AIX Operating System SMP Performance

By William Alexander, Robert Dimpsey, and Bret R. Olszewski

This article provides an introduction to Symmetric Multiprocessor (SMP) performance concepts. It also discusses using measurements on uniprocessors to successfully tune the SMP version of AIX. The result is an exceptionally well-performing SMP operating system.

Performance was a major emphasis in AIX Version 4.1. Analysts and developers worked together to improve Multiprocessor (MP) performance using measurements and analysis of AIX 4.1 on uniprocessors before the MP hardware was available. This strategy allowed work to begin on the MP hardware with an operating system that already performed reasonably well, and to concentrate on the more complex performance issues that arise from dynamic MP system behavior. The result of this effort is that AIX 4.1 performs exceptionally well for a first-release SMP operating system. IBM has tools and techniques in place to further improve performance as we gain experience from customer workloads.

SMP Performance and Scalability

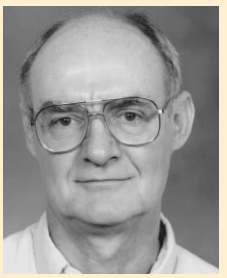
In an SMP system, multiple user tasks can run concurrently, and multiple copies of operating system routines can run concurrently on their behalf. However, there is only one copy of most system data structures: one run queue, one process table, one file table, and so on. If two system routines try to access the same system data simultaneously, the operating system must ensure that they access the data one at a time (serially). A lock is the primary way to serialize access to shared data.

Locks

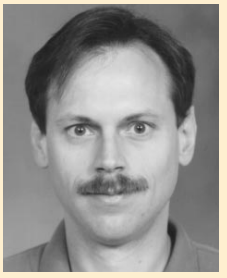
A *lock* is a bit associated with each piece of shared data. Before a routine accesses shared data, it must check the associated lock. If the bit is off, no other routine is currently accessing the data. The routine then sets the bit and accesses the data (testing the bit and setting it must be atomic). If a routine wanting to access the data finds the bit on, it must wait until the bit is turned off by the routine currently accessing the data.

Even with locks, it is difficult to ensure correct operation in an SMP. Hewlett-Packard® has stated that the most difficult technical problem moving to an MP system was adding protection to the data structures to allow multiple processes to execute[1]. This process is so complex that it is often started by using a single lock for the entire kernel. This implementation was used in SunOS Versions 4.1.2 and 4.1.3; however, this solution can result in unsatisfactory performance. It requires some kernel routines to wait unnecessarily when two routines access entirely disjoint sets of data, and could therefore safely run concurrently.

Although locks are necessary, they cause many performance problems in SMP systems, either directly or indirectly. First, acquiring and releasing locks adds path length to nearly all system routines. To minimize the negative effect of this extra lock-unlock path length on the performance of uniprocessor systems, IBM provides two versions of the AIX 4.1 kernel—one for uniprocessors and the other for SMPs. The uniprocessor version of the kernel contains fewer locks than the SMP version (some locks are still required in the uniprocessor kernel because AIX is preemptable).



William Alexander



Bret R. Olszewski

[1]Larson, Douglas V. and Polychronis, Kyle A. "A Multiprocessor HP/UX Operating System for HP 9000 Computers." *Hewlett-Packard Journal.* December 1992.

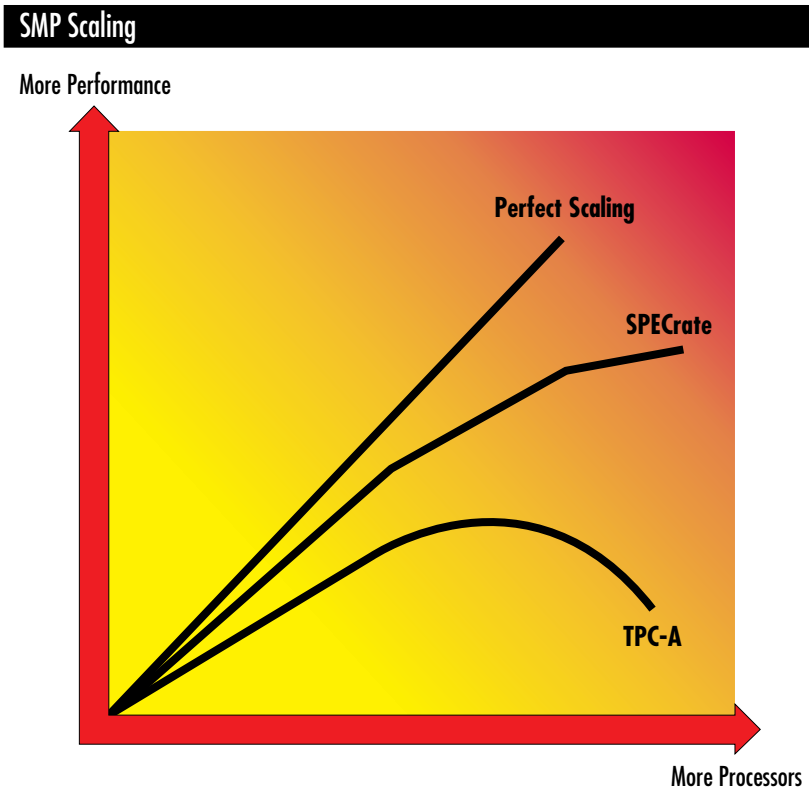


Figure 1. SMP scaling

Performance problems can occur with locks that protect heavily used data structures. The memory locations containing such locks can become very "hot," causing undesirable cache effects. This and other hardware considerations are discussed briefly in the next section.

The most serious performance problem caused by locks is the processor power wasted by routines waiting for locks held by another thread. If the lock will be released soon, the most efficient action is to *spin*, repeatedly testing the lock bit until it becomes free. If the lock will not be released soon, the waiting thread should be suspended and put in a queue waiting for that lock. Another thread must be dispatched, and the waiting thread must later be awakened and redispatched. Either way, these cycles are wasted from the user's perspective. Because waiting time grows non-linearly as the number of competing tasks increases, waiting has a very adverse effect on scalability.

Scalability

Scalability is the effective speedup of the system as processors are added. Typically, we calculate scalability as the effective throughput of a given workload running on multiple processors divided by the throughput of that same workload running on a single processor (the base). Scalability is often expressed as a fraction; for example, 2.5/4 means that a particular four-processor SMP achieves two and a half times the performance of a single processor in the same system on a given workload.

Measuring scalability can be straightforward because most SMPs allow the dynamic disabling of processors from software. We can turn off three processors on a 4-way SMP and measure throughput with a workload running on one processor, then measure it again with the workload running on all four processors.

Customers who upgrade to an SMP generally compare the performance of a 4-way SMP running AIX 4.1 with that of a uniprocessor running AIX 3.2 rather than AIX 4.1. The performance of the SMP running AIX 4.1 will be less than the uniprocessor running 3.2 because of the extra lock-unlock path length. Much of this extra path length penalty is masked because there are many performance enhancements in AIX 4.1 unrelated to MP.

For real-life workloads on SMPs, each additional processor adds less throughput than the previous one until a point is reached where adding processors decreases throughput. As shown in Figure 1, scalability is highly workload-dependent, so the point of diminishing returns will be different for different workloads and benchmarks. For example, a benchmark such as SPECrate™—which suffers few cache misses, makes few operating system calls, and has little shared user data—will scale linearly on most SMPs. At the same time, a system-intensive workload may gain little benefit from more than a certain number of processors.

Although many impediments to perfect scaling in SMP systems exist, the following four are the most important:

- Contention for memory subsystem hardware, especially the bus and data switch
- Increased overhead of cache misses because of coherency protocol
- Communication and contention for shared data within applications
- Contention for shared data within the operating system

Each processor in IBM's SMPs has its own Level 1 (L1) and Level 2 (L2) caches. There may

be copies of a given memory location in the caches of any of the processors. If there are copies in more than one cache, the copies must be kept consistent. If one processor executes a load instruction for a particular location, and another processor has a copy of that location in its cache, the hardware ensures that the load is satisfied from the other processor's cache rather than from memory. This guarantees that the load instruction has the freshest value.

A *lateral read* is reading from another processor's cache, not from memory. If two processors have a copy of the same location in their caches and one processor modifies the value, the hardware invalidates the copy in the second processor's cache; this is called a *cache cross-invalidate*. Although lateral reads and cache cross-invalidates are necessary to keep the caches consistent, this cache-to-cache traffic increases the load on the system bus as shown in Figure 2.

With multiple processors, contention for hardware elements of the memory subsystem is a scaling impediment inherent in SMPs. Caches successfully reduce memory accesses, but the cache coherency protocol adds bus traffic of its own. Together, contention for memory access and cache coherency protocol overhead cause the average Cycles Per Instruction (CPI) for a given workload to grow as more processors are added to a system. Naturally, if it takes more cycles to execute the same number of instructions, performance suffers.

These hardware effects are not directly visible to software; a program cannot directly detect when a load from memory had to wait for access to the bus, or when a store caused a cache cross-invalidate. But this does not mean that software developers should ignore these hardware-scaling impediments. Programs can reduce cross-invalidates by ensuring that two "hot" data items or locks are not in the same cache line.

Because a cache miss is more costly on a multiprocessor than on a uniprocessor, the principles of distributed memory apply at the cache level for shared memory machines: keep the data close to the processor. The workload scalability is closely related to the percentage of memory references that can be satisfied in the nearest memory (L1 and L2 caches for an SMP). Hewlett-Packard has stated that the bulk of their tuning work for the 4-way HP™ 9000 827 was spent reducing cache effects[2].

Cache-to-Cache Bus Traffic

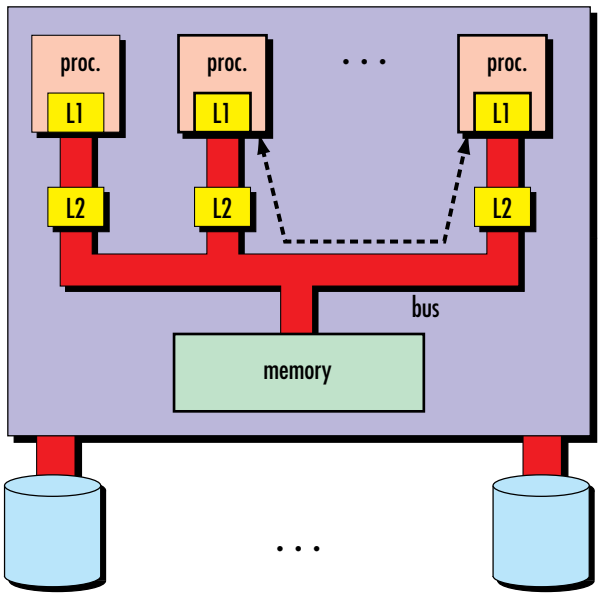


Figure 2. Cache-to-cache bus traffic

If a single application consists of multiple tasks, then scalability can be reduced by communication and synchronization among tasks. Contention for shared application data can also limit scalability. Both considerations apply in some database implementations.

The remainder of this article will focus on our efforts to confront the fourth impediment to scaling: contention for system data structures.

System Performance Goals

During the early development of AIX 4.1, we established two types of performance goals. The first applies to base performance on a single processor—a path length goal for system components. The second goal is related to scalability.

The base goal, which applies to the MP version of the kernel, states specifically that the performance of our benchmarks would not degrade more than 15% when run on the MP-enabled kernel compared to the AIX 4.1 Uniprocessor (UP) kernel. Enabling AIX to run on SMPs adds path length to many kernel components because of its support for threads and the extra lock-unlock code. Our approach was to apply standard path length analysis techniques to the new code while it ran on a uniprocessor. In addition, we made performance improvements to AIX that were unrelated to MP, which partially offset the path

[2] Ibid.

Component	TPC-A C/S	TPC-C C/S	LADDIS	SDET	Kenbus
Virtual Memory Manager (VMM)	High	High	High	High	High
Process Management (PROC)	High	High	Low	High	High
Journaled File System (JFS)	None	None	Low	High	Low
Logical File System (LFS)	Low	Low	Low	High	High
Network File System (NFS)	None	None	High	None	None
TCP/IP	Low	Low	High	None	None
Network Device Drivers	Low	Low	High	None	None
Loader (LDR)	None	None	None	Low	High
Disk Device Drivers	Low	Low	Low	Low	Low

Figure 3. System component usage

length increases that could not be avoided. Most of these improvements applied to both kernels. Our measurements show that the AIX 4.1 UP kernel performs better than AIX 3.2.5 on most internal performance regression tests, and the MP kernel running on a uniprocessor meets the 15% goal on all industry-standard benchmarks.

AIX 4.1 UP kernel performs better than AIX 3.2.5 on most internal performance regression tests.

Since the second goal concerned operating system scalability, we focused on locks. Lock contention is the principal software culprit limiting scalability. As more processors are added, more system routines contend for the same locks, the locks are held longer, and waiting time rises.

Experience with other SMPs and queueing theory suggests that waiting time for a given lock becomes unacceptable if the lock is held more than 20% of the time. We therefore adopted the goal that no AIX lock be held more than 20% of the time under our test benchmarks. This implies that in a four-processor system, one processor cannot hold a particular lock more than 5% (20%/4) of the time. With a design goal to provide scalability to eight processors, the target hold time per processor was set to 2.5%. As AIX 4.1 was being developed, we measured the fraction of time each lock was held as we ran our benchmarks on a uniprocessor, and flagged any lock that was held more than 2.5% of the time. This enabled us to make substantial progress on scalability, far ahead of the availability of MP hardware. In fact, we could accurately predict the locking rates that we later measured on the SMP.

Since we began early and AIX developers were involved in the performance effort from the beginning, the AIX 4.1 SMP kernel meets the 5% goal for all locks on all of our benchmarks. In addition, most locks on benchmarks also clear the 2.5% mark. These facts, plus early scalability measurements, give us confidence that most workloads will scale well on our 4-way SMP. We will continue to achieve 8-way scalability.

Workload-Based Design

Because IBM's SMP is a general-purpose server, it must perform well in complex multi-user situations. A performance measurement process was designed around a few workloads that were used for performance evaluation of the AIX 4.1 MP software. The selected workloads—industry-standard benchmarks representing a cross section of the critical markets defined for the product—included the following:

TPC-A: At the time of initial SMP planning, this was the most frequently used database benchmark. All implementations studied were client/server.

TPC-C: This is a significant database benchmark. All implementations studied were client/server.

SPEC SFS (LADDIS): This benchmark tests NFS® server performance. Typically, it is run with multiple Ethernet™ or Fiber-optic Data Distribution Interface (FDDI) networks sending read, write, and lookup requests to the server.

SPEC SDM SDET: This workload tests multi-user performance by simulating several users executing typical UNIX® commands.

SPEC SDM Kenbus: This workload tests multi-user software development performance. It is similar to SDET, but simulates thousands of users and must run on large memory systems.

All these benchmarks involve large amounts of disk I/O. A typical four-processor MP system requires 20 to 40 disks to sustain peak throughput on these workloads. The client/server TPC and LADDIS benchmarks also require network interaction with the server systems.

In addition to these real-world workloads, we have developed stress tests that focus on the disk, communication, and Streams framework subsystems.

Each workload was analyzed on AIX 3.2, categorizing path length use in components and subsystems. This analysis was used to plan the software development effort needed for parallelization. Figure 3 shows a summary of the findings.

The operating system components exercised during the analysis vary with the workload. No one workload can characterize the real world well enough to guarantee good general-purpose performance.

Lower level analysis showed that use within components varies considerably. For example, although the Virtual Memory Manager (VMM) is used extensively in all workloads, different VMM functions are exercised by various workloads. For the TPC workloads, VMM is used almost exclusively for copy operations between user space and the kernel for system calls. For LADDIS, VMM exercises page operations related to I/O. For Kenbus and SDET, VMM exercises operations associated with process creation and deletion.

By understanding which components were not critical, we could avoid investing too much in making them parallel. We targeted the components that were critical to achieving excellent system-level performance.

Lock Implementation

Locks are requested frequently in commercial workloads, so the implementation lock and unlock code is crucial for good performance. Early measurements of locking rates are shown in Figure 4.

Because locks are so central to SMP performance, considerable effort was invested in their implementation. Principal goals in the lock primitives design included different locking options, low path length, and sophisticated instrumentation. Two basic lock types, both based on the OSF/1® model, are used: simple locks and complex locks.

Simple locks are mutually exclusive locks; only one entity may be holding the lock at any time. Simple locks in AIX 4.1 can behave as spin locks. When a process cannot obtain a lock because it is held by another process, the lock code spins in a tight loop, continuously testing the lock bit until it can gain the lock or until a threshold of spinning is reached (`MAX_SPIN`). Threads that reach the threshold of spinning are put to sleep and awakened when the lock is free. Interrupt handlers, which are not allowed to sleep, must spin until the lock becomes free. Another useful feature of simple locks is that if the thread holding the lock is not currently dispatched, the requester is put to sleep immediately. That is because it is unlikely that the required unlock will occur soon.

Complex locks are mutually exclusive when used for operations that modify data (WRITE access), but allow multiple entities to hold the lock simultaneously for viewing data (READ access). Otherwise their implementation is similar to simple locks.

	Lock Rate/sec Quad
SDET	102,000/sec for 4 processors at 75 MHz
TPC-C	57,000/sec for 4 processors at 75 MHz
LADDIS	56,000/sec for 4 processors at 75 MHz

Figure 4. Locking rates for workloads

Lock instrumentation is critical for understanding the dynamic behavior of the system. There are two sets of instrumentation: counters and trace hooks. Counters allow gross level analysis of lock and lock miss rates in the system and can be viewed through the `lockstat` command. Counters indicate which locks are hottest. Trace hook instrumentation allows sophisticated analysis of system behavior, such as identifying lock hierarchy problems.

Lock instrumentation is crucial for SMP tuning, but it has a noticeable (2% to 3%) impact on performance, even when the tracing facility is not activated. For this reason, AIX can be run with lock instrumentation enabled or not; this decision is made when the kernel is booted.

Principal goals in the lock primitives design included different locking options, low path length, and sophisticated instrumentation.

Tuning AIX During Development

This section describes performance improvements that were made running AIX 4.1 on UP platforms, before MP hardware was available.

To help compensate for the unavoidable path length expansion required to support threads and implement locking, a significant tuning effort was undertaken. This resulted in the performance of the AIX 4.1 UP kernel being better than the AIX 3.2.5 kernel, in spite of supporting threads and some restructuring related to MP enablement. The improvements to the UP kernel also typically improved MP performance.

One large class of problems is associated with copying data between user and kernel space. The primary difficulty is that thread support requires locking the process address space, which is not necessary in previous versions of AIX. For single-threaded processes, significant design changes mean that copies run much faster in AIX 4.1 than in any previous version of AIX.

One pleasant surprise was the performance benefit of compiling the operating system using the data-in-toc compiler option. This option allows the compiler to move some constants into the Table of Contents (TOC) area—removing one instruction of overhead for each access. An

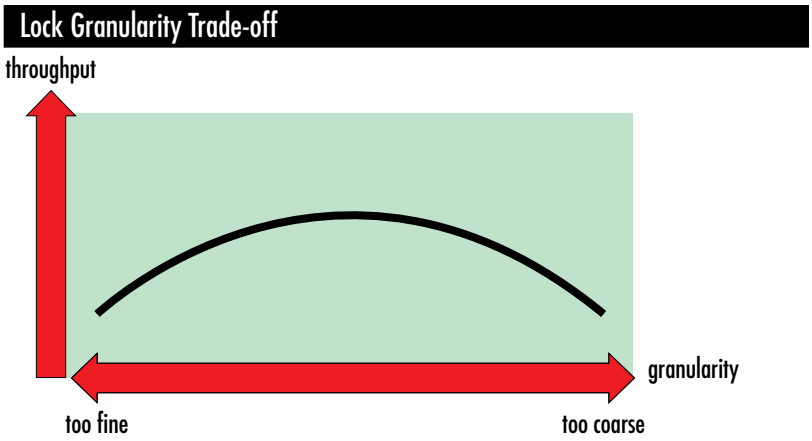


Figure 5. Lock granularity trade-off

average of 3% improvement in system performance was observed merely by rebuilding the operating system with the data-in-toc compiler option.

The path length for the lock and unlock operations is one key to good performance. The path length associated with locking was initially higher than expected. Three optimization passes were performed on the locking code to reach the current implementation. The lock and unlock primitives are processor type-specific, so performance work centered on the implementation for the PowerPC. PowerPC systems have four special registers available to software. The pointers to the current thread structure, the current save area (`csa`), and the per-processor data area (`ppda`) are kept in these special registers. This allows these data structures to be accessed in one instruction. Since the thread identifier is used in all lock operations, a readily available pointer to the current thread improves lock performance on the PowerPC.

When measurements show that a lock is held in excess of the 2.5% target, one common technique is to reduce its granularity. For example, a lock protecting a table might be replaced with an array of locks, each protecting one entry in the table. This technique was used in the `pmap` layer of the VMM.

Sometimes during AIX 4.1 development, measurements showed that path length was unacceptably degraded because of too many locks. The solution was to coalesce them. For example, the initial granularity of the communication subsystem's memory allocator turned out to be too fine, and locks were coalesced to protect larger structures. This is a classic SMP tuning trade-off: decreasing the number of locks decreases path length in both UP and MP versions of the kernel, but may increase lock holding and waiting times in MP systems. Finding the right balance is more art than science. Figure 5 illustrates the trade-off between granularity and throughput.

Another example of increased UP performance obtained by removing a lock occurred in the logical filesystem. Here, the user file descriptor (`ufd`) per-process structure must be modified when a new file is opened. Initially, a lock was taken to support threads, since two or more threads could try to open a file simultaneously. An easy optimization was to check if there was only one thread in the process, in which case locking was not needed.

Another common optimization was converting code that modified single data fields under lock to use atomic operations such as `fetch_and_add`. Significant processor cycle savings can occur using atomic operations.

Some non-critical statistics counters were converted to being "fuzzy"; that is, counters are not 100% accurate, but memory inconsistency may cause a count to be lost from time to time. Some statistics, such as the number of free pages available to the VMM, cannot be fuzzy; but many can be with no noticeable effect to users. Several counters in TCP/IP were made fuzzy.

Some large system effects were observed, even when running on UP systems. For example, during development a convoy problem was noted with the Kenbus workload. Frequently, a process holding a lock was preempted at the end of its time slice. The next process to attempt to gain the lock would miss it. The lock miss code identifies the process holding the lock and tries to ensure that it will run soon because it is causing lock contention. A priority boosting mechanism now bumps the priority of the process holding the lock so that it will run again soon.

The large system effect was related to the many processes in Kenbus; the bumped process queued at a more favorable priority, but the list of ready processes on the priority queue was so long that subsequent lock misses occurred on the same lock. The solution implemented in AIX 4.1 was to bump the process and queue it at the front of the priority queue. This improved performance, and perhaps as importantly, reduced variance in system response time.

MP System Dynamics During Development

The most serious limitation of tuning SMP software on a UP is that the dynamic behavior of any code will be different on an SMP than it is on a UP. Therefore, it is difficult to predict how any tuning change will affect SMP behavior. This section provides a few examples of the kind of tuning work that could be accomplished on SMP hardware.

After a "hot" (frequently accessed) lock is broken into multiple locks to reduce waiting time, affected kernel routines run faster. The result is that the rate at which the kernel requests certain other locks is now increased. This increase in access rate will not be uniform across all locks, and it is not always possible to determine which locks will now be hottest.

The two major "hot" locks identified through UP analysis were the Journaled File System (JFS) lock and the VMM lock. Both subsystems were initially constructed with just one lock protecting all of their respective data structures. This MP Safe version of the operating system was coded so runnable software would be available when functioning hardware was first ready. Developing and implementing a more granular set of locks to protect these data structures was done in parallel with designing the first software and hardware.

Our initial MP measurements highlighted the VMM and JFS locks, as expected from the UP measurements and the lock protecting the process table. Installing the more efficient VMM and JFS locking design dramatically improved performance. For example, SDET performance improved by over 70%.

Fixing the first tier of hot locks also revealed the next tier, which we had not been able to predict from our UP measurements. This set included two new locks introduced with the more efficient versions of JFS and VMM: the `inode` cache lock and the `pmap` lock.

The JFS `inode` cache lock protects the `inode` pool. Contention for this lock was reduced by improving the hashing algorithm used to locate the `inode` in the pool. This reduced the path length under lock; no additional locks were introduced. The `pmap` lock was split into multiple locks.

One interesting problem that UP-based analysis could not reveal concerned an interaction between sequential disk I/O and interrupts. AIX uses a write-behind algorithm to greatly increase sequential write throughput. Because SMP interrupts can generally be fielded by any processor, the system can sometimes send smaller than optimal write requests to the disks during sequential I/O. We developed a coalescing algorithm to correct this problem.

Another non-intuitive observation came with analysis of the optimum value for `MAX_SPIN` (number of times to spin on a lock before sleeping). We expected this value to be important to system performance, but experiments showed that most workloads were not sensitive to it. This is because our design reduces the number of lock misses. Of the misses that do occur, some fraction are in interrupt handlers that are not allowed to sleep, and thus do not use `MAX_SPIN`.

Two internal trace postprocessing tools were indispensable in the tuning effort. One displays events from the AIX trace log in a graphical manner, making abnormal or undesirable system behavior easier to spot. The other tool summarizes system call times, lock holding times, and lock miss rates; this tool highlighted the remaining hottest locks after each successive wave of tuning.

The dynamic behavior of any code will be different on an SMP than it is on a uniprocessor.

Processor Affinity Scheduling

Our processor affinity scheduling policy could not be addressed solely from UP data. On SMP systems, several dispatching disciplines can be implemented. The most obvious allows any thread to be dispatched on any processor, enabling the system to keep all processors busy. This policy usually results in the best system throughput.

A policy that results in fewer cache misses binds each thread to a particular processor and allows it to execute only on that processor. For example, if a system has a large second-level cache for each processor, dispatching a thread on the processor on which it last ran benefits from reusing the lines of the cache still containing context for that thread. This policy, called *strict affinity*, can result in better response times for high-priority tasks, but generally causes lower system throughput because one or more processors may become idle while there are threads ready to dispatch.

The AIX dispatching strategy implements *opportunistic affinity*. When dispatching, the list of threads available at the highest priority is searched. If one of these threads was the last to run on the processor doing the dispatch, it is selected; otherwise, the first thread available is

dispatched. In this way, most benefits of affinity are preserved, but system throughput is also maintained.

Applications may use the `bindprocessor()` system call to explicitly bind threads to processors, if desired.

References

- Campbell, Mark, et al. "The Parallelization of UNIX System V Release 4.0." Presented at USENIX, Winter 1991. Dallas, Texas.
- Campbell, Mark; Holt, Russ; and Slice, John. "Lock Granularity Tuning Mechanisms in SVR4/MP." *Distributed & Multiprocessor Systems* (SEDMS II), USENIX Association.
- Kleinman, Steven, et al. "Symmetric Multiprocessing in Solaris 2.0. USENIX, Summer 1992. San Antonio, Texas.
- Eykholt, J.R., et al. "Beyond Multiprocessing: Multithreading the SunOS Kernel." Presented at USENIX, Summer 1992. San Antonio, Texas.

William Alexander, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Dr. Alexander is a senior programmer. Since joining IBM in 1991, he has worked on AIX and MP performance. He has a BA in Philosophy from Rice University and a PhD in Computer Science from the University of Texas at Austin.

Robert Dimpsey, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Internet: dimpsey@ austin.ibm.com. Mr. Dimpsey is an advisory programmer. He has worked in the AIX Performance group since joining IBM in 1992.

Bret R. Olszewski, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Olszewski is an advisory programmer working on MP performance. He joined IBM in 1989 and has worked on various aspects of AIX performance. He has a BS in Computer Science from the University of Minnesota.