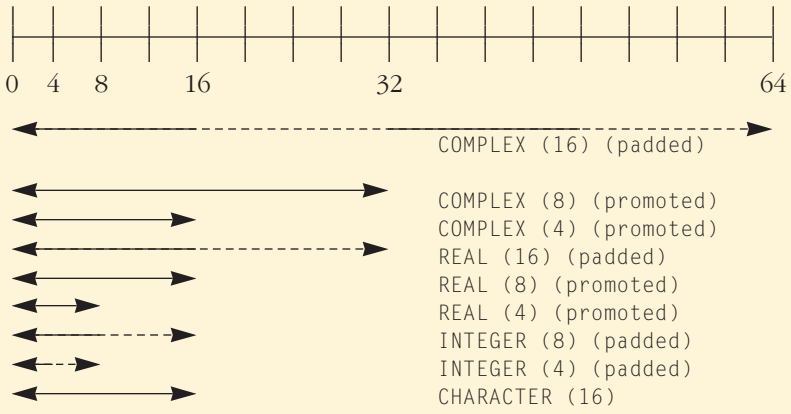
Figure 3 illustrates promotion and padding using -qautodbl=dblpad. Figure 4 shows how data types are mapped in memory when -qautodbl=dblpad is used.

Subscription	Function
none	Does not promote or pad any objects that share storage. This is the default for -qautodbl
dbl4	Promotes 4-byte floating-point objects, including objects composed of these objects, to 8-byte objects. For example, a complex(4) object is promoted to complex(8).
dbl8	Promotes 8-byte floating-point objects (including objects composed of these objects) to 16-byte objects.
dbl	Performs the same promotions as both dbl4 and dbl8.

Figure 1. Suboptions to promote objects

Subscription	Function
dblpad4	Performs dbl4 promotion, and also pads objects of other types (except character) if they possibly share storage with promoted objects.
dblpad8	Performs dbl8 promotion, and also pads objects of other types (except character) if they possibly share storage with promoted objects.
dblpad	Performs the promotions and padding done by both dblpad4 and dblpad8.

Figure 2. Suboptions to pad objects



The dashed lines represent the padding.

Figure 3. Storage relationships with -qautodbl=dblpad

```
@process autodbl(dblpad)
      complex(4) x8      /(1.123456789e0,2.123456789e0)/
      real(16) r16(2)    /1.123q0,2.123q0/
      integer(8) i8(2)   /1000,2000/
      character*5 c(2)  /"abcde","12345"/
      common /named/ x8,r16,i8,c
      end
@process autodbl(none)
      subroutine s()
         complex(8) x8
         real(16) r16(4)
         integer(8) i8(4)
         character*5 c(2)
         common /named/ x8,r16,i8,c
           x8    = (1.123456789d0,2.123456789d0)   ! promotion occurred
          r16(1) = 1.123q0                         ! padding occurred
          r16(3) = 2.123q0                         ! padding occurred
           i8(1) = 1000                            ! padding occurred
           i8(3) = 2000                            ! padding occurred
            c(1) = "abcde"                         ! no padding occurred
            c(2) = "12345"                         ! no padding occurred
      end subroutine s
```

Figure 4. Mapping of data types with -qautodbl=dblpad

Integer Pointers

Integer pointers are the XL Fortran version of CRAY pointers. XL Fortran integer pointers reference the address of a variable, which enables a program to step through part of storage or overlay parts of storage. The following example represents an XL Fortran integer pointer:

```
pointer (p,x)
```

The `p` is the integer pointer, which references the address of `x`, a variable or array declarator (generally called the *pointee*)

An integer pointer is a scalar variable of type `integer(4)` that cannot have a type explicitly assigned to it. The integer pointer can be used in any expression or statement in which an `integer(4)` can be used.

The XL Fortran compiler does not allocate storage for the pointee. Storage is associated with the pointee at execution time by assigning the address of a block of storage to the pointer.

The following subsections describe the differences you must consider when porting Fortran code between CRAY and the RISC System/6000.

Pointer Size

XL Fortran integer pointers are 4 bytes in size, whereas CRAY pointers are 8 bytes. If an absolute address is assigned to a pointer, ensure that the address can fit in four bytes of storage and that it is valid on the RS/6000.

Incrementing/Decrementing Pointers

Adding or subtracting an integer *n* to an integer pointer increments (or decrements) the value of the pointer by *n* bytes. Adding or subtracting an integer *n* to a CRAY pointer increments (or decrements) the value of the pointer by *n* words. To maintain compatibility when porting, multiply the value of *n* (CRAY words) by the size of the RS/6000 machine word size (8 bytes). Figure 5 shows the differences between a pointer incrementing using XL Fortran and CRAY Fortran.

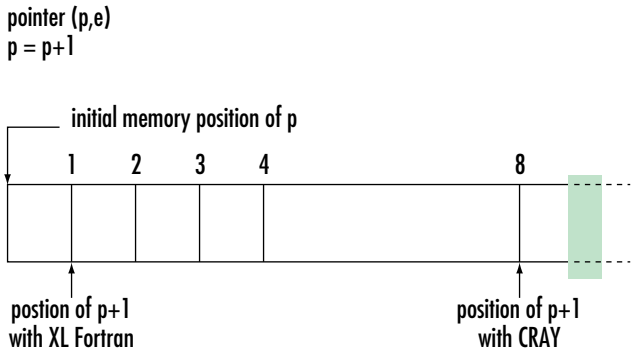


Figure 5. Pointer arithmetic: XL Fortran and CRAY Fortran

To port the code segment in Figure 5 to XL Fortran, modify the assignment statement as follows:

```
integer, parameter :: word_size = 8
p = p + (word_size*1)
```

LOC Intrinsic

The LOC intrinsic function returns the address of a pointee. This function is useful for finding the address of a variable and assigning it to the integer pointer of another pointee (overlaying the storage of the two variables).

Figures 6 and 7 show the modifications required when porting code from CRAY to XL Fortran.

Fortran 90 alternative to Integer Pointers: To dynamically dimension arrays in Fortran 90, use deferred-shape array specifications, and then explicitly allocate memory for the arrays as shown below:

```
integer, dimension(:), allocatable ::
   array
I = 5
allocate(array(5))
array = (/1,2,3,4,5/)
deallocate(array)
end
```

Service and Utility Procedures

XL Fortran supports several CRAY procedures as industry extensions. Some of these are described below. These procedures (except for CLOCK_) have the same names as the corresponding CRAY routines.

clock_: The clock_ function returns the time in hh:mm:ss format. (The XL Fortran routine is CLOCK_, not clock, because the libc library of AIX already contains a clock routine).

date: The date function returns the system date in mm/dd/yy format.

irtc: The irtc function returns an integer(8) value of the number of nanoseconds elapsed since the initial value of the machine's real-time clock, as follows:

```
integer(8) :: a,b,elapsed,irtc,x=3
a = irtc()
do m = 1,20000
  x = x**2
end do
b = irtc()
elapsed = b-a      ! elapsed = 33052928
```

rtc: The rtc function returns a real(8) value of the number of seconds elapsed since the initial value of the machine's real-time clock. Figure 8 shows an example.

```
integer big_array(100)
integer element, dim_size
integer array1(5), array2(5), array3(10)
pointer (p1,array1)
pointer (p2,array2)
pointer (p3,array3)
! Assign storage to array1
p1 = loc(big_array)
! Initialize array1
array1 = (/1,3,5,7,9/)
! Assign storage to array2
p2 = p1 + 5
! Initialize array2
array2 = (/2,4,6,8,10/)
! array3 holds data of array1 and array2
p3 = loc(big_array)
if ((array3(3) .ne. 5) .or. (array3(8) .ne. 6)) stop 1
! Increment the area of storage where array1 begins by 2 words
p1 = p1 + 2
element = array1(5)
if (element .ne. 4) stop 2
end
```

Figure 6. Original CRAY code

timef: The timef function returns the elapsed time in milliseconds since the first instance timef was called. The first instance timef is called and the value 0.0d0 is returned. Figure 9 shows an example.

jdate: The jdate function returns the system Julian date in yyddd format.

Fortran 90 alternatives to CRAY Fortran functions: You can use the DATE_AND_TIME intrinsic function to access date and time information, and the SYSTEM_CLOCK intrinsic function to inquire on your system clock.

CVMGx Procedures

XL Fortran supports conditional vector merge functions similar to the CRAY. These functions take three arguments. The values returned depend on the result of the test that is done on the third argument. The first argument is returned if the test done on the third argument is true. The second argument is returned if the test done on the third argument is false. When the arguments are arrays, testing is done on an element-by-element basis.

The conditional vector merge functions are as follows:

CVMGP	Test for positive values or zero
CVMGM	Test for negative values
CVMGZ	Test for zero values
CVMGN	Test for nonzero values
CVMGT	Test for logical true values

```
      @process intsize(8)               !<—- Added code
      integer big_array(100)
      integer element, dim_size
      integer array1(5), array2(5), array3(10)
      pointer (p1,array1)
      pointer (p2,array2)
      pointer (p3,array3)
      integer word                      !<—- Added code
      parameter (word = 8)              !<—- Added code
      ! Assign storage to array1
      p1 = loc(big_array)
      ! Initialize array1
      array1 = (/1,3,5,7,9/)
      ! Assign storage to array2
      p2 = p1 + 5*word                  !<—- Modified code
      ! Initialize array2
      array2 = (/2,4,6,8,10/)
      ! array3 holds data of array1 and array2
      p3 = loc(big_array)
      if ((array3(3) .ne. 5) .or. (array3(8) .ne. 6)) stop 1
      ! Increment the area of storage where array1 begins by 2 words
      p1 = p1 + 2*word                  !<—- Modified code
      element = array1(5)
      if (element .ne. 4) stop 2
      end
```

Figure 7. CRAY code (Figure 6) ported to the RS/6000

```
real(8) :: a,b,elapsed,rtc,x=2.0
a = rtc()
do m = 1,20000
  x = x**2
end do
b = rtc()
elapsed = b-a   ! loop required 0.8463557684564229 78e-02
```

Figure 8. Using the rtc function

```
real(8) elapsed, timef
elapsed = timef()     ! elapsed = 0.0d0
do m = 1,20000
  a = a**2
end do
elapsed = timef()     ! elapsed = 15.2616500997857362
```

Figure 9. Using the timef function

```
! CRAY
r = cvmgp(1.0,2.0,3.0)    ! r = 32-bit representation of 1.0
i = cvmgp(1.0,2.0,3.0)    ! i = 32-bit representation of 1.0
                          ! i = 1065353216
! XL Fortran
r = cvmgp(1.0,2.0,3.0)    ! r = 32-bit representation of 1.0
i = cvmgp(1.0,2.0,3.0)    ! i = int(1.0)
                          ! i = 1
```

Figure 10. CVMGx procedure

Be cautious when you are porting code to XL Fortran. Although the CVMGx routines accept integer and real arguments, these arguments and the function return type are interpreted as Boolean data types on a CRAY. Since XL Fortran does not have a Boolean data type, the argument and function return types are treated as integer, logical, or real intrinsic data types (as shown in Figure 10). The only place the user would see a difference between the XL Fortran and CRAY behavior is if the function is referenced in an assignment or expression where type conversion is expected to occur.

Fortran 90 alternative to CRAY CVMGx intrinsic functions: Use the Fortran 90 `MERGE` intrinsic function instead of the CVMGx intrinsic functions. Figure 11 shows the same function implemented using the XL Fortran extensions and Fortran 90 intrinsic functions.

Summary

XL Fortran Version 3 supports a variety of features to facilitate your porting needs, including several industry extensions found in CRAY products. Also, because XL Fortran fully supports Fortran 90, you can modify your code for portability to any platform that supports this latest Fortran standard.

For more information about porting, contact AIX Support Family at callaix@vnet.ibm.com or 1-800-CALL-AIX (U.S. and Canada only).

Emad ElHamahmy, IBM Canada Ltd., 844 Don Mills Road, North York, Ontario, Canada, M3C 1V7. Internet: emad@vnet.ibm.com. Mr. ElHamahmy is a development analyst currently working on the XL Fortran compiler. He has co-authored a workshop on Fortran 90 using the XL Fortran compiler. Mr. ElHamahmy received a BSc in Computer Science and Mathematics from the University of Toronto.

Luc Chamberland, IBM Canada Ltd., 844 Don Mills Road, North York, Ontario, Canada, M3C 1V7. Mr. Chamberland is an information developer on the XL Fortran team, focusing primarily on Fortran language documentation. He recently co-authored and taught a workshop on Fortran 90. Mr. Chamberland received both his BA in English and MA in Religious Studies from the University of Toronto.

Code Using CVMGx Routines

```
integer i
integer i_a(3)
integer array_t(3), array_f(3), mask(3)
i = cvmgp(1.0,2.0,3.0)        ! i = 1
array_t(1) = 1
array_t(2) = 2
array_t(3) = 3
array_f(1) = 4
array_f(2) = 5
array_f(3) = 6
mask(1)=7
mask(2)=0
mask(3)=8
i_a = cvmgz(array_t,array_f,mask)   ! i_a(1)=4, i_a(2)=2,
                                    ! i_a(3)=6
end
```

Fortran 90 Alternative

```
integer :: i
integer, dimension(3) :: i_a
integer, dimension(3) :: array_t, array_f, mask
i = merge(1.0,2.0,3.0 >= 0.0)           ! i = 1
array_t = (/1,2,3/)
array_f = (/4,5,6/)
mask = (/7,0,8/)
i_a = merge(array_t,array_f,mask == 0)  ! i_a = (/4,2,6/)
end
```

Figure 11. CVMGx functionality in Fortran 90