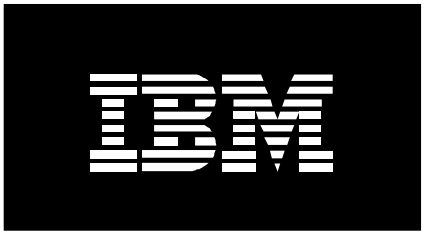


IBM @server pSeries 650 Performance and Tuning

Bret Olszewski
breto@us.ibm.com

December 4, 2002

Introduction

On November 12 2002, IBM announced the IBM @server™ pSeries™ 650 server. This is the first POWER4+™ microprocessor in a UNIX ® mid-range system. The p650 gives customers unprecedented performance, granularity, and reliability in a package which should appeal to the most demanding customers. This document discusses aspects of software tuning associated with the p650 system.

p650 Structure

The p650 system structure is inspired by its award-winning[1] sibling, the pSeries 690 server. The p650 is the first pSeries system to utilize the POWER4+ chip. This chip is an enhanced version of the POWER4™ microprocessor that powers the p690. The POWER4+ chip is fabricated with IBM's latest 0.13 micron technology. This allows the chip to be faster, smaller and to consume less power, even with an increase in the amount of on-chip cache. Since a POWER4+ chip contains two independent microprocessors, the processor granularity, or increment of processing power, is by chip. Thus, a p650 system may have two, four, six, or eight POWER4+ microprocessors. Each POWER4+ chip is packaged on a single chip module (SCM), as part of the processor "book". Also incorporated onto this processor book is a Level 3 (L3) cache and eight memory DIMM slots.

Memory must be populated in groups of four identical DIMMs. Each processor book can contain two four DIMM groups of differing sizes. All processor books must have the same DIMM configuration. Sustainable bandwidth is not significantly affected by the size of the memory DIMMs.

The p650 is packaged to allow flexibility in both processing density and I/O expandibility. Each p650 occupies an eight EIA unit (8U) drawer, allowing up to five p650 systems in an 19-inch T42 IBM rack. It is possible to add up to eight I/O expansion drawers, giving a maximum of 55 I/O slots. IBM's hardware management console (HMC) support is optional, but is required to support logical partitioning (LPAR) and clustering (planned for 1H03). LPAR allows running multiple operating system instances simultaneously on a single system. The p650 LPAR support, when used with AIX 5L™ v5.2, allows dynamic reconfiguration of processors, memory, and I/O between operating system instances without the need to reboot the partition.

Figure 1 shows an overview of the p650 packaging and features.

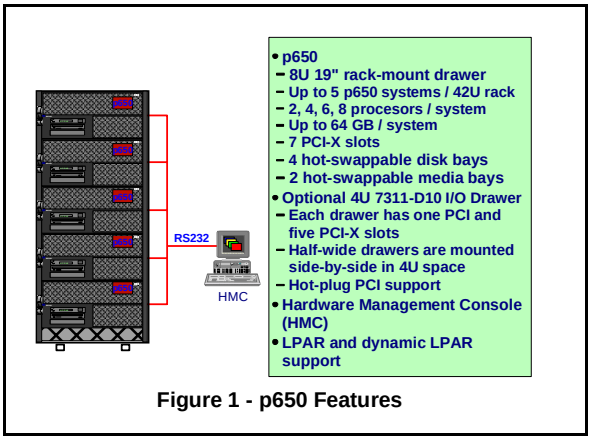


Figure 1 - p650 Features

p650 Specific System Tuning

The p650 system generally functions with excellent performance for all applications. System specific tuning is not required for either 32-bit or 64-bit applications. The AIX 5L v5.2 operating system optimizes a number of common library operations (e.g. bzero, memset, memmove) using architecture-specific instruction sequences. These optimizations maximize performance for POWER4 systems and are transparent to applications. The AIX 5L operating system provides two kernel alternatives, the 32-bit kernel and the 64-bit kernel. Both AIX® kernels support the 64GB maximum physical memory configurable on the p650. However, kernel intensive workloads usually show slightly better performance using the 64-bit kernel. In general the performance differences between the kernels on p650 typically measure at most a few percent. The p650 may operate with logical partitioning (LPAR mode) or without (symmetric multiprocessing mode). The overhead of running under LPAR is normally proportional to the amount of time in the operating system. This is a result of the fact that the hypervisor, or software which provides isolation between operating system instances, runs function on behalf of the operating system. For kernel intensive workloads, LPAR generally results in overhead on the order of two to five percent.

Computationally intensive applications can benefit from POWER4 specific optimizations. The IBM compilers, including C for AIX Version 6.0, Visual Age C++ Professional for AIX Version 6.0 and XL Fortran for AIX Version 8.1, support POWER4 code optimizations. The subject of POWER4 coding and optimization is covered in the IBM Redbook "The POWER4 Processor Introduction and Tuning Guide", ISBN0738423556.

As with the p690, the p650 provides a hardware option which can be used to optimize performance of applications.

p650 Performance Firmware Option

The p650 provides one level of hardware specific tuning to the end customer. This is applicable only on the 1.45 GHz version of the product. The tuning mode can be selected via the service processor menu, as shown in Figure 2.

```
                L3 Mode Menu

     Default L3 Mode :  Currently Shared (/Private)

     1. Private/Shared L3 Mode:
           Currently Shared (/Private)

    98. Return to Previous Menu
0>
```

Figure 2 - Firmware Menu

To understand the effects of this mode option, it is necessary to describe the cache structure of the p650. As described earlier, each processor book contains a POWER4+ chip, memory, and an L3 cache. An eight-way system is constructed using four processor books, each with a POWER4+ chip, L3 cache and memory. On the 1.45 GHz model, each L3 cache is 32MB in size, giving a total of 128MB of L3 cache for an eight-way system. The 1.45 GHz model is identified by the processor card feature code 5208 or 8051. On the 1.2 GHz model, each L3 cache is 8MB in size, giving a total of 32MB of L3 cache on an eight-way system. The 1.2 GHz model is identified by the processor card feature code 5122 or 8050. The memory and L3 which is on the processor book is referred to as "local" when accessed by a POWER4+ processor on the book. When memory or L3 is accessed by a processor from another processor book, it is referred to as a "remote" access. While the latency differences between local and remote accesses are fairly small, on the order of 16 percent, they are measurable in some application environments.

The processor books are connected in a ring topology. The address and data paths that connect the processor books are separate and are generically referred to as the fabric bus. Addresses and data for bus transactions that involve moving data between books in the system traverse the ring. Within each book, multiple paths exist for address and data traffic. Figure 3 shows the internal topology for the processor book. The figure shows address bus paths to the POWER4+ microprocessor's L3 cache controller (L3C), the on-chip shared Level 2 (L2) cache, the GX I/O controller logic, and two non-cacheable units (NCU). Data paths exist connecting the L3C, the L2 and the GX bus along with additional internal data paths. If either of the two processors on the chip extract data from the local L3 cache or the local memory, the data travels internally within the processor book, avoiding a hop on the inter-book bus fabric.

The internal working of the microprocessor make it possible to efficiently source data, whether from the memory, L3 cache, or one of the internal microprocessor caches to the fabric bus. The wealth of data paths makes the system very scalable as contention to data paths is minimized.

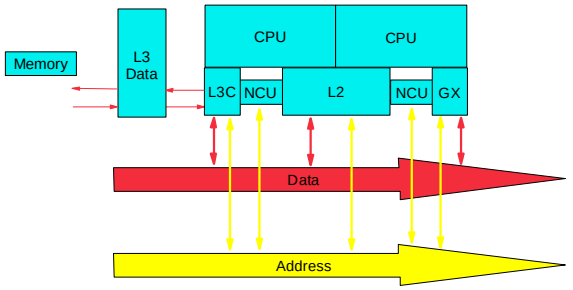


Figure 3 - p650 Book Dataflow

Two modes of L3 operation are permitted on the 1.45 GHz model, private and shared. In private mode, shared data for the same cache line may exist in other L3 caches simultaneously, but modified data for a cache line can only be resident in the L3 associated with its memory. So, in general, a processor has access to the amount of L3 cache local to it for unmodified data. The mapping of an address to memory location is on a page basis. All of the cache misses for a physical page will go to the same processor book's memory.

In shared L3 mode, there is always a one-to-one mapping from an address to a cache block where the data resides. This has three obvious benefits. First, for shared data there is never more than one copy occupying cache. In private mode, it is possible for a shared item, for example part of a program's executable file, to occupy space in all of the L3 caches at the same

time. Second, it gives a single program the ability to use all of the cache available on the system if needed. Third, the mapping of address to physical memory is striped on 512-byte boundaries. This gives the processor the ability to distribute the access for cache lines of a page across memory on all of the processor books in the system. Figure 4 shows each 4096 byte page is backed by memory on each processor book on an eight-way system. The first 512 bytes of the page map to one book, with each 512 bytes mapping to the next book, until after 2048 bytes the mapping returns to the first book.

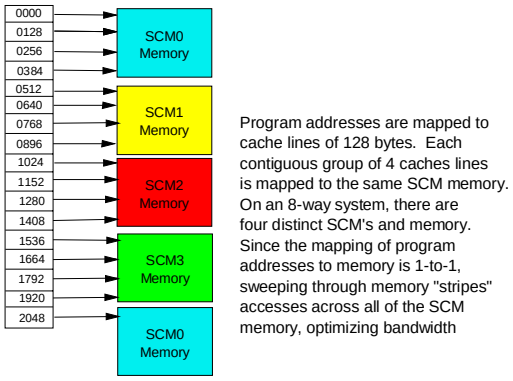


Figure 4 - Shared L3 Page Mapping

Shared L3 mode usually provides the best performance for applications. But, shared L3 mode has a performance drawback for some workloads. Due to the dataflow properties of the POWER4+ microprocessor, data which is sourced from memory to a remote book's processor is not always cached in L3. The impact of this is that workloads that share large amounts of data which is not modified will sometimes run more efficiently in private L3 mode. The effect has been seen to be largest in workloads with extremely large executable files. Since executable files are not usually modified, the instruction fetches may go all the way to memory when they could be cached in L3. In general, the range of performance difference between shared L3 mode and private L3 mode is less than ten percent, with most workloads observed at just a few percent.

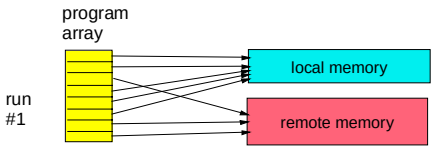
Shared L3 mode requires symmetric memory subsystem configuration. On a multi-book system, if some component of the memory subsystem fails on a card, such as a DIMM, the next system reboot will be in private L3 mode. Thus, some hardware failures which can be functionally tolerated will result in a change to system performance behavior. The six-way 1.45 GHz system always runs in private L3 mode. When benchmarking an 8-way system with two processors disabled, it is not equivalent to a true 6-way system. In this case, the L3 associated with the disabled processors continues to run, allowing additional L3 cache as well as the opportunity to run in shared mode. Additionally, memory attached to the processor book is allocated whether or not the processors on that book are disabled.

Run Time Variability

The elapsed or CPU time it takes to run a task can be impacted by a number of factors. While most customers understand that the run time for a given task can be affected by other tasks on the system as well as nuances of operating system resource management, there are cases where having consistent CPU consumption across runs of the same program is desirable. Since CPU consumption is accounted by a sampling technique in AIX, it is unreasonable to assume that run to run CPU consumption will be consistent for tasks that run only a short period of time.

For programs with long execution time, variability can be introduced a number of ways. For example, if two or more threads are running on the same POWER4+ chip, they may compete for shared resources. These shared resources include the L2 cache and the bandwidth from the chip to memory. Normally, the high level of associativity of the L2 cache will keep variability to a minimum. But if a thread assumes it has the entire L2 cache to itself, say to "block" an array, the cache will not be exclusively available if a thread on the other processor on the chip is also running. The bandwidth to memory is infrequently an issue with POWER4+, but it needs to be considered.

Run time variability can be encountered on the p650 when the L3 caches are run in private mode. The variability is a consequence of the latency and bandwidth to local memory being different to remote memory. Since the allocation of pages to a program is the responsibility of the operating system, the program's ratio of "local" pages to "remote" pages can vary from run to run. Normally this isn't an issue, since program working sets tend to fit within the L2 and L3 caches local to the processor. But, if the program is memory bandwidth or memory latency intensive, this can result in CPU time variability. Figure 5 shows that a program, run twice with exactly the same data set could get two completely different mappings of a program array to local and remote memory. If the program is extremely bandwidth intensive, the second run with less local page allocation would take more CPU time to execute than the first run.



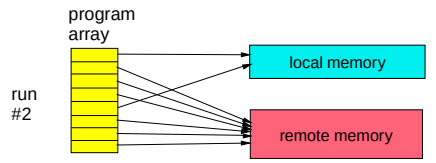


Figure 5 - Memory Allocation

Memory Affinity

With AIX 5L v5.1.0.25 and above, support for memory affinity is available on POWER4 systems with most levels of firmware. Memory affinity is a feature of the AIX 5L operating system which attempts to allocate memory for programs based on system topology. On the p650, memory affinity is irrelevant if running in shared L3 mode. This section briefly touch on AIX memory affinity support. This feature is documented in the AIX Performance Management Guide.

As of this writing, memory affinity is not supported under LPAR. To determine if an AIX 5L instance is running under LPAR or not, use the uname -L command. This command returns the partition number and name when running under LPAR. If not running under LPAR, the command returns "-1 NULL". Memory affinity requires firmware support, which is included on all p650 systems.

With AIX 5L v5.1, memory affinity is enabled with the following command:

vmtune -y 1

With AIX 5L v5.2, centralization of performance tuning commands is accomplished with the vmo command. To enable memory affinity, enter the following command:

vmo -r -o memory_affinity=1

After running vmtune or vmo it is necessary to run the bosboot command. Once the bosboot command is executed, the system must be rebooted for memory affinity to take effect. Memory affinity allows AIX to manage the physical memory of the system in "pools". One or more pools may be used to represent the local memory for a group of POWER4+ chips. The memory configuration rules of the p650 require that each processor book have the same amount of physical memory attached. This results in AIX having equal sized memory pools. Since AIX loads some kernel data in fixed locations of memory, there is less memory available in the pool(s) associated with the first processor book when the L3 cache is configured in private mode.

Memory affinity usage is different between AIX 5L v5.1 and v5.2. In AIX 5L v5.1, the enablement of memory affinity causes each page instantiation to attempt to map the new page to memory local to where the thread is currently running. This can be undesirable for some kinds of applications. For example, if a multi-threaded application is coded such that a single thread does the initial touch of all memory and then creates threads for parallel execution, the memory for all the threads may be allocated on one processor book.

Under AIX 5L v5.2, flexibility is improved to provide a measure of user control over memory affinity. When memory affinity is enabled, the default behavior of AIX is to allocate pages across all of the pools in a round-robin fashion. For example, on an eight-way p650, the allocation of four pages faults in a row by a thread will result in each of the pages on different processor books' memory. The striping of memory is very desirable in general for obtaining repeatable processor time consumption for tasks. It has the effect of randomizing the allocation of memory throughout the system. Though striping will usually result in repeatable performance, it will not always result in optimal performance. For some applications, primarily simple single-theaded ones, it is best to allocate memory on the processor book associated with the processor. This may be accomplished with the use of the MEMORY_AFFINITY environment variable. To trigger local memory for threads started in a shell, set the environment variable as:

MEMORY_AFFINITY=MCM

To some extent, the decision on p650 is a tradeoff between latency and bandwidth. Striping memory access across processor books can provide higher bandwidth for sequentially accessed data. But if data is more randomly accessed, there is less memory latency for local affinity. If you don't have a clear

understanding of the application, it may be necessary to run it both striped and with MCM affinity to determine which mode provides the balance between absolute performance and repeatable CPU consumption.

Sensitivity of the Choices

In general there are three classes of workloads in regard to L3 cache and memory affinity tuning:

- Workloads that exercise memory bandwidth
- Workloads that fit well in cache
- Workloads that fit well in cache without significant sharing of unmodified data

A good example of a workload that exercises memory bandwidth is STREAM (http://www.cs.virginia.edu/stream). This benchmark is composed of a number of subcomponents which drive various patterns of memory access. Table 1 shows single process results generated on an eight processor p650 1.45 GHz model using the three modes. The data shows that shared L3 provides the most bandwidth to a single processor. If using private L3, striping the processor storage over all of the processor books of the system allows data prefetch to take advantage of more bandwidth inherent in accessing more DIMMs in parallel. But MCM memory affinity provides the least bandwidth, as all of the accesses go to the same processor book's DIMMs.

	Shared L3	**Private L3 striped memory affinity**	**Private L3 MCM memory affinity**
copy	4,669MB/sec	3,699 MB/sec	2,200 MB/sec
scale	4,726MB/sec	4,120 MB/sec	2,025 MB/sec
add	4,343 MB/sec	4,380 MB/sec	2,287 MB/sec
triad	4,360 MB/sec	4,329 MB/sec	2,403 MB/sec

Table 1 - Single Process STREAM Measures

In general, the multi-process STREAM results follow the same trend. Shared L3 and private L3 with striped memory affinity tend to have very close performance. Private L3 mode with MCM memory affinity has the least sustainable bandwidth.

A good example of workloads that fit well in cache is the SPEC SDM 057.sdet workload. This workload is composed of a set of shell scripts which mimic users. Because the shell scripts contain commands of very short duration, the processes tend to exist in the caches of the system, and they rarely spill to memory. Because the workload doesn't drive much memory bandwidth, its performance is about the same running in any of the three modes. Another application, one representing Web serving, with a bigger footprint than Sdet, runs approximately four percent better in shared L3 mode. An internal workload which mimics online transaction processing (OLTP) is an example of an application with considerable sharing of unmodified data. This large application with an enormous instruction and data footprint is the best example of a workload which benefits from private L3 mode with MCM-level memory affinity. It runs approximately ten percent faster in this mode than in shared L3 mode.

Table 2 shows the L3 mode used for a number of announced benchmarks on p650. These benchmark results show the benefit of POWER4 technology in both commercial and scientific and technical computing environments.

Benchmark	**# of processors**	**L3 mode**
SPECint_rate2000	8	Private
SPECfp_rate2000	8	Private
SPEC OMPM2001	8	Private
SPECweb99	8	Shared
SPECsfs97_R1	8	Shared
SPECjbb2000	8	Shared

Table 2 - Benchmark L3 Modes

Pitfalls of Memory Affinity

While memory affinity can be effective in many cases, there are situations where tradeoffs between affinity and balanced use of resources must be made. The most fundamental decision to be made is in weighing the value of memory affinity versus processor utilization. When starting a program with memory affinity, the operating system chooses the processor where the new program's initial thread will start running. Once the thread is running on a processor, it begins to build its address space by touching pages and experiencing page faults. The initial thread placement decision is based on the utilization of processors in the system. Generally, if there is an idle CPU, it will be the recipient of the new thread. This methodology will optimize the system for CPU balance, but not necessarily for memory balance. It is entirely possible that the least busy CPU will be associated with memory which is mostly consumed by previously running processes. In this case, as the new thread runs and touches new pages, the physical memory allocated for those pages will contain a higher percentage of remote memory. The difficulty of placement is compounded by the fact that the operating system does not know when starting a program how much memory it will consume.

A related effect occurs as a system runs. While the placement of new threads might be good, the balance of threads running in the system will likely change over time. Figure 6 shows a progression where the initial thread placement of job#5 was made based on the fact that all four of the CPU's in the complex were busy. But, since CPU1 became idle while job#1 and job#5 were both running on CPU0, we took advantage of the idle CPU by moving job#5 to CPU1. While this allows job#5 to make more rapid progress, since it is no longer competing with job#1 for CPU time, its memory accesses are now remote. This results in job#5 taking more CPU time to complete than if it had remained on CPU0. But, it will complete more rapidly in wall clock time.

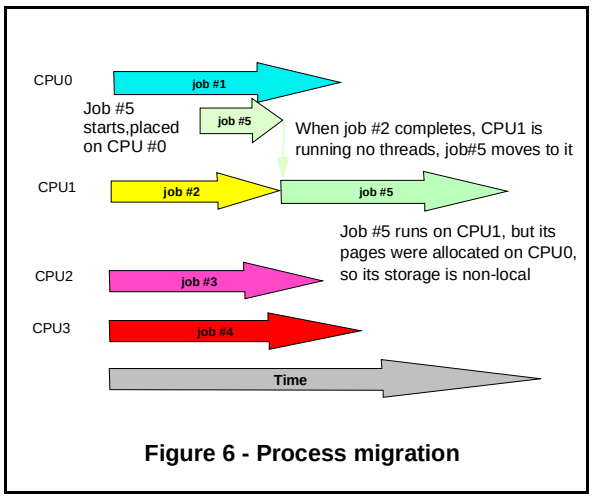


Figure 6 - Process migration

Strict placement of threads is possible with the bindprocessor command or bindprocessor system call. Binding to a processor prohibits a thread from being load balanced onto a different processor. In general, bindprocessor should be used with care, as it is not a very flexible mechanism for resource management. For more information on bindprocessor, see the AIX documentation.

RSETS

If CPU time variability is a greater concern than throughput, or the workload environment is managed to keep the number of active threads consistent with the number of processors on the system, explicit control over CPU placement may be beneficial. To that end, AIX 5L v5.2 provides mechanisms to create resource sets consistent with system topology. It is possible to initiate programs attached to resource sets in order to manage their use of system resources. For more information, consult the AIX 5L v5.2 documentation.

Recommendations

On either 1.2 GHz or 1.45 GHz models with two processors, the L3 mode firmware option is ignored. Additionally, there is no value running with memory affinity as memory is all local on the single processor book.

On 1.2 GHz models, only private L3 mode is available. If running under LPAR, where there is currently no memory affinity support, some CPU variability will be noted. If running in SMP mode, variability can be mitigated to some extent by enabling memory affinity. Expectations on the effectiveness in memory affinity solving the problem must be tempered by an understanding of the limits of the operating system with respect to the customer environment.

On four and eight processor 1.45 GHz models, the default shared L3 mode will typically be the best overall choice, whether running LPAR or SMP. In general, the tradeoff between performance and repeatable CPU consumption is best served in this mode. Under LPAR, it is particularly advantageous, since memory affinity is currently not supported. Note that six processor models must run in private L3 mode, so the recommendations for those are the same as for 1.2 GHz models. For some applications, throughput improvements can be noted in SMP mode running with memory affinity and private L3 mode. The performance gains for these atypical environments are usually on the order of a few percent, with the most extreme case improvement on the order of ten percent.

[1] pSeries 690 awards include eWeek magazine's eXcellence award for best server hardware (April 2002) and VARBusiness' Editors' Choice Award (December 2001).

IBM Corporation
Marketing Communications
Server Group
Route 100
Somers, New York 10589

Published in the United States of America
12-02