IBM

Performance and Tuning Considerations for the p690 in a Cluster 1600

Look under the hood of the pSeries 690 and the SP Switch2 interconnect

Marvel at the new features that are relevant to performance

Watch as the authors play "what if" games

Dino Quintero
Marc Genty
Soon-Kwon Ha
Tony Pirraglia
Gabriel Radu
Marcelo Ramos

Redbooks

ibm.com/redbooks

International Technical Support Organization

Performance and Tuning Considerations for the p690 in a Cluster 1600

August 2002

SG24-6841-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (August 2002)

This edition applies to Version 3, Release 4, of IBM Parallel System Support Program, for use with the IBM AIX Operating System Version 5 Release 1.

Contents

Figures

Tables

Notices

Trademarks

Preface

This redbook is based on the firm belief that all good tuning begins and ends with a fundamental understanding of how your system operates within your own environment. There are three main parts to the book. The first two are conceptual. The coverage in the first part includes pSeries 690 internals, such as POWER4, MCMs, memory subsystem, I/O subsystem, and network connectivity (including SP Switch2, SP Switch2 PCI Attachment Adapter, and EtherChannel). Coverage in the second part includes details about performance relevant features, such as affinity logical partitions (ALPARs), scheduling (processor) and memory affinity, technical large page support, and IP vs. US protocol over the new interconnect fabric. The third part details a series of investigations used to see how the new features work and interact. The coverage there includes technical large page, scheduling (processor) and memory affinity, LPAR and ALPAR, Tivoli Storage Manager (TSM), AIX mkramdisk, and IP vs. US. In addition, we have included *Configuring p690 in an IBM @server Cluster 1600*, REDP0187 as an additional appendix, which will tell you how to configure a pSeries 690 in a Cluster 1600.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a Project Leader at the ITSO Poughkeepsie Center. He currently concentrates on pSeries clustering technologies by writing Redbooks and teaching workshops.

Marc Genty is a Software Engineer in the Supercompute Systems Group at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado, where he supports (among other things) a 318 node Cluster 1600 complex and an early-release pSeries 690 system. He has worked in IT for over 22 years and in the UNIX environments for the last 12 of those years. He is an RS/6000 Certified Advanced Technical Expert (CATE), and he holds a BS degree in Manufacturing from Colorado State University. Marc has also been a contributing author on two other IBM Redbooks, *Exploiting RS/6000 SP Security: Keeping It Safe*, SG24-5521 and *Additional AIX Security Tools on IBM @server pSeries, IBM RS/6000, and SP/Cluster*, SG24-5971.

Soon-Kwon Ha is a senior IT specialist in IBM Korea. He has worked at IBM for six years as a RS/6000 and pSeries post-sales support engineer. His areas of expertise include AIX, HACMP, PSSP, system performance tuning, and storage area network (SAN). Currently, Soon-Kwon is working technical support for pSeries and AIX. He has been a contributing author on the IBM Redbook *Configuring the IBM VSS for Performance and Availability*, SG24-5279.

Tony Pirraglia is a senior Engineer in the ISV Technical Support group in Poughkeepsie, NY. While currently supporting ISVs in the Life Sciences area, Tony has been involved with Scientific and Technical computing on IBM platforms in one form or another since 1988, when he first joined IBM. He received his Doctorate in Chemical Engineering from Columbia University.

Gabriel Radu is a senior IT specialist in IBM Romania. He is an IBM Certified Specialist in AIX, HACMP, and Problem Determination. He has nine years of experience in UNIX systems. His areas of expertise include AIX, HACMP, PSSP, system performance tuning, and Linux. He graduated from the Electronics and Telecommunications Faculty in Bucharest. Currently, Gabriel Radu provides technical support for the RS/6000 and IBM pSeries and AIX.

Marcelo Ramos is an Advisory I/T Specialist in IBM Brazil. He is an AIX Certified Specialist and has been working for almost five years in the IBM Integrated Technology Services team, with customer support and services delivery for AIX, HACMP, and PSSP. His areas of expertise include AIX, HACMP, PSSP, ESS, and SAN. He also has some knowledge of Tivoli Storage Manager implementation and support.

Thanks to the following people for their contributions to this project:

International Technical Support Organization, Poughkeepsie Center
Al Schwab, Margarita Hunt, David Bennin, Peter Bertolozzi, Michael Schwartz

IBM Boulder
Tom Farwell and Ishmael Weingarten

IBM Germany
Stefan Andersson, Christoph Pospiech

IBM Austin
Raj Panda, Chris Strauss, Steve White, Bruce Hurley

IBM Dallas
Hari Reddy

IBM Poughkeepsie
Joan McComb, Jan Ranck-Gustafson, Bernard King-Smith, Frank Johnston,

James Dykman, Hok Chau, Al Dimisko, Xinghong He, Swamy Kandadai, Farid Parpia, Barry Spielberg, Clarisse Taaffe-Hedglin, Jim Wang

IBM Watson
Bob Walkup and Marge Momberger

NCAR, Boulder
George Fuentes, Gene Harano, Bernie O'Lear, Pete Peterson, Al Kellie, Dan Anderson, Pam Gillman, Tom Bettge, Bill Anderson, John Ellis

Become a published author

Comments welcome

1 Introduction

We begin with a story.

Not long ago, several close friends from the U.S. were vacationing together in Europe. The day was drawing to a close, so they decided to take a room in a small town in the south of France. The cost of the room for the night was $30.00 U.S. They paid the fee and went up to their room to get settled. Shortly afterwards, the owner of the hotel realized that he had charged them for the single-occupancy rate rather than the lower, multiple-occupancy rate. He owed them a $5.00 refund. He knew that they were planning to leave early the next morning, so he summoned his son to the front desk and asked him to take the $5.00 up to the room and explain what had happened. On the way up to the room, the son realized that splitting $5.00 three ways was far beyond his math skills. So, instead, when he got to the room, he explained the overcharge and gave them each back $1.00. Now, that means that each of them paid $9.00 for the room. Three times $9.00 is $27.00, plus the $2.00 that the son pocketed is $29.00. What happened to the other $1.00?

Performance and tuning on computers in general and on clusters in particular is much like trying to find the missing $1.00. You know the answer is right in front of you, and, as soon as you see the problem, you wonder why you were not able to see it all along. The intent of this redbook is to arm you with the additional tools, techniques, information, and references that will help you more quickly find the missing $1.00.

This redbook is meant to complement rather than replace existing Redbooks and manuals on the subject of performance and tuning. It focuses specifically on how to tune the pSeries 690 in a Cluster 1600. To help you more easily understand where this book fits in the grand scheme of things, Table 1-1 provides an abbreviated list of this and other related IBM books on the subject.

Table 1-1 Other related IBM publications

Description	**Redpaper, redbook, whitepaper, or manual**
How to tune the p690 in a Cluster 1600.	*Performance and Tuning Considerations for the p690 in a Cluster 1600*, SG24-6841 (redbook).
How to configure the p690 in a Cluster 1600.	*Configuring p690 in an IBM @server Cluster 1600*, REDP0187 (Redpaper). We have also included this redpaper as Appendix D, “Integrating p690 in an IBM eServer Cluster 1600” on page 183 in this redbook.
How to tune an RS/6000 SP system.	*RS/6000 SP System Performance Tuning Update*, SG24-5340 (redbook).
How to tune an AIX system.	*AIX Performance Monitoring and Tuning Guide*, SC23-2365 (manual).
How to tune applications for the POWER4 processor.	*The POWER4 Processor Introduction and Tuning Guide*, SG24-7041 (redbook).
How to monitor and measure the systems in the cluster.	*AIX 5L Performance Tools Handbook*, SG24-6039 (redbook).

The what, where, when, why, and how of performance tuning in a clustered environment are well documented in the *RS/6000 SP System Performance Tuning Update*, SG24-5340. What follows is a very brief summary of what we feel are the key points to performance tuning in a clustered environment:

- Know your system, and know your workload.
- Document before and after you make a change.
- Change one thing at a time, and have a backout plan.
- Know how to measure what you are changing.
- When possible, measure with more than one tool.
- Distinguish between systemic versus isolated performance issues.
- For isolated (node) problems, look for bottlenecks in the following order: CPU, memory, disk I/O, and network. For systemic (cluster) problems, look for bottlenecks in the opposite order.

In the chapters that follow, we give you an overview of the hardware components that you can use to build a Cluster 1600 from one or more POWER4 pSeries 690 frames. We cover two types of interconnects: the SP Switch2 with the SP Switch2 PCI Attachment Adapter, and the 10/100/1000 Ethernet. Following the hardware overview, we move on to cover new features that are important to performance, such as logical partitions (LPARs) and affinity logical partitions (ALPARs), scheduling and memory affinity, technical large page support, 32-bit versus 64-bit kernel, aggregate IP over the SP Switch2 with the SP Switch2 PCI Attachment Adapter, and the use of Internet Protocol (IP) versus User Space (US) protocol across the SP Switch2. Finally, we end with a description of and findings from a number of “what if” scenarios that we tested in our laboratory. In the appendices, you will find a summary of the tools that we found useful for measuring, monitoring, poking, and prodding these clustered systems. This is also where we have collected the scripts that were used during our experiments in the laboratory, as well as those that we brought with us from our home sites and thought might be of use to others.

2

Hardware overview

This chapter provides the conceptual framework for the remainder of the book. We discuss some of the key components that comprise a Cluster 1600 complex. This is by no means a complete list of every possible configuration, nor is it an in-depth treatise on hardware internals. There are already a number of good IBM manuals, Redbooks, whitepapers, and Redpapers on these subjects. Instead, this chapter focuses on the new components that you can use to construct or integrate into your Cluster 1600 complex. The topics covered are:

- What is a Cluster 1600?
- The pSeries 690 POWER4 building blocks.
- The interconnect fabric.

You may have noticed that there is no mention of external storage systems. There is already a wealth of information available on this subject, so there is no need to duplicate it here. Here are a few references to get you started:

- Logical Volume Manager (LVM): *AIX Logical Volume Manager, From A to Z: Introduction and Concepts*, SG24-5432
- Virtual Shared Disk (VSD) and General Parallel File System (GPFS): *RS/6000 SP System Performance Tuning Update*, SG24-5340
- Journaled File System 2 (JFS2): *AIX 5L Differences Guide*, SG24-5765

In Chapter 3, “Features relevant to performance” on page 35, we build on the foundation laid by this one. We go into more detail about the performance implications and trade-offs that you need to consider when architecting and configuring these components into your cluster. The information presented in this chapter is a prerequisite for all that follows.

2.1 What is a Cluster 1600

So, what is a Cluster 1600 anyway? According to one definition it is:

> **Cluster 1600 (klus´ter siks´teen´ hun´drid) noun.** 1. A Cluster 1600 is a new cluster name and ID that unifies pSeries cluster offerings, broadens the range of building blocks, increases configuration flexibility, and extends the cluster management domain.

In other words, a Cluster 1600 complex is what used to be called an RS/6000 SP complex, but the name has been changed to emphasize the fact that it now encompasses much more than frames with SP nodes and an occasional SP-attached server. In fact, it is possible to have a Cluster 1600 complex comprised entirely of pSeries nodes. Table 2-1 lists the currently supported node building blocks for the Cluster 1600.

Table 2-1 Cluster 1600 node building blocks

Model type	Description	Switch adapters
7040-681	pSeries 690 Model 681	SP System Attachment Adapter/SP Switch2 PCI Attachment Adapter
7017-S85	pSeries 680 Model S85	SP System Attachment Adapter/SP Switch2 PCI Attachment Adapter
7040-671	pSeries 670 Model 671	SP System Attachment Adapter/SP Switch2 PCI Adapter
7026-6M1	pSeries 660 Model 6M1	SP System Attachment Adapter/SP Switch2 PCI Attachment Adapter
7026-6H0	pSeries 660 Model 6H0	SP System Attachment Adapter/SP Switch2 PCI Attachment Adapter

Model type	Description	Switch adapters
7026-6H1	pSeries 660 Model 6H1	SP System Attachment Adapter/SP Switch2 PCI Attachment Adapter
9076-2058	375MHz SMP High Node	SP Switch MX2 Adapter/SP Switch2 Adapter
9076-2057	375/450MHz SMP Wide Node	SP Switch MX Adapter/SP Switch2 MX2 Adapter
9076-2056	375/450MHz SMP Thin Node	SP Switch MX Adapter/SP Switch2 MX2 Adapter
9076-2055		N/A

This redbook focuses exclusively on the performance of pSeries 690 node types. For information on the performance of other node types, see the *RS/6000 SP System Performance Tuning Update*, SG24-5340.

There are now new switch interconnect options available with the Cluster 1600. In addition to the supported SP Switch, there is the SP Switch2, which brings with it the SP Switch2 PCI Attachment Adapter for the pSeries 690 nodes. Table 2-2 lists the raw peak performance numbers for both latency and bandwidth on the two SP switch types.

Table 2-2 Interconnect hardware latency and peak bandwidth

Nodes in the cluster	Interconnect type	Latency (usec)	Bandwidth (MB/sec) [unidirectional]	Bandwidth (MB/sec) [bidirectional]
Up to 16	SP Switch2	1.0	500	1000
17 to 80		1.5		
81 to 512		2.5		
Up to 16	SP Switch	1.3	150	300
17 to 80		1.9		
81 to 512		3.3		

For more information on the SP Switch versus SP Switch2 performance numbers, see the *RS/6000 SP: SP Switch and SP Switch2 Performance* whitepaper at:

`http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch_perf.html`

The pSeries 690 nodes and the new interconnect fabric are covered in the sections that follow.

2.2 The pSeries 690 POWER4 building blocks

The latest addition to the list of Cluster 1600 node building blocks is the new pSeries 690 POWER4 system. The pSeries 690 is an enterprise class server incorporating the following features:

- POWER4 processors (1.1 GHz or 1.3 GHz)
- From one to four Multi-Chip Modules (MCMs), each with:
  - Either four 2-way or four 1-way (HPC option) POWER4 chips.
  - Individual L1 data and instruction caches.
  - Shared L2 cache running at processor speed. (Note that the L2 is not shared on models configured with the HPC option.)
  - The L3 directory and cache controller.
  - The fabric bus controller, providing inter-chip and intra-chip interconnection.
  - The GX bus controller, providing connection to external I/O devices.
- Scalable SMP, from 8-way up to 32-way
- Scalable memory, from 8 GB up to 256 GB
- On-chip L1 and L2 cache and off-chip L3 cache
- Support for logical partitioning (LPARs)
- Scalable I/O, from one to eight drawers. Each drawer contains 20 PCI slots and up to 16 disk drives

There are three types of configurations for the pSeries 690:

- The turbo configuration, with eight 1.3 GHz POWER4 processors per MCM and shared L2 cache. It is available as an 8-way (one MCM), a 16-way (two MCMs), a 24-way (three MCMs), or a 32-way (four MCMs) system.

- The standard configuration with eight 1.1 GHz POWER4 processors per MCM and shared L2 cache. It is available as an 8-way (one MCM), a 16-way (two MCMs), a 24-way (three MCMs), or a 32-way (four MCMs) system.
- The High Performance Computing (HPC) configuration with four 1.3 GHz POWER4 processors per Multi-Chip Module (MCM) and dedicated L2 cache. It is available as either an 8-way (two MCMs) or a 16-way (four MCMs) system. This configuration has only four processors per MCM, which effectively doubles the amount of per processor L2 and L3 cache and increases the effective memory bandwidth. This is because the memory subsystem is the same as with the turbo and standard configurations, but the number of processors per chip is half.

Unless otherwise noted, the turbo configuration will be used to illustrate the concepts in this chapter and the next.

In a Cluster 1600 configuration, each pSeries 690 appears as a frame and each LPAR appears as a thin node. There can be from one to 16 LPARs per pSeries 690, but, as we will explain later, there are certain constraints that can put a practical limit on this number to something below 16.

In the discussion that follows, we break the hardware down into:

- The processor subsystem
- The memory subsystem
- The input/output subsystem

Once these have been described, we move on to cover the subject of logical partitions (LPARs). An LPAR is a strange form of ethereal spirit that dwells in the realm between the world of hardware and the world of software. To hardware, LPARs look like software, but to software, LPARs look like hardware. This is a bit of an oversimplification, but as you begin to try to explain this stuff to your coworkers and others, you will perfectly understand the dilemma.

Before we dig into the guts of the pSeries 690, we thought it might be fun to give you the big picture. The heart of the pSeries 690 is contained in a Central Electronics Complex (CEC). This is where the MCMs, memory cards, and connections to the I/O drawers are located. Much of what is shown in Figure 2-1 on page 11 will not be covered in this redbook. Consider this the Rosetta stone for the pSeries 690.

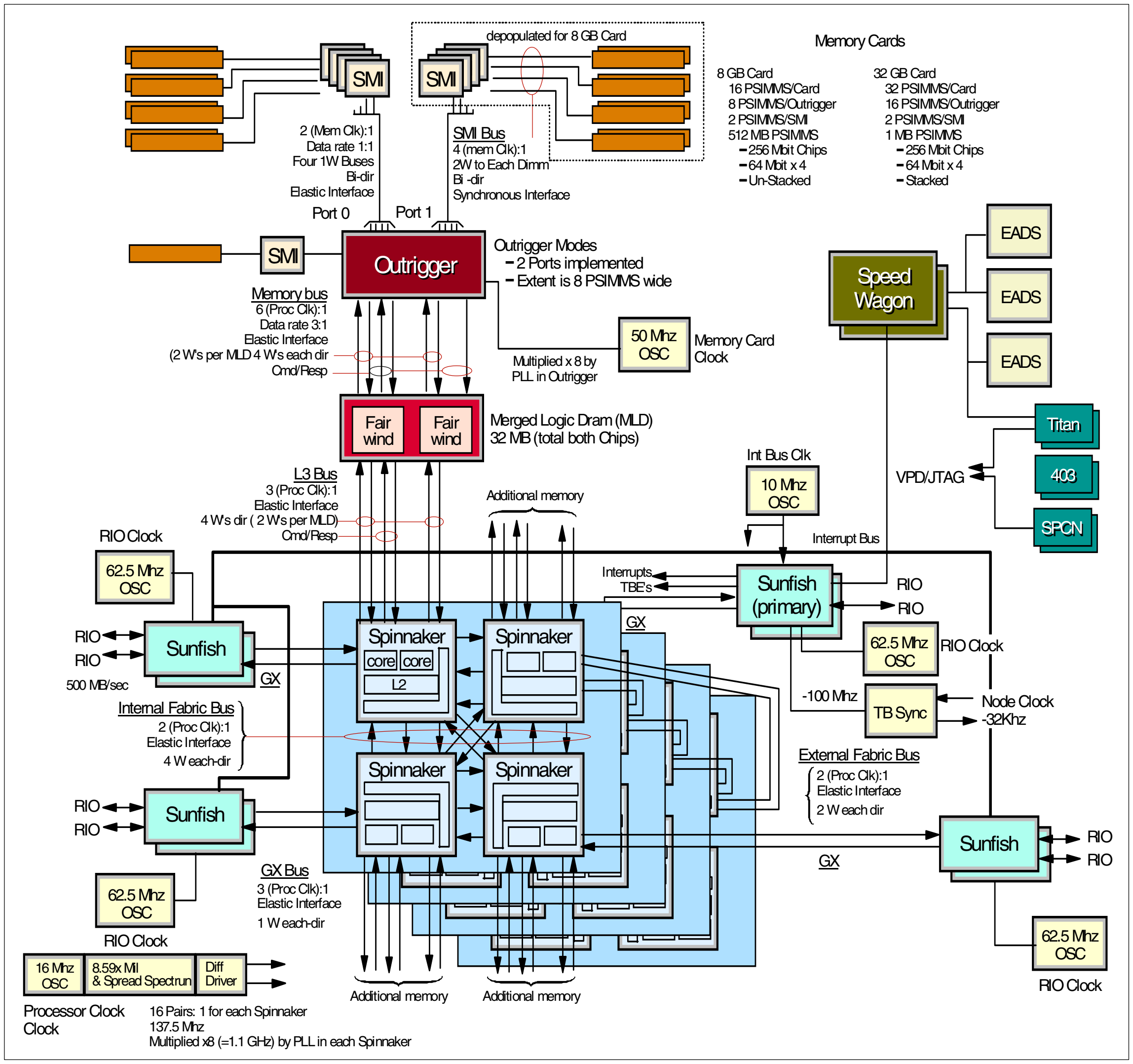


Figure 2-1 The pSeries 690 architecture

Table 2-3 provides the translation between the diagram component code names and their common external names. Both names are included here because you are likely to run across either name in other publications about the POWER4 architecture.

Table 2-3 Mapping of POWER4 architecture component names

Diagram component name	Common external name
Spinnaker	POWER4 chip
Sunfish	RIO hub
Fairwind	L3 cache chip
Outrigger	Memory controller
Speed Wagon	RIO to PCI bridge chip
EADS	PCI Host Bridge (PHB) chip

If you would like to read more about the low-level internals of the pSeries 690 nodes, we recommend the following two whitepapers:

- *The IBM @server pSeries 690 Reliability, Availability, Serviceability (RAS)*, found at:

  `http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690_ras.html`

- *IBM @server POWER4 System Microarchitecture*, found at:

  `http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html`

(The pSeries 690 architecture figure is from the first whitepaper listed here.)

2.2.1 The processor subsystem

The pSeries 690 Central Electronics Complex (CEC) houses from one to four Multi-Chip Modules (MCMs). Each MCM houses either four 2-way or four 1-way POWER4 chips, the shared L2 cache, the Fabric Controller for intra-chip and inter-chip communications, and four GX bus links for connections to external I/O devices. Also associated with, but not located on each MCM, is an L3 cache, comprised of two 16 MB eDRAM chips and a memory controller chip that connects to the L3 cache on one side and the Synchronous Memory Interface (SMI) chips on the other. Figure 2-2 on page 13 provides a simplified logical view of an MCM.

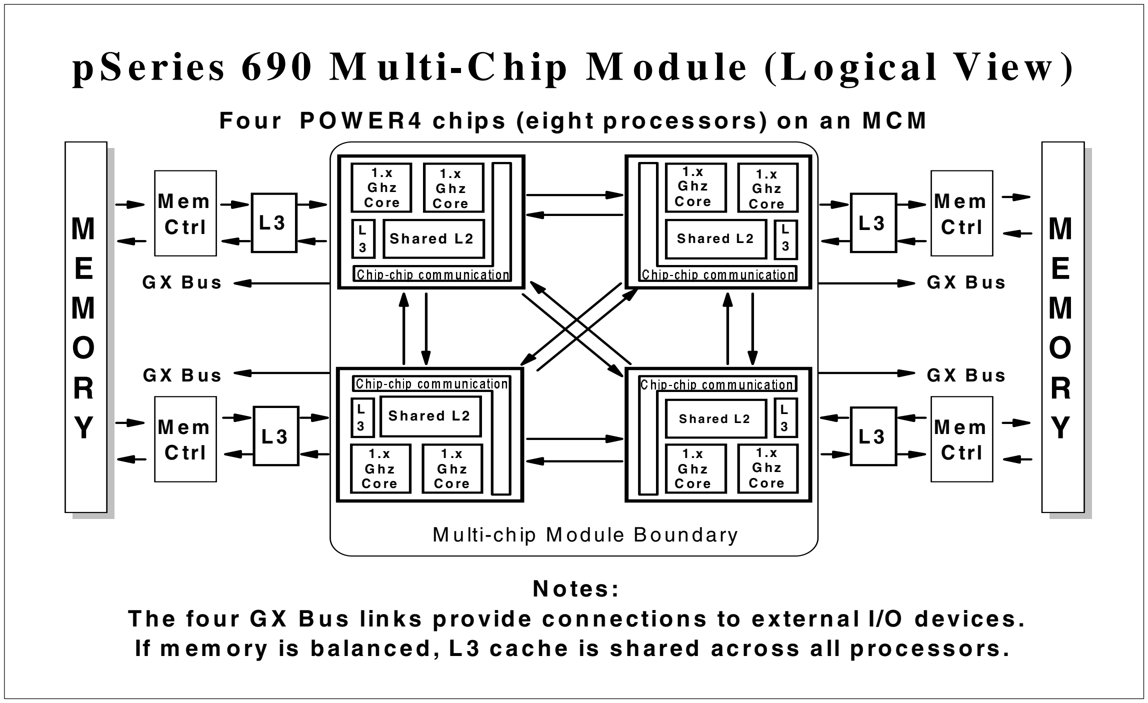

pSeries 690 Multi-Chip Module (Logical View)
Four POWER4 chips (eight processors) on an MCM
MEMORY
Mem Ctrl
L3
GX Bus
1.x Ghz Core
Shared L2
L 3
Chip-chip communication
Multi-chip Module Boundary
Notes:
The four GX Bus links provide connections to external I/O devices.
If memory is balanced, L3 cache is shared across all processors.

Figure 2-2 The pSeries 690 Multi-Chip Module (MCM)

It is interesting to note that each POWER4 chip contains more than one mile of connections in the seven layers of metal on the chip and is made up of more than 174 million transistors. The two processors and cache on an MCM are able to transfer data at speeds approaching 125 GB per second. This is the equivalent of downloading 25 full-length DVD movies in a single second.

There can be from one to four MCMs in a CEC. The MCMs are connected to each other through the Fabric Controller, which runs at half of the processor speed and provides a bandwidth of less than 40 GB/sec. The inter-MCM connection topology is that of a unidirectional ring. Figure 2-3 on page 14 shows a fully-populated CEC with four 8-way MCMs.

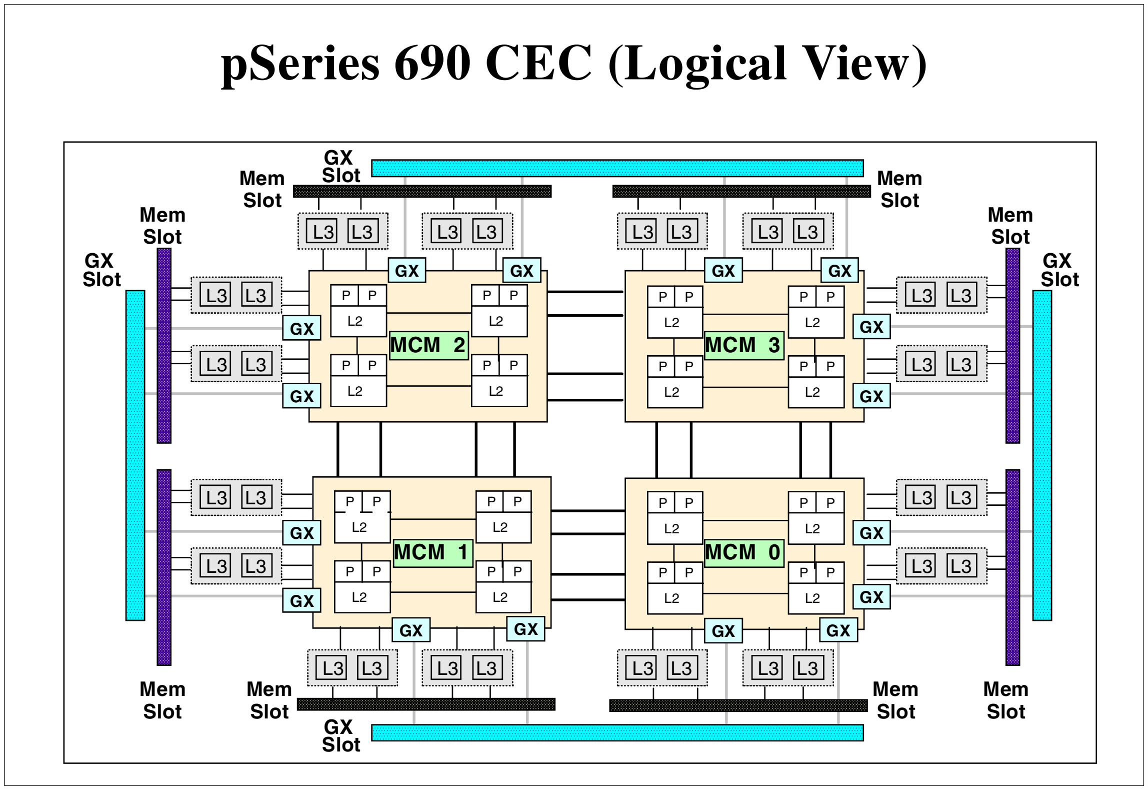


Figure 2-3 pSeries 690 fully-populated CEC

Now let us take a deeper look into the internals of the POWER4 chip to see how it connects to the other chips within the local MCM as well as to the other MCMs within the CEC. Figure 2-4 on page 15 provides a logical view of the POWER4 chip.

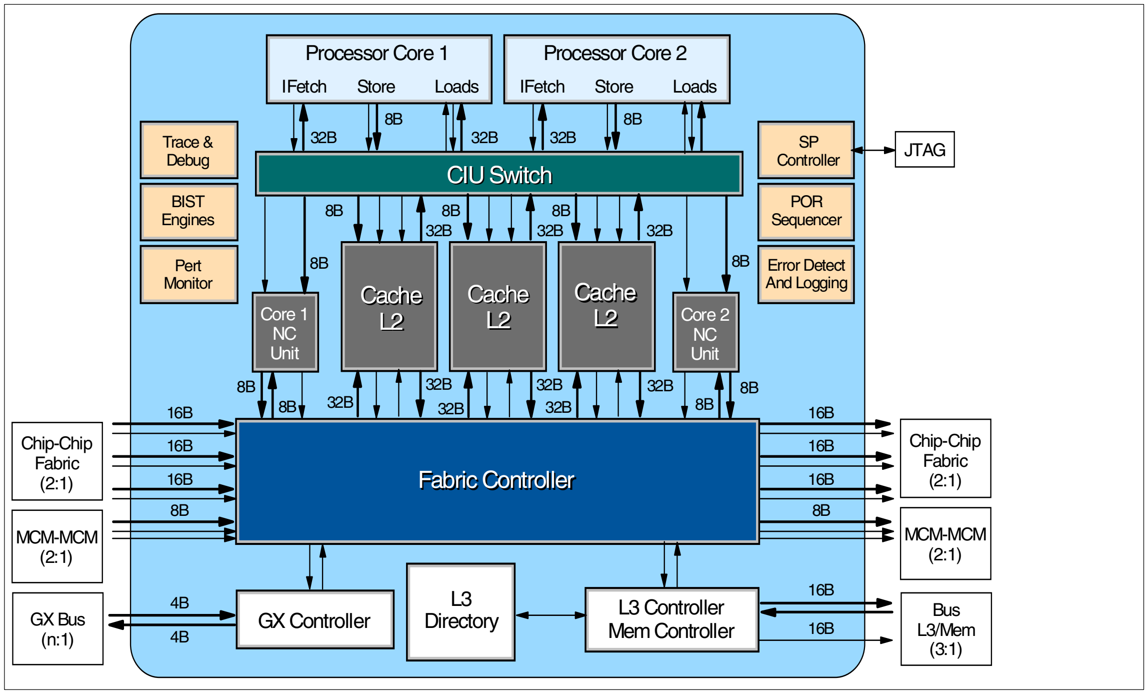


Figure 2-4 Inside the POWER4 chip

Each POWER4 chip has a maximum of two 64-bit PowerPC microprocessors, which are based on a speculative superscalar out-of-order execution design. Each microprocessor has its own dedicated data and instruction L1 caches. Also on the chip is a unified L2 cache shared by both microprocessors through the Core Interface Unit (CIU) switch. The L2 cache is physically divided into three, equal-sized parts, each with its own L2 cache controller. The CIU switch connects each of the three L2 cache controllers to each processor through separate 32-byte wide data reload and instruction reload ports. Also connected to the CIU switch from each microprocessor is an 8-byte wide store port that is used to store data through the appropriate L2 cache controller. Each microprocessor also has an associated non-cacheable unit (NCU) that is logically considered part of the L2 cache.

The directory for the L3 cache and the L3 cache controller are also located on the POWER4 chip. (The L3 cache itself is located off the chip.) Having the L3 cache directory and controller on the chip improves performance by reducing the latency to memory.

The Fabric Controller acts as a crossbar switch to provide master control of the network of buses on the POWER4 chip. These buses provide the communications pathways between:

- Components within a single chip (local traffic)
- Chips within a single MCM (intra-chip traffic)
- MCMs within a single CEC (inter-chip traffic)

The Fabric Controller also handles the snooping and cache-coherency duties for these buses. Some of the communications pathways for which the Fabric Controller provides control are:

- The 16-byte wide buses running at half of the processor speed, which implement a point-to-point, unidirectional network between each of the four chips on an MCM (intra-chip).
- The 8-byte wide buses running at half of the processor speed, which connect each chip to a corresponding chip in a neighboring MCM (inter-chip).
- The 16-byte wide buses running at a third of the processor speed, which implement an unidirectional connection between the POWER4 chip and the off-chip, L3 cache.
- The connection to the GX controller. The GX controller then handles the communications across the two 4-byte wide buses, which connect the POWER4 chip to the external I/O devices.

The interconnection topology appears like a bus-based system from the perspective of a single chip, but from the perspective of the MCM, it appears like a switch. All of the buses that interconnect the POWER4 chips, whether or not they are on or off the module, operate at half the processor speed. As future technology is exploited, allowing chip sizes to decrease and operating frequencies to increase, system balance is maintained, as bus speeds are no longer fixed but instead tied to the processor frequency.

Finally, there are areas on the chip that fall under the category of pervasive functions. For our purposes, the most notable of these is the Performance Monitoring Unit (PMU). The PMU implements low-level performance monitoring features at the hardware level that can be exploited through software. For more information about the POWER4 Performance Monitor, see *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041.

2.2.2 The memory subsystem

The dividing line for where the memory subsystem begins is not black and white. For the purposes of this book, we define the memory storage hierarchy as follows:

1. L1 data and instruction caches
2. L2 shared caches
3. L3 directories and controllers
4. L3 off-chip caches
5. Memory controllers and cards (packaged within memory books)

Table 2-4 lists the organization and per-chip capacity for each of the components in the memory storage hierarchy.

Table 2-4 POWER4 memory and cache organization and per-chip capacity

Component	**Organization**	**Capacity (per chip)**
L1 Instruction	Direct map 128-byte line (4 x 32-byte sectors)	128 KB/chip (64 KB/processor)
L1 Data	Two-way 128-byte line	64 KB/chip (32 KB/processor)
L2	Eight-way 128-byte line	1440 KB/chip
L3	Eight-way 512-byte line (4 x 128-byte sectors)	32 MB/chip (128 MB/MCM)
Memory	N/A	0 - 16 GB

The following brief descriptions of the caches are excerpted from *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041.

L1 instruction cache

Each POWER4 microprocessor (which is one of the components on the POWER4 chip) contains an L1 instruction cache that is 64 KB in size, direct mapped, and indexed by the effective address of the instruction cache line. It is capable of either one 32-byte read or write each cycle.

L1 data cache

Each POWER4 microprocessor (which is one of the components on the POWER4 chip) contains an L1 data cache that is 32 KB in size, two-way set associative, and has a replacement policy of first-in-first-out (FIFO). It is capable of two eight-byte reads and one eight-byte write per cycle.

L2 cache

Each POWER4 chip contains an L2 cache that is supervised by three L2 controllers, each of which manages 480 KB, for a total L2 cache size of 1440 KB. Cache lines are hashed across the three controllers. Cache line replacement is implemented as a binary-tree, pseudo-least-recently-used (LRU) algorithm. The L2 cache is a unified cache, caching data, instructions, and page table entries. If there are two POWER4 microprocessors on the chip, they share the single L2 cache. Memory coherency in the system is enforced primarily at the L2 cache level by the L2 cache controllers.

L3 cache

The L3 cache is an eight-way, set-associative organized in 512-byte blocks, but with coherence still maintained in the system cache line size of 128 bytes. POWER4 chips are connected to memory through an L3 cache. The L3 cache is designed to be combined with other L3 caches on the same MCM in pairs or quadruplets to create a larger, address-interleaved L3 cache of 64 MB or 128 MB. Combining L3 caches into groups not only increases the L3 cache size, but it also increases the L3 bandwidth available to any processor. When combined into groups, L3 caches (and the memory behind them) are interleaved with a 512-byte granularity.

Memory

Each POWER4 chip can have an optional memory controller attached behind the L3 cache. Memory controllers are packaged two to a memory card and support two of the four POWER4 chips on an MCM. Each MCM can attach to zero, one, or two memory cards, for a total of eight memory cards in a fully-populated CEC. The memory controllers have either one (single-port) or two (dual-port) ports to memory, depending on the capacity of the memory card. (The 4 GB and 8 GB memory cards are single-port, and the 16 GB and 32 GB memory cards are dual-port. The dual-port cards provide higher performance through greater bandwidth.) The minimum amount of memory a pSeries 690 can have is 8 GB, and the maximum is 256 GB. A logical view of memory is shown in Table 2-5 on page 19.

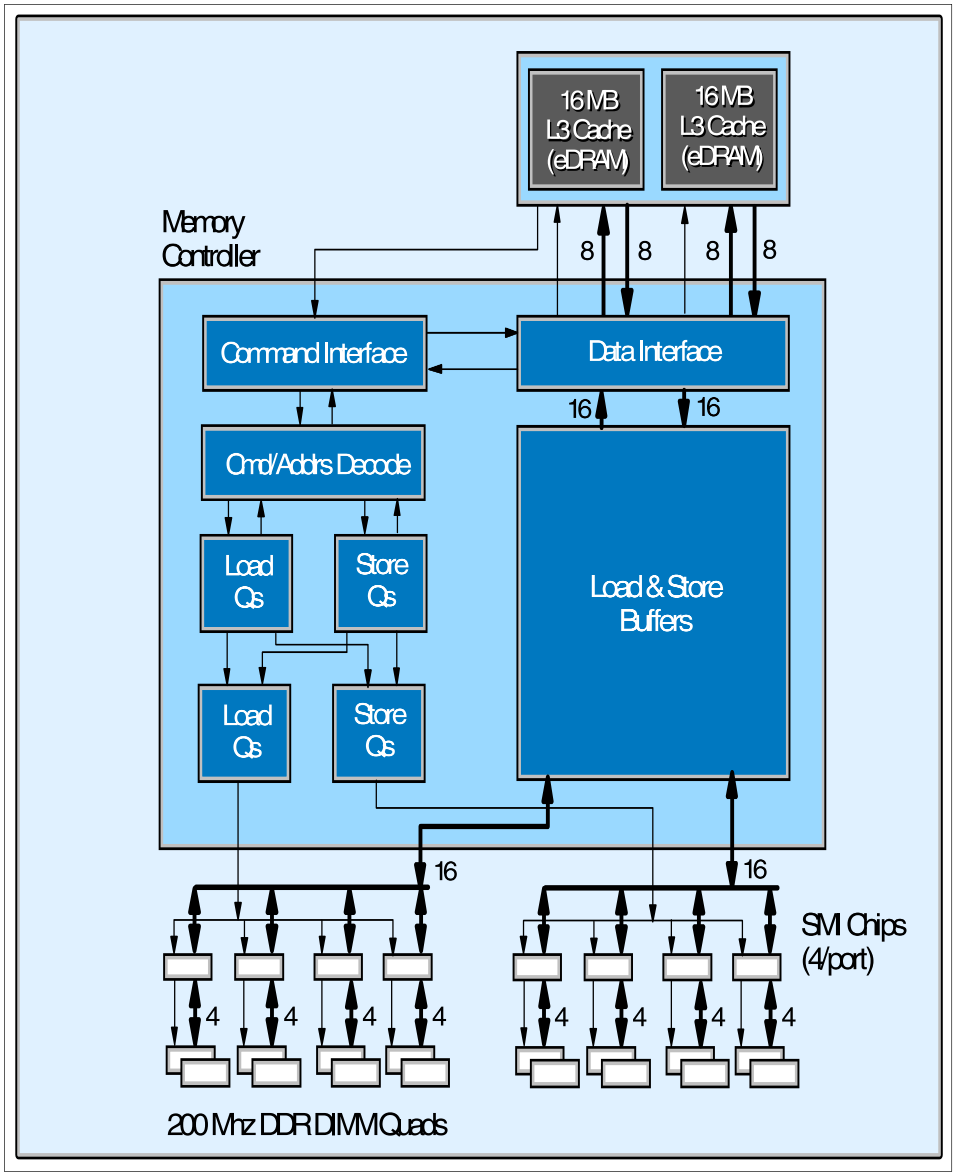


Figure 2-5 Logical view of POWER4 memory

Memory balancing

In general, a balanced memory configuration is critical for achieving optimum and consistent performance in both commercial and technical computing environments. To maximize memory performance on the pSeries 690, memory interleaving is employed. If an MCM has two memory cards of the same size installed, memory is interleaved in a round-robin manner across the four memory controllers with a 512-byte granularity. If an MCM has two memory cards of different sizes attached, the two cards are treated independently with each card being two-way interleaved. If an MCM has only one memory card attached, the single card will be interleaved by two of the four L3 caches, leaving the other two unused.

For best memory performance, we recommend the following:

- All available memory slots should be populated with memory cards. In other words, it is better to have eight 4 GB cards than four 8 GB cards for a CEC fully-populated with four MCMs.
- Memory sizes should be balanced as closely as possible across all populated MCM locations.
- Given the choice between fully populating all memory slots with single-port cards or leaving some slots empty and using dual-port cards, go with the single-port cards. If you can populate all slots with dual-port cards, so much the better.
- Note that maximum sustainable bandwidth for an MCM is achieved when memory is equal to or greater than 32 GB per MCM (two 16 GB cards).
- The 32-bit kernel can address a maximum of 96 GB. If the system is configured with more than 96 GB, the only way to reach the additional memory is by configuring the system into multiple logical partitions (LPARs).

Important:

- Each MCM has two memory slots available. Therefore, one MCM is limited to two memory books.
- If you only have one memory book, then you are cutting your L3 cache in half on the MCM.
- It is important to populate the two memory slots available when adding an MCM to a pSeries 690.

2.2.3 The input/output (I/O) subsystem

The I/O subsystem on a pSeries 690 is comprised of from one to eight remote I/O (RIO) drawers. Each drawer provides 20 hot-pluggable PCI slots and four drive bays, each capable of holding four disk drives. A maximum of two RIO drawers can be attached per MCM, which means that a 16-way (non-HPC) pSeries 690 can have a maximum of four RIO drawers, rather than eight. Each RIO drawer is four Electronics Industries Association (EIA) units high, and is identified to the pSeries 690 by the EIA position within the pSeries 690 frame. For example, the one required RIO drawer is U1.9, with the additional RIO drawers in the same frame being U1.5, U1.1, and U1.13. U1.9, U1.5, U1.1, and U1.13 are the EIA positions in the frame.

Each RIO drawer is divided into two halves, with each being controlled by a RIO-to-PCI bridge chip. These bridge chips connect to the RIO hub via the RIO bus, which provides dual-path failover to the hub. The RIO hub is directly connected to the GX bus. The two halves of the RIO drawer are referred to as *Bonnie and Clyde*.

Bonnie and Clyde

There are two halves to each RIO drawer. One half is referred to as *Bonnie*, and the other half is referred to as *Clyde*. The RIO subsystem (Bonnie plus Clyde) consists of two planar I/O boards (20 PCI slots total), two riser cards, up to 16 disk drives (capacity for more than 500 GB of disk storage), four DASD backplanes (two each per Bonnie or Clyde, refer to Figure 2-6 on page 24), a midplane card, four blower fans, and two DC power supplies (DCAs). Each planar I/O board (Bonnie or Clyde) can hold up to ten PCI cards and a riser card.

Up to eight RIO drawers can be attached to a pSeries 690 CEC. Each RIO drawer connects to the system through two RIO ports, with 1.1 GB/sec of total sustained bandwidth per drawer.

Each RIO drawer can be shared by several active partitions, but PCI slots cannot be shared by multiple, active partitions. PCI slots that share PCI bridges or drawer connections can be allocated to different partitions, but all child devices, such as DASD, under an adapter must be assigned to a single partition.

The pSeries 690 internal I/O subsystem scalability and expandability supports:

- Up to eight RIO drawers, each containing up to 20 PCI hot-swappable (via advance blind swap books for easy replacement) adapters
- Up to eight RIO loops, with support for one RIO drawer per loop
- Up to 873.6 GB of internal disk storage

The RIO subsystem provides an industry-standard PCI subsystem that supports 64-bit adapters. Improved application throughput is achieved with two independent RIO ports per drawer. These RIO ports provide up to 1.1 GB per second of aggregate bandwidth per drawer. (RIO drawers are connected to a single pSeries 690 system through RIO cables.)

There are four disk storage bays connected to the four integrated Ultra3 SCSI adapters, with each bay connected to a SCSI controller. Each DASD cage can contain four disk drives for a total of 16 internal disk drives per RIO drawer. Each individual or group of I/O slots can be assigned to a logical partition (LPAR), and the integrated SCSI adapters can also be individually assigned to an LPAR.

However, all disks in a disk 4-pack must be assigned to a single LPAR because they share the same SCSI bus connection to the adapter. Supported disk drives are as follows:

- FC 3157 - 10K RPM, 18.2 GB capacity
- FC 3158 - 10K RPM, 36.4 GB capacity
- FC 3159 - 10K RPM, 73.4 GB capacity
- FC 3260 - 15K RPM, 18.2 GB capacity
- FC 3261 - 15K RPM, 36.4 GB capacity

There are two front and two rear media bays. The first front bay can contain either a CD-ROM or a DVD-RAM. The second front bay can contain either a CD-ROM, a DVD-RAM, or a 4 mm tape drive. Use of the rear media bays is optional. They can only contain a CD-ROM. The front and the rear media bays each require a power cable, a SCSI cable, and a SCSI adapter.

The RIO power connections provide redundant communication connections to redundant (logical) System Power Control Network (SPCN) processors in the RIO drawers.

I/O adapters

The following I/O adapters are supported on pSeries 690:

- Communication adapters
  - FC 2732 - Serial Short-Wave HiPPI PCI adapter
  - FC 2733 - Serial Long-Wave HiPPI adapter
  - FC 2741 - SysKonnect FDDI-LP SAS adapter
  - FC 2742 - SysKonnect FDDI-LP DAS adapter
  - FC 2946 - Turboways 622 ATM adapter
  - FC 2969 - Gigabit Ethernet SX adapter
  - FC 2975 - 10/100/1000 Base-T Ethernet adapter
  - FC 4953 - ATM UTP adapter
  - FC 4957 - ATM MMF adapter
  - FC 4959 - Token-Ring PCI adapter
  - FC 4962 - 10/100 Ethernet adapter
  - FC 4961 - 10/100 4-port Ethernet adapter
- SP switch attachment adapters
  - FC 8396 - SP System Attachment Adapter
  - FC 8397 - SP Switch2 PCI Attachment Adapter
- Disk and tape attachment adapters
  - FC 2751 - S390 ESCON Channel PCI adapter
  - FC 6203 - Ultra3 SCSI adapter
  - FC 6204 - Universal Differential SCSI adapter
  - FC 6206 - PCI Single-Ended Ultra-SCSI adapter
  - FC 6228 - Gigabit Fibre Channel adapter
  - FC 6230 - Advance Serial RAID Plus adapter

  - FC 6231 - 128 MB Option Card adapter
  - FC 6235 - 32 MB Fast-Write Cache Option Card adapter
- Cryptographic adapters
  - FC 4960 - e-business Cryptographic Accelerator adapter
  - FC 4963 - Cryptographic Coprocessor adapter
- Asynchronous port adapters
  - FC 2737 - 4-Port USB adapter
  - FC 2943 - 8-Port Asynchronous EIA-232E/RS-422A adapter
  - FC 2944 - 128-Port Asynchronous Controller adapter
  - FC 2962 - 2-Port Multi-protocol adapter
  - FC 2947 - 4-Port RVX adapter
- Graphics adapters
  - FC 2848 - GXT 135P Graphics adapter

For information on the correct placement for each type of adapter, see the *PCI Adapter Placement Reference Guide*, SA38-0538, and consult with your local IBM Customer Engineer (IBM CE).

The pSeries 690 RIO is illustrated in Figure 2-6 on page 24. This figure includes the bandwidths and bus sizes for the PCI slots. Planar 1 is Bonnie, and Planar 2 is Clyde.

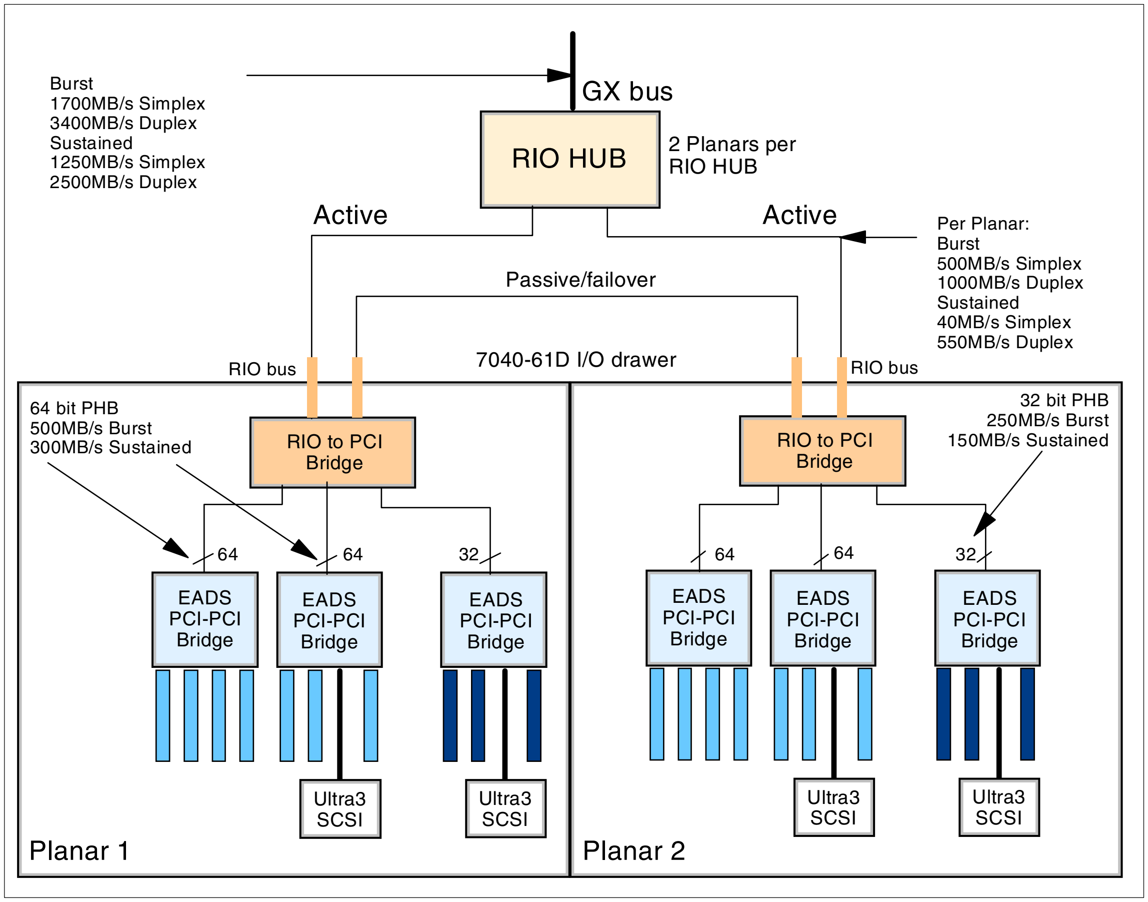


Figure 2-6 The many stages of the RIO subsystem

Notes for Figure 2-6:

- The PCI-PCI Bridge chips are also known as EADS PCI-PCI Bridge chips or PHB (PCI Host Bus).
- If one of the I/O planar fails, the adapters can be accessed via the second active planar using the passive/failover path.
- For more information on adapter placement, refer to “Adapter placement for p690” on page 202, “SPLAN adapter” on page 217, Section 2.3.2, “The SP Switch2 PCI Attachment Adapter” on page 29, or consult the *PCI Adapter Placement Reference Guide*, SA38-0538.

I/O sizing

In order to determine the I/O sizing of the system, you need to know what kind of application will be running on the system.

In general, there are two main categories of workload: technical and commercial.

Technical workloads

Technical workloads typically place high bandwidth requirements on the I/O subsystem because they read and write large amounts of sequential data for short periods during execution. Many technical workloads have been optimized for execution to enable staging of time-sensitive data from the I/O devices, which in turn reduces the peak I/O demand.

Commercial workloads

Commercial workloads (with the exception of business intelligence, which behaves more like a technical application) tend to require low I/O bandwidth. These workloads, such as transaction processing and Web server applications, normally have large numbers of reads and writes spread randomly over the I/O devices. While the volume of disk accesses for this category are large, the amount of data transferred for each access is relatively small (4 KB to 16 KB). Database systems often cache table indexes in memory for use during processing to minimize table reads, and this also tends to keep I/O bandwidth requirements low. The large volume of random disk accesses leads to the need for large numbers of disk arms. It is common for systems supporting database systems to require so many disk arms that there is unused space on the disks containing the database. In this case, it is usually advisable to purchase the smallest capacity, high-speed disk drives available to provide for the large numbers of disks.

Once each workload has been characterized, the next step is to identify the numbers and types of I/O devices that will be required to support each workload, and the amount of I/O bandwidth that will be handled by each device. Note that device peak bandwidth may be much higher than sustained bandwidth, and that peak loads rarely occur on all devices simultaneously. To minimize I/O latencies to the pSeries 690 processors and optimize the overall performance of the pSeries 690, the adapters and I/O subsystems should be planned, both in numbers and through placement, to operate at 60 percent to 80 percent of their hardware capabilities.

Table 2-5 on page 26 lists some commonly used I/O adapters and typical bandwidths that can be expected from each, along with the maximum numbers of each type that can be plugged into a RIO drawer.

Table 2-5 Commonly used adapters, bandwidths, and limits per RIO drawer

	Bandwidth	64-bit EADS (4)	32-bit EADS (2)
SP Switch2 PCI Attachment Adapter FC 8397	179 MB/sec	4/drawer	0
SP System Attachment Adapter FC 8396	N/A	0	2/drawer
2 Gb Fibre Channel FC 6228	150 MB/sec	8/drawer	2/drawer
1 Gb Ethernet FC 2969 & FC 2975	150 MB/sec	8/drawer	2/drawer
Dual Ultra-3 SCSI FC 6203	175 MB/sec	8/drawer	2/drawer
622 ATM FC 2946	100 MB/sec	8/drawer	2/drawer
SSA 40 FC 6230	90 MB/sec	10/drawer	2/drawer

Table 2-6 lists the stages of the I/O subsystem, along with the bandwidths supported by each. The RIO hub stage, with its sustained duplex and simplex bandwidth rates of 1100 MB/sec and 800 MB/sec, respectively, provides the effective limit to the numbers of adapters that can be plugged into each RIO drawer. The pSeries 690 I/O architecture enables the bandwidth supported by the system to scale with the number of drawers attached.

Table 2-6 Bandwidth capabilities by I/O stage

Stage	Burst duplex	Burst simplex	Sustained duplex	Sustained simplex
64-bit PHB (each)		500 MB/sec		300 MB/sec
32-bit PHB (each)		250 MB/sec		150 MB/sec
PHB Total (6 PHBs)		2500 MB/sec		1500 MB/sec
RIO Hub	2000 MB/sec	1000 MB/sec	1100 MB/sec	800 MB/sec
GX Bus	3400 MB/sec	1700 MB/sec	2500 MB/sec	1250 MB/sec

2.3 The interconnect fabric

Cluster applications, many times, require fast communication between the nodes in the cluster. In some cases, the performance of the application depends on the performance of this communication.

Here we will show the next step in the evolution of the SP interconnect fabric: the SP Switch2 and the SP Switch2 PCI Attachment Adapter.

Note: For more information on the SP Switch and the SP Switch adapters, refer to Chapter 9 of the *RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280. For information on the performance characteristics of the SP Switch and the SP Switch adapters, see the *RS/6000 SP System Performance Tuning Update*, SG24-5340.

2.3.1 The SP Switch2

Because the SP Switch2 is an evolution of the SP Switch and High Performance Switch (HPS), it is fully compatible with applications written to the older switches. This switch supports Message Passing Interface (MPI), Low-level Application Programming Interface (LAPI), and Internet Protocol (IP).

This new switch is used to interconnect the new Cluster 1600 nodes:

- The pSeries 660 models 6M1, 6H1, and 6H0
- The pSeries 680 Model S85
- The RS/6000 7026 models H80 and M80
- The RS/6000 7017 models S80 and S7A
- The pSeries 690 and pSeries 670

SP Switch2 features

Like the other SP switches, the SP Switch2 uses eight (8x8) crossbar switch chips to route data through the system.

The SP Switch2 uses asynchronous clocking rather than the previous synchronous clocking. The switch chips now also maintain the Time of Day (TOD) control logic and the performance monitoring logic that handle hardware synchronization.

TOD clocking

Each switch planar has two redundant internal clocks. Each switch chip has its own internal clock operating at a unique frequency. The system software selects one of the chips to act as a primary TOD chip, which then propagates a TOD signal to all switch boards and adapters. To handle differences in phase and frequency of the transmitted TOD, each switch chip sends a ping character and counts the cycles until the return character comes back; the hardware then calculates the delay, determines the cable length, and adjusts the TOD for delays due to cable length and intermediary switches. Every 872 microseconds, the primary TOD chip sends a TOD signal to maintain synchronization.

The eight switch chips of the switch board can be connected to 32 interposer chips that provide the interface to external components. The SP Switch2 planar has 32 interposer slots that require one switch interposer card for each switch connection. If a port is not in use, the interposer card is not required, but a null card is required to maintain cooling air flow. The wrap plug is now used only for diagnostic purposes.

Each SP Switch2 interposer card has one component called the Self-Timed Interface (STI) interposer chip, which is capable of transmitting and receiving data at a full duplex rate of 1000 MB/sec (500 MB/sec per direction). The STI chip uses a proprietary STI link protocol and hardware for unidirectional, differential STI I/O. It has two interfaces, one for local communication and one for long distance communication (over the switch cable to the STI chip on the SP Switch2 adapter).

SP Switch2 reliability, availability, serviceability (RAS)

The new SP Switch2 subsystem incorporates design improvements for better reliability, availability, and serviceability (RAS). These improvements include:

- Switch management simplification
  - Power supply redundancy.
  - Hot-pluggable power supplies, supervisor card, interposer cards, and switch cables.
  - Code patches for the supervisor card and switch cards that can be installed online.
  - Each switch chip has its own oscillator, which is synchronized to all other oscillators. This eliminates the need for an external clock and the **`Eclock`** command.
- Switch subsystem fault tolerance
  - Error Correction Circuitry (ECC) has been incorporated into the design of the RAMBUS memory on the SP Switch2 adapter card.
  - Parity checking.
  - Cyclic Redundancy Checks (CRC) are used to detect link errors.
  - Nodes can be unfenced even during correctable transient errors.

- System diagnostic improvements

  A fault service daemon monitors failures in the SP Switch2. Administrative intervention in case of permanent adapter errors is no longer required, because the adapter automatically resets itself. Also, a new level of recoverable errors (that do not require adapter resets) have been defined. The SP Switch2 also employs bit-tunable thresholds on recoverable errors that allow for better recovery of non-critical, transient errors and fault isolation in case of critical, but transient, adapter errors.

  The SP Switch2 also has improved definitions for error classification types, error sources, and bit descriptions for system error logging. In addition, label improvements were added to recoverable errors like hardware, microcode, threshold, bad packet, transient, and service queue full.

- System problem determination

  Miswire detection has improved. The system calls out miswired links and cables, and gives visual aids to the field engineer solving these problems. Additional features were added to help resolve subsystem software problems:

  - A supervisor location register is used to alert the software of a miswire condition and helps to determine the miswire location(s).
  - The “Return on Same Route Option” provides a response even on miswire.
  - The “Port Received on Register” function specifies the switch port that received a particular service.
  - Sender hang detect logic determines when a sender is unable to make progress and re-times the link.

  It is also possible to test switch interposers, interposer connectors, and switch cables using the SP Switch2 hot-plug feature.

For more information on the SP Switch2, see the *RS/6000 SP: SP Switch2 Technology and Architecture* whitepaper, found at:

`http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch2.pdf`

2.3.2 The SP Switch2 PCI Attachment Adapter

The SP Switch2 PCI Attachment Adapter is a full-length PCI card that has a 64-bit interface with a 66 MHz maximum clock speed. This adapter has 16 MB of RAMBUS memory to buffer large blocks of data between the fabric and the system. The main components in the adapter are:

- A TBIC3 switch fabric controller.
- A 740 PowerPC microprocessor.

- Two STI chips (like in the interposer card of the switch). However, only one is available for general use. The other STI chip and associated port are reserved.

The raw peak performance for this adapter is 500 MB/sec unidirectional and 1000 MB/sec bidirectional. This represents the maximum rate at which data can be given to or taken from the switch by the node.

There are many factors that can affect the overall performance of the communication through the adapter and switch, so that peak performance is rarely achieved by an application.

One of the factors that can affect the adapter and switch performance is the local bus bandwidth where the adapter is located. In the pSeries 690, you have to take the PCI Host Bus (PHB) and the Remote I/O (RIO) bus bandwidths into account. For this reason, here are some recommendations about SP Switch2 PCI Attachment Adapter placements in the pSeries 690:

- The SP Switch2 PCI Attachment Adapter in the pSeries 690 takes up two slots because of its large heat-sink.
- This adapter can only go in slots 3 and 5 on the left side and/or slots 3 and 5 on the right side of the RIO drawer.
- There can be a maximum of eight SP Switch2 LPARs per pSeries 690.
- There can be up to two adapters per LPAR.
- There can be up to four adapters per I/O drawer. This means up to four switched LPARs per I/O drawer.
- The SP LAN adapter can share the same PCI-PCI bridge with the SP Switch2 PCI Attachment Adapter.

Figure 2-7 on page 31 shows an example using four SP Switch2 PCI Attachment Adapters and four SP LAN adapters in a four LPAR configuration with a single-plane SP Switch2.

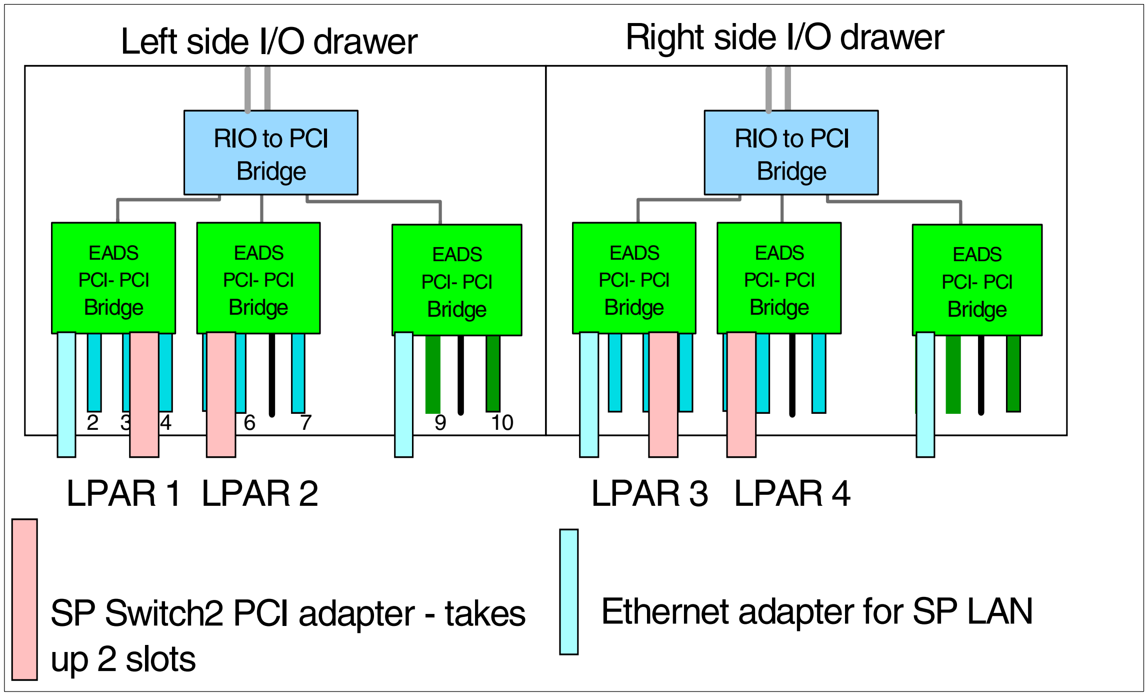


Figure 2-7 SP Switch2 PCI Attachment Adapter location for a single plane switch

Figure 2-8 on page 32 shows an example using four SP Switch2 PCI Attachment Adapters and two SP LAN adapters in a two LPAR configuration with a dual-plane SP Switch2.

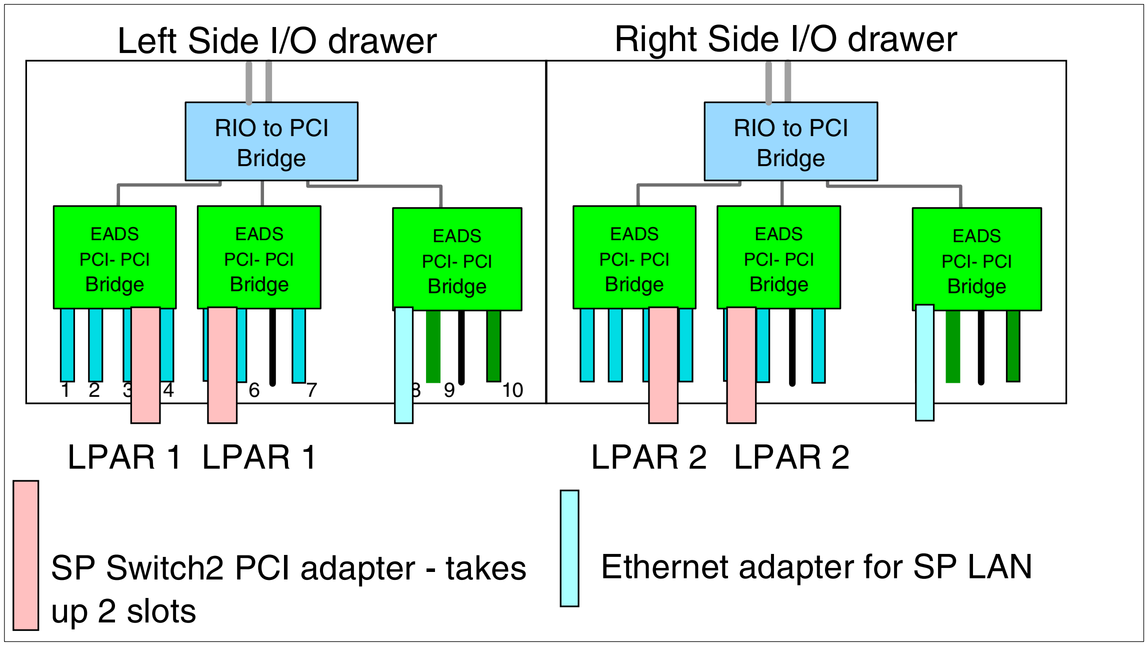


Figure 2-8 SP Switch2 PCI Attachment Adapter location for dual plane switch

For more information about the SP Switch2, see the *RS/6000 SP: SP Switch and SP Switch2 Performance* whitepaper at:

http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch_perf.html

2.3.3 Internet Protocol (IP) and User Space (US) windows

The IP family of protocols provide the underpinnings for the majority of networks around the world. IP is a robust protocol that supports multiple users and the reliable transfer of data. However, it requires a higher processor overhead compared with other protocols. This overhead also occurs on the PCI bus. For example, transmitting 64 KB messages with IP can incur a 60 KB overhead on the PCI bus.

In the Cluster 1600, Message Passing Interface (MPI) applications can use either Internet Protocol (IP) protocol or User Space (US) protocol to exchange messages across the switch. US is considered a *lightweight* protocol, because it requires fewer processor cycles than IP to transmit a given amount of data. This also applies to the PCI bus. For example, transmitting the same 64 KB messages from above with US only incurs a 1 KB overhead on the PCI bus.

User space communication is most commonly used in the scientific and technical arena, and IP (TCP/IP) socket communication is more commonly used in the commercial arena.

The advantage to using multiple threads in MPI tasks has been augmented with the SP Switch2. With the previous SP Switch, there was no bandwidth increase when changing an MPI task from one to two threads. With the SP Switch2, there is a bandwidth increase for up to four threads (at which point, the adapter bandwidth is reached); after that, there is no gain in bandwidth with additional threads.

With IP, higher bandwidth is achieved by using more than one CPU. However, multiple IP connections must be coupled with these multiple CPUs to achieve this higher bandwidth. If only one IP connection is available, the throughput will be similar to the single CPU throughput, due to the single-threaded nature of the memory-to-memory access in the IP stack.

2.4 Summary

This has been a very brief, conceptual overview of the new components that you can use in your Cluster 1600. If you would like more in-depth, low-level technical detail on the pSeries 690, see the *IBM @server pSeries 690 System Handbook*, SG24-7040.

Chapter 3, “Features relevant to performance” on page 35 is also largely conceptual in nature. It continues where this one left off and describes the features of these new Cluster 1600 components that are important to performance. The primary purpose of this chapter has been to build the foundation for the information contained in the next chapter.

3

Features relevant to performance

This chapter picks up where Chapter 2, “Hardware overview” on page 5 left off. We assume that you are familiar with the concepts covered there. If you are not, please go back and read that chapter before beginning on this one.

The focus of this chapter is on those features of the pSeries 690 and the new interconnect fabric that are relevant to performance. As with the previous chapter, this one is also primarily conceptual in nature. The topics covered are:

- Logical partitions (LPARs)
- Affinity logical partitions (ALPARs)
- Technical large page support
- Memory and scheduling affinity
- The 32-bit kernel versus the 64-bit kernel
- Compiler options
- Network connectivity (for example, SP Switch2, SP Switch2 PCI Attachment Adapter, and GigE)

3.1 Logical partitions (LPARs)

The pSeries 690 supports carving the system up into logical partitions (LPARs). Simply put, an LPAR is a way to divide the hardware resources of the system into one or more logical systems. Each LPAR is comprised of:

- Processors
- Memory
- I/O slots

The minimum configuration for an LPAR is one processor, 1 GB of memory, a boot device (I/O slot), and a network interface (I/O slot). Although not required, we recommend that you allocate a minimum of one half of an I/O drawer (either Bonnie or Clyde) to each LPAR. This provides one 10-slot PCI planar, two integrated SCSI controllers, and two 4-pack SCSI disk backplanes to each LPAR, and it helps to ensure balanced I/O bandwidth across all of the LPARs.

In a Cluster 1600 environment, the pSeries 690 appears as a frame, and each LPAR appears as a thin node (with a maximum of 16 allowed). Logical partitions are defined and managed from the Hardware Management Console (HMC).

Note: LPARs do not provide for dynamic load balancing. Once defined, they remain static entities.

3.1.1 The Hypervisor

The magic behind the LPARs on the pSeries 690 is provided by a piece of firmware known as the *Hypervisor*. The POWER4 microprocessor supports an enhanced form of system call known as hypervisor mode. This mode allows privileged program access to certain hardware facilities. The support also includes the protection of those facilities within the processor. Hypervisor mode allows the processor to access information about the system that is located outside of the boundaries of the LPAR in which the processor is located.

Another capability of the POWER4 microprocessor is the ability to include an address offset when using real-mode (non-virtual) memory addressing. This means that the operating system can use real-mode addresses to access low address locations, and the hardware can transparently relocate that access to any location in real memory. A bounding register is used to limit the range of real-mode addressing. The address offset support is required because the operating system expects real-mode memory to start at address zero, and there is only one physical address zero in the server. Therefore, the Hypervisor offsets the base address for each LPAR and translates what the LPAR sees as real-mode addresses to their physical counterparts in real memory.

Figure 3-1 is a logical view of where the Hypervisor sits in relation to the system software within an LPAR.

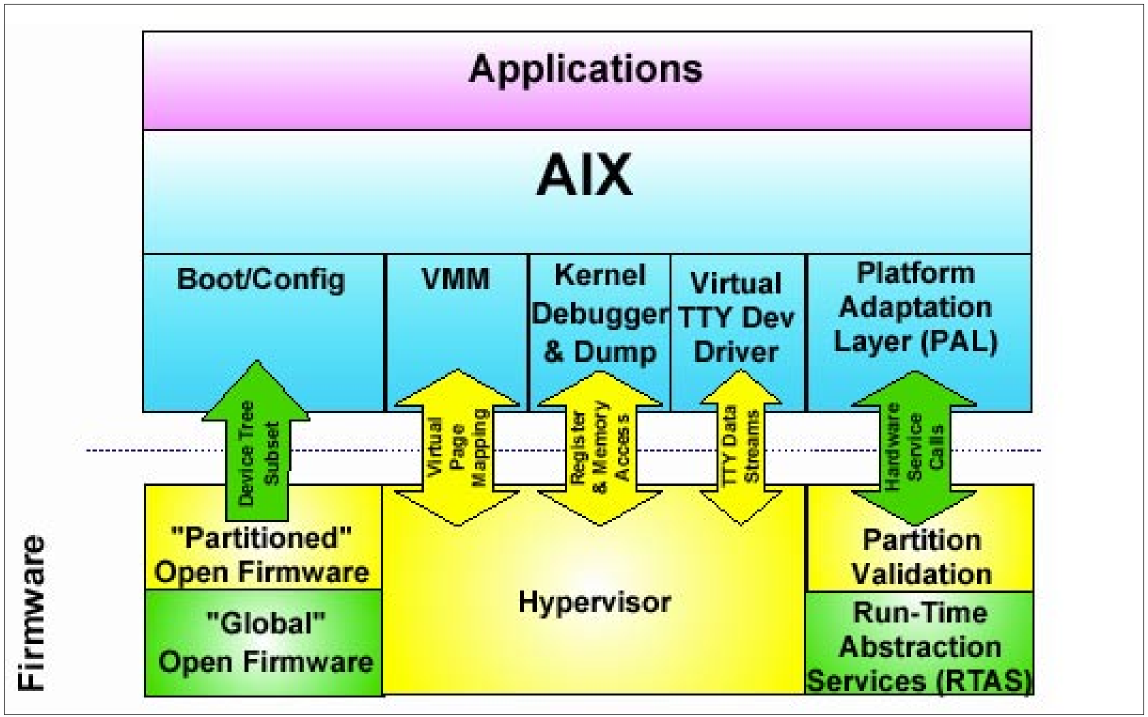


Figure 3-1 The POWER4 Hypervisor

The Hypervisor is a passive object that is loaded into the first 256 MB memory block of physical memory. It is only loaded into the environment; it does not reserve a processor resource for itself. The Hypervisor only runs when an LPAR needs a service executed on its behalf, such as creating a page table entry. The Hypervisor can be thought of as a call-back library for the LPAR. Care has been taken to minimize the number of instructions required to implement the call-backs, so that, in most cases, AIX performance is identical between a non-partitioned environment (where call-backs are not made) and a partitioned environment (where call-backs are required).

The Hypervisor implements the following three major categories of service calls:

- Virtual memory management
- Debug register and memory access
- Virtual tty support

From a performance and tuning point of view, there is nothing that you need to do (or, for that matter, can do) with the Hypervisor. However, it is important to understand it from a conceptual point of view. You will likely see it mentioned in other publications, and you may run across software (for example, debuggers, profilers, and so on) that are unable to work because of it. Prior to purchasing any software for the pSeries 690, you should verify with the vendor that the software has been certified to run on the POWER4 platform. This typically only applies to software that is operating at a very low level within the machine. Normal applications (as well as AIX itself) run inside an LPAR in the same way that they run on a stand-alone server or cluster node, and to the end users and system administrators, it all looks the same as well.

For more information on the Hypervisor (as well as the other firmware components), see the *IBM @server pSeries 690 System Handbook*, SG24-7040.

3.1.2 LPAR memory overhead

There is memory overhead associated with LPARs on a pSeries 690. It is important to understand how memory allocation is handled and how the memory overhead requirements are determined, because this ultimately determines the number and size of the LPARs that you can create for any given memory configuration.

Starting at the bottom of real memory, at address zero, the first 256 MB is occupied by the Hypervisor firmware. Starting at the top of real memory and extending downward, real memory is set aside for I/O and direct memory access (DMA) translation, with the amount determined by the number of I/O drawers:

- For one to four I/O drawers (80 I/O slots), 256 MB are used.
- For five or more I/O drawers (100 plus I/O slots), 512 MB are used.

A page table is created for every LPAR. The page table is the amount of contiguous memory equal to 1/64th of the memory in the partition, rounded up to the nearest power of two. For example, a 1.5 GB partition requires a page table of 24 MB, but rounding up to the next power of two takes it to 32 MB. Additionally, the page table must be aligned on a boundary that is an integer multiple of the rounded page table size. For example, a 32 MB page table must be aligned on a 32 MB, 64 MB, 96 MB, 128 MB, and so on, boundary.

The memory used by the LPAR requires some amount of real-mode memory. The hardware can allocate contiguous real-mode memory in quantities of 1 GB to 16 GB. These real-mode memory chunks must also be aligned on a boundary that is an integer multiple of the chunk itself (for example, 1 GB, 2 GB, 3 GB, and so on, or 16 GB, 32 GB, 48 GB, and so on). Allocation of memory to LPARs is as follows:

- For LPARs less than or equal to 16 GB, one 1 GB real-mode memory chunk is allocated, and the remaining balance is given in 256 MB logical memory blocks (LMBs).
- For LPARs greater than 16 GB, one 16 GB real-mode memory chunk is allocated, and the remaining balance is given in 256 LMBs.

Figure 3-2 illustrates the allocation of physical memory to these various components.

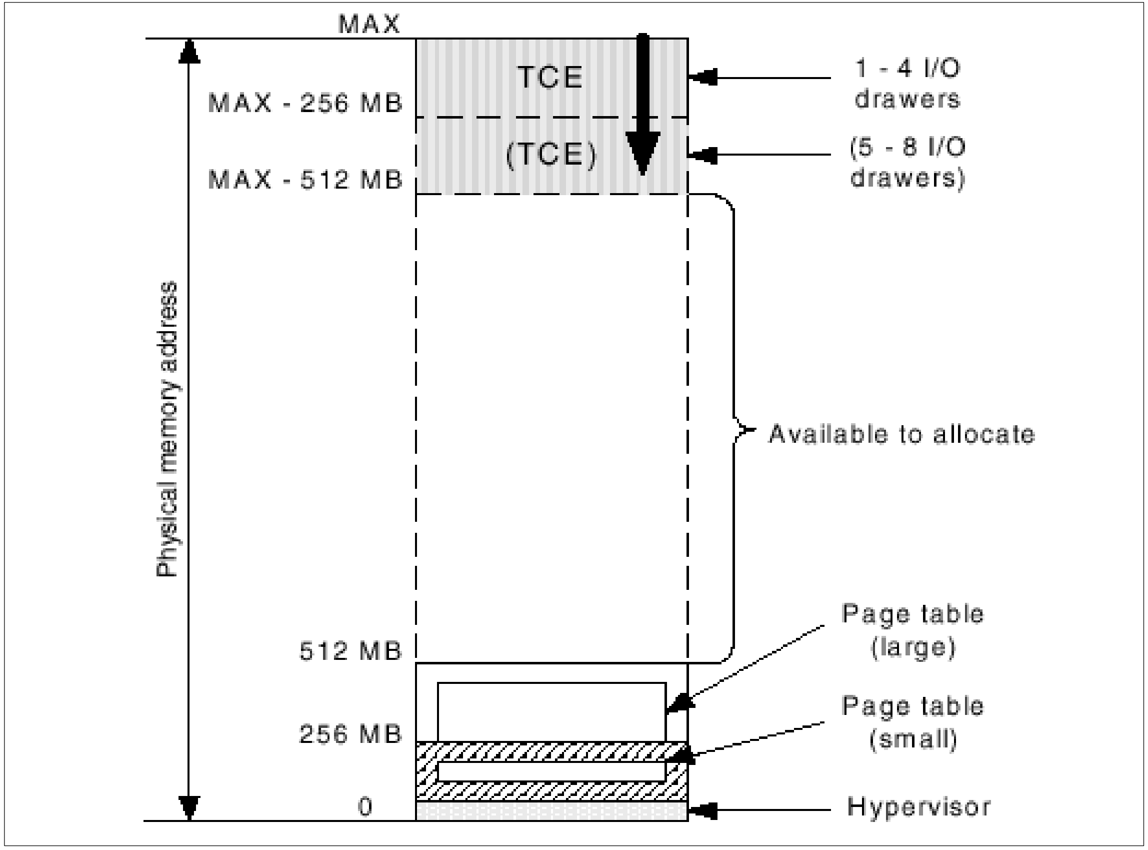


Figure 3-2 Reserved memory in an LPAR environment

For more information about the distinction between real-mode and logical-mode memory, see the Logical Partitions chapter in the *IBM @server pSeries 690 System Handbook*, SG24-7040.

Table 3-1 provides a summary of the memory overhead and LPAR sizing constraints for the pSeries 690.

Table 3-1 LPAR memory overhead

Total memory	Memory overhead (approximate)	Partitionable memory (approximate)	Maximum number of partitions (less than 16 GB / equal to 16 GB)
8 GB	0.75 GB to 1.00 GB	7.00 GB to 7.25 GB	6 / 0
16 GB	0.75 GB to 1.00 GB	15.00 GB to 15.25 GB	14 / 0
24 GB	1.00 GB to 1.25 GB	22.75 GB to 23.00 GB	16 / 0
32 GB	1.00 GB to 1.25 GB	30.75 GB to 31.00 GB	16 / 0
48 GB	1.25 GB to 1.75 GB	46.25 GB to 46.75 GB	16 / 1
64 GB	1.50 GB to 2.00 GB	62.00 GB to 62.50 GB	16 / 2
96 GB	2.00 GB to 2.50 GB	93.50 GB to 94.00 GB	16 / 4
128 GB	2.50 GB to 3.50 GB	124.50 GB to 125.5 GB	16 / 6
192 GB	3.50 GB to 4.50 GB	187.50 GB to 188.50 GB	16 / 10
256 GB	5.00 GB to 6.00 GB	250.00 GB to 251.00 GB	16 / 14

3.1.3 LPAR mode versus Full System Partition mode

The pSeries 690 can operate in two different modes: LPAR mode or Full System Partition mode. In Full System Partition mode, the pSeries 690 looks and behaves as a traditional stand-alone SMP server. In other words, there are no LPARs. Full System Partition mode is also referred to as symmetric multiprocessing (SMP) mode. From a theoretical performance standpoint, Full System Partition mode is better than LPAR mode, but from a practical performance standpoint, the difference between the two is insignificant. Here are the trade-offs:

- Memory overhead

  There is additional memory overhead associated with running a pSeries 690 in LPAR mode. This has already been covered.

- Paging performance

  An operating system running in an LPAR has slightly less page table management performance, as it must use the Hypervisor services for page table management. An operating system running in a Full System Partition has full use of all of the system memory and native virtual memory management performance. In a high-volume paging environment, system performance is slightly less with LPAR mode than it is with Full System Partition mode. (Of course, if the system has a high-volume of paging activity, paging performance is probably the least of your problems.) In normal paging environments, there is no observable difference in performance.

- 32-bit kernel limitation

  The 32-bit kernel is only able to address 96 GB of memory. If your pSeries 690 has more than 96 GB of memory, then LPAR mode is the only way for you to use the additional memory.

- Fast reboot

  Rebooting an operating system instance in an LPAR is much faster than rebooting that same instance in a Full System Partition. An LPAR reboot is merely a re-establishment of the pSeries Open Firmware OS boot loader environment. A Full System Partition reboot repeats all of the hardware initialization phases of the processors, caches, memory, I/O drawers, and I/O adapters. Because Full System Partition reboots go through all of the normal system initialization phases, they take roughly the same amount of time to reboot as other large SMP pSeries servers do.

- Additional SP Switch2 PCI Attachment Adapters

  Each LPAR can have two SP Switch2 PCI Attachment Adapters (on a dual-plane SP Switch2 system). So you can effectively increase the interconnect bandwidth off a pSeries 690 by configuring it with multiple LPARs as compared with leaving it in Full System Partition mode. (Note that there is a limit of 16 SP Switch2 PCI Attachment Adapters per pSeries 690.)

- Memory affinity

  Memory affinity is a feature that is only available in Full System Partition mode. With memory affinity turned on, AIX attempts to allocate memory from the memory books connected to the MCM where the process is running. This improves locality of reference for the running process, and is something that is especially important in the scientific and technical arena. Memory affinity is covered in detail in Section 3.4, “Memory affinity” on page 52.

Clearly, there are trade-offs between single LPAR mode and Full System Partition mode. With single LPAR mode, you gain administrative flexibility. With Full System Partition mode, you gain some performance. You also gain the ability to turn on memory affinity. So, given that you are not going to be configuring a pSeries 690 with multiple LPARs, which should you use, a single-LPAR or a Full System Partition? If you are in a scientific and technical environment where locality of reference is important, we recommend that you go with Full System Partition mode. For all others, we recommend that you go with single-LPAR mode over Full System Partition mode.

3.1.4 LPAR memory and processor allocation

When you allocate processors to an LPAR, you have no control over which processors get allocated. For example, if you allocate four processors to an LPAR, it is possible for each of those processors to be on a different MCM. The same holds true with memory. You have no control over how it is allocated or from where it is allocated. Access to memory not located on the same MCM as the processor exhibits some non-uniform memory access (NUMA) characteristics and can introduce variability into run times. For example, suppose you have two identically configured LPARs, each with four processors and 4 GB of memory. If the first LPAR has all of the processors and memory local to one MCM, and the second LPAR does not, then running the same code with the same input on the two LPARs will most likely result in slightly different run times. (The multi-MCM configuration provides a worst case memory access latency of slightly over 10 percent more than the best case memory access latency.) This is generally only an issue for high performance computing (HPC) codes in the technical and scientific arena. If this is an issue in your environment, then consider using Affinity LPARs (ALPARs) instead of traditional LPARs.

3.2 Affinity logical partitions (ALPARs)

An affinity logical partition (ALPAR) is a special type of logical partition (LPAR) with physical affinity to the MCM. ALPARs use system-defined resource divisions for memory and processors. These divisions are defined such that the memory and processors within an ALPAR remain in close proximity to each other, and this helps to maximize the performance of the ALPAR. Like traditional LPARs, ALPARs are created from the pSeries 690 Hardware Management Console (HMC).

There are only two supported types of ALPARs:

- Four processor (1/2 MCM)
- Eight processor (1 MCM)

This means that for a fully-loaded, 32-way pSeries 690, you have a choice of either four 8-way ALPARs or eight 4-way ALPARs.

Once you have picked one of these two, the system creates as many ALPARs of the selected type as possible, sets up the default profiles for each, and creates one system profile containing them all. You still have to select which I/O slots you want included with each, and the constraints for that are the same as with traditional LPARs.

There are also some new constraints with ALPARs that you need to consider:

- For any single pSeries 690, prior to configuring the system, you need to decide which way you are planning to go, either LPARs or ALPARs. Running LPARs and ALPARs together on the same pSeries 690 is not supported.
- To switch between 8-way ALPARs and 4-way ALPARs or vice versa requires deletion of the existing ALPARs and the addition of the new ALPARs. The 8-way ALPARs and the 4-way ALPARs cannot coexist on the same pSeries 690.
- For fully-loaded pSeries 690s, there is a 16 partition limit imposed by the HMC. (For partially loaded pSeries 690s, the limit may be lower.) It makes no difference how many partitions you plan to have active at a time; the HMC will not allow you to *define* more than 16. The defined partitions can be a mix of ALPARs (8-way or 4-way) and LPARs (even though only one or the other can be active at any given time). We recommend that you define the ALPARs before defining the LPARs.
- Each partition definition (LPAR or ALPAR) is assigned a *Partition ID* ranging from 001 to 016. In a stand-alone pSeries 690 environment, the Partition ID does not much matter. What does matter in that environment is the hard limit of 16 partition definitions. However, when you place a pSeries 690 into a Cluster 1600, the Partition ID becomes very important. It is what PSSP uses to identify the logical node contained within that LPAR or ALPAR. The implications of this are that you cannot have dual partition definitions for each partition. For example, suppose you were planning on having four partitions for a pSeries 690 in your Cluster 1600, and you wanted to set it up so that you could test the difference between LPARs and ALPARs. Suppose further that you create definitions for four ALPARs and definitions for four LPARs. In each ALPAR/LPAR pair, you include the exact same hardware so that the only difference is that one definition is for an ALPAR, and the other is for an LPAR. Now, you should be able to bring the system up either as all ALPARs or all LPARs, right? Wrong. The problem is the Partition ID. The ALPARs will have Partition IDs of 001-004, and the LPARs will have Partition IDs of 005-008. Therefore, the LPARs will fail to boot (LED E1F7) because PSSP does not recognize them. The moral of the story is that if you want to experiment between ALPARs and LPARs, you need to first delete the current set of definitions, then add the second. In other words, to switch between ALPARs and LPARs, delete the ALPAR definitions, then add the LPAR definitions using the same hardware information for each LPAR that you used for the corresponding ALPAR. This will bring the new set of partitions in with the same Partition IDs as the old ones, and PSSP will not be the wiser for it.
- Not all pSeries 690 configurations are equipped to use ALPARs. If you are thinking about using them, be sure to work with your IBM Sales Representative to make sure that the configuration that you order supports them.

ALPARs are targeted at applications that have very stringent performance requirements. These applications typically fall in the realm of technical, scientific, or business intelligence (BI). In certain environments (for example, a system with unbalanced memory books), using ALPARs rather than LPARs may actually result in a decrease in performance, rather than the other way around. Remember that you are trading configuration flexibility for potential performance gains. Be sure that what you are gaining more than makes up for what you are giving up.

Table 3-2 provides a summary comparison between LPARs and ALPARs. The numbers are based on a fully-loaded, 32-way pSeries 690.

Table 3-2 Comparison of LPARs to ALPARs

Description	**LPAR**	**ALPAR**
Number of partitions	0-16.	4-processor: 0-8. 8-processor: 0-4.
Memory	Defined by the user.	Configured by the system. Cannot be modified by the user.
I/O slots (devices)	Defined by the user.	Defined by the user.
Partition profile	Defined by the user.	Configured by the system. Some modification allowed by the user.
System profile	Defined by the user.	Configured by the system. Some modification allowed by the user.
Partition addition	User can add individual partitions.	It is all or nothing. User cannot add individual ones.
Partition deletion	User can delete individual partitions.	It is all or nothing. User cannot delete individual ones.
Partition activation	If no ALPARs are active, then the user can activate these, individually or in groups.	If no LPARs are active, then user can activate them, individually or in groups.

3.3 Technical large page support

The pSeries 690 supports two virtual memory page sizes:

- The 4 KB traditional page
- The 16 MB technical large page

The technical large page support is aimed primarily at the high performance computing (HPC) environments. (That is why it is called “technical large page” rather than just “large page”.) However, it may also be of value in commercial environments running things such as business intelligence (BI) applications.

Memory intensive applications that use large amounts of virtual memory may gain improvements in performance by switching to technical large page. The technical large page improvements are attributable to:

- Reduced translation lookaside buffer (TLB) misses due to the TLB being able to map a larger virtual memory range
- Improved memory prefetching by eliminating the need to restart the prefetch operations on 4 KB boundaries

More information about the details of how technical large pages effect TLB and hardware data prefetching behavior can be found in the *IBM* @server *POWER4 System Microarchitecture* whitepaper, available at:

http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

Technical large page support is available for both 32-bit and 64-bit applications, and both the 32-bit version and the 64-bit version of the AIX kernel support it. The 32-bit kernel and virtual memory management (VMM) architecture are used to illustrate the concepts covered here.

3.3.1 Technical large pages and the Virtual Memory Manager (VMM)

AIX processes access memory through segment-based addresses. A segment-based address is calculated using a segment register and a segment offset. In the 32-bit VMM environment, there are 16 segment registers. Each segment register can reference a 256 MB segment (for a total process address space of 4 GB). Some of the registers are used to address kernel memory while others can be used for multiple purposes. By default, Segment 2 holds the process private data, which contains, among other things, the stack and the heap. The stack starts at the top of the segment and grows downwards, and the heap starts at the bottom of the segment and grows upwards.

Each segment can be further classified by its storage type. There are two: working and persistent. The storage type indicates which medium is used to back the segments when they are moved from physical memory. Working segments are backed by paging space, and persistent segments are backed by disk files.

The 32-bit VMM environment is illustrated in Table 3-3 on page 46.

Table 3-3 The 32-bit VMM (user) process image

Segment number	Description	Storage type
0 (0x0)	Kernel	Working
01 (0x1)	Text (program code)	Persistent
02 (0x2)	Private data	Working
03 (0x3) through 12 (0xC)	Shared memory / memory mapped files - or - Additional data space	Persistent - or - Working
13 (0xD)	Shared library text	Working
14 (0xE)	shmat / mmap	Persistent or working
15 (0xF)	Shared library data	Working

For additional details on the inner workings of the VMM, see *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041.

The POWER4 technical large page architecture requires that all virtual pages in a 256 MB segment be the same size. AIX uses this architecture to support a mixed-mode process model where some segments are comprised of 4 KB pages and others are comprised of 16 MB pages. Applications can request technical large pages for two types of segments:

- Heap segment(s)
- Shared memory segment(s)

All other segments in the process are comprised of 4 KB pages.

The memory pools for 4 KB pages and 16 MB pages are separate. The 16 MB pool is specified through the **`vmtune`** command, and the physical memory for these pages is allocated at system boot time. Further changes to the allocation require a reboot. It is important to note that the memory allocated to the 16 MB technical large page pool is pinned, and there is no paging support for it. You can configure the technical large page pool to use up to 85 percent of the total usable memory of the system or LPAR, and the remaining memory is then allocated to the 4 KB page pool.

Process data contained in technical large pages remains in physical memory until the process completes. It is never paged out. Authorization to use technical large pages is granted at the user-level and controlled by a new security access control mechanism. This prevents unauthorized processes from using technical large pages and starving out the legitimate ones.

It is also important to understand that the 32-bit process model changes when you use technical large pages. The new 32-bit process model for technical large pages is very similar to the existing 32-bit large memory model. The existing large memory model is the one that is used when the executable was built with the -bmaxdata:0x########/dsa flag or modified with the **`/usr/bin/echo '\0200\0\0\0' | /bin/dd of=<executable_filename> bs=4 count=1 seek=19 conv=notrunc`** command sequence. The difference between the existing large memory model and the new technical large page model involves the location of the privately loaded library text. In the existing large memory model, privately loaded library text and data are placed in the heap. In the new technical large page model, privately loaded library text and data are placed in the low addresses of Segment 2. The 64-bit process model with large technical pages is the same as the 64-bit process model without large technical pages.

3.3.2 Technical large page usage

An authorized process can use technical large page in two ways:

- To back its data and heap segments
- To back its shared memory segments

The first can be accomplished without any code changes or recompilations. The second requires both a code change and a recompilation.

Technical large pages for data and heap segments

A process can request that its initialized program data, uninitialized program data (BSS - Block Started by Symbol), and heap segments be backed with technical large pages. There are two ways to accomplish this:

- The program executable file can be marked to request technical large pages.
- An environment variable can be set to control technical large page usage.

Technical large page use is established when the process is created by the exec system call, and technical large page use is inherited by child processes across a fork system call.

Marking the executable

The XCOFF (and XCOFF64) header in the executable file contains a new flag to indicate that the process wants to use technical large pages to back its data and heap segments. This flag can be set at link time by specifying the -blpdata option on the **`ldedit`** command. The flag can also be set and cleared after the executable has already been built. The commands to do this are:

- To set the flag:

  ```
  ldedit -blpdata <executable_filename>
  ```

- To clear the flag:

```
ldedit -bnolpdata <executable_filename>
```

Note also that the **`ldedit`** command can be used to set the maxdata value on an executable file.

Setting the environment variable

Authorized users can set a new environment variable to indicate that they want to use technical large pages for an application’s data and heap segments. Note that the environment variable takes precedence over the technical large page flag setting in the executable file. In other words, the environment variable setting can control whether or not the marked executable actually executes with technical large pages.

Technical large page usage is provided as an option within the LDR_CNTRL environment variable, and there are three settings of interest:

- Use technical large pages (advisory mode):

  LDR_CNTRL=LARGE_PAGE_DATA=Y
- Use technical large pages (mandatory mode):

  LDR_CNTRL=LARGE_PAGE_DATA=M
- Do not use technical large pages:

  LDR_CNTRL=LARGE_PAGE_DATA=N

Note also that the LDR_CNTRL environment variable can be used to set the maxdata value:

- To request technical large pages as well as maxdata of 2 GB:

  LDR_CNTRL=MAXDATA=0x80000000@LARGE_PAGE_DATA=Y

Important: Set these environment variables in scripts that are used to invoke the applications, rather than in your local dot startup files. In other words, set them at the application level and not at the login level.

Setting these environment variables at the global level turns technical large pages on globally. Any executable that is marked for technical large page usage will attempt to execute with them. This can produce undesirable effects leading to a significant loss in performance for the overall system. It is better to set and unset the environment variables on a case-by-case basis.

Advisory versus mandatory modes

A process can indicate that it wants to use technical large pages to back its data and heap segments in either *advisory mode* or *mandatory mode*.

In advisory mode, the process will be given technical large pages, if possible. The conditions for this to happen are:

- The user ID that owns the executable is authorized to use technical large pages.
- The POWER4 system has been configured with a technical large page pool.
- There are enough free pages in the technical large page pool to back the entire data or heap segment with technical large pages.

If all of these conditions are met, the data and heap segments of the process will be backed with technical large pages. Otherwise, the data and heap segments will be backed with normal 4 KB pages. Depending on free memory in the technical large page pool, it is possible for a process to have some of its data or heap segments backed by technical large pages and others backed by normal 4 KB pages. Remember that it is only a segment that has to be backed by homogeneous pages, not the entire memory footprint of the process.

In mandatory mode, it is all or nothing. If there are not enough free technical large pages in the technical large page pool to back all of the data and heap segments for the process, it is terminated. In mandatory mode, you must put special attention on monitoring the technical large page pool to ensure that it is sized appropriately for your workload. In mandatory mode, if it is not sized appropriately, applications will fail. In advisory mode, these same applications may at least be able still run, however slowly.

The primary disadvantage to using advisory mode is that it can introduce variability into the run times of the process. One run of the code may be fully backed by technical large pages while a subsequent run may have none. For this reason, we recommend that, if you are running in an environment where minimizing variability in run times is important, you use mandatory mode. Scientific and technical environments typically fall into this category.

Note: Executable files that have been marked to use technical large page support will do so in advisory mode. You must use the LDR_CNTRL=LARGE_PAGE_DATA=M environment variable setting to turn on mandatory mode.

Technical large pages for shared memory segments

In general, an application source code change is required in order to use technical large pages to back shared memory segments. Specifically, the SHM_LGPAGE and SHM_PIN flags have to be set on the shmget system call.

The exception to this is with shared memory MPI. If your application is using shared memory MPI and the LPAR and account are set up for technical large page, then the shared memory segments used by MPI will be backed by technical large pages.

The request to back shared memory segments with technical large pages is advisory. There is no way to make it mandatory. Technical large pages back shared memory segments under the same conditions that data and heap segments are backed in advisory mode. If the technical large pages are available, they will be used. If not, the normal 4 KB pages will be used. For this reason, we recommend that technical large pages not be used to back shared memory segments in environments (such as scientific and technical) where minimizing variability in run times is important.

Authorizing technical large page use

The technical large page pool is a fixed size, pinned memory resource. For this reason, it is important to control who has access to it. You grant this access to non-root users through the **chuser** command as follows:

```
chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE <userid>
```

Note that this grants technical large page usage for data and heap segments as well as for shared memory segments. There is no way to separate them with the **chuser** command. However, you can enable or disable the shared memory portion at the system or LPAR level via the **vmtune** command.

Configuring a pSeries 690 to use technical large pages

By default, a pSeries 690 has no memory configured for technical large page. To configure the technical large page pool, you use the **vmtune** command, and a system or LPAR reboot is required.

The **vmtune** executable is located in the /usr/samples/kernel directory. It is part of the bos.adt.samples fileset. There is also a 64-bit version called **vmtune64** that goes with the 64-bit kernel.

To allocate physical memory to the technical large page pool, run:

```
/usr/samples/kernel/vmtune -g <page_size> -L <page_count>
```

The **vmtune** command supports two values for page_size. They are 16777216 (16 MB) and 268435456 (256 MB). Note that even though the **vmtune** command supports the 256 MB page size, the POWER4 does not. The page_count is the number of page_size pages to allocate to the technical large page pool. You can allocate up to 85% of the available system memory to the technical large page pool.

For example, to set the technical large page pool size at 4 GB, run:

```
/usr/samples/kernel/vmtune -g 16777216 -L 256
```

Once you have issued the **vmtune** command, you then need to run the **bosboot** command and reboot the system or LPAR.

To enable technical large page support for shared memory segments, you must use the **vmtune -S 1** command to enable the SHM_PIN flag in the shmget system call. The SHM_PIN flag must be re-enabled after every system or LPAR reboot, so it is a good idea to add it to one of your system startup files, such as /etc/rc.local or /etc/inittab.

Technical large page command support

In addition to the **vmtune** command, several other system commands, such as **svmon**, have been extended to report on technical large page usage. In Chapter 4, "Investigations" on page 77, we illustrate the use of both **vmtune** and **svmon** in a technical large page environment.

Technical large page support considerations

If you are thinking about using technical large pages, here are some things to consider:

- Technical large page support is a special purpose performance improvement feature. It is not recommended for general use. The types of applications that can likely benefit from it are long running, memory intensive applications that use large amounts of virtual memory.
- Consider the overall system performance when deciding to use or not use technical large pages. While some specific applications may benefit from technical large page use, the overall performance of your system may suffer. Memory is pre-allocated to the technical large page pool and, therefore, is no longer available for normal applications. A system that never used to page may begin doing so because of the lack of available memory in the normal 4 KB page pool.
- Oversizing the technical large page pool may cause the system to crash. During system boot, AIX reserves enough physical memory for 4 KB pages to ensure that the system will boot, but system failures may occur after booting if there are not enough 4 KB pages available in the pool.
- There is no such thing as *reverse advisory mode*. In advisory mode, a process requesting technical large pages can use normal 4 KB pages if there are not enough technical large pages to back an entire segment. However, the opposite is not true. A normal application, user or system, cannot use technical large pages if there are not enough 4 KB pages available.
- Each change to the technical large page pool size requires a reboot of the system or LPAR.
- The mprotect system call cannot be used with technical large pages. Some of the debug tools that are used to find problems with memory leaking

applications (that is, applications that are having problems with the malloc system call) use mprotect to attempt to diagnose the problem. These tools will not work properly with technical large pages, which means that the problem application will need to be debugged running in a normal 4 KB page environment.

- Because technical large pages use pinned memory, they are treated as unmanaged resources by the Workload Manager (WLM).

3.4 Memory affinity

The POWER4-based IBM @server pSeries server has CPUs that are organized into multiple MCMs. Each MCM may or may not have memory associated with it. The memory attached to a single MCM has the same access characteristics for any location within it and is said to fall within a single affinity domain. A processor can access memory attached to its local MCM faster (that is, lower latency) than it can access memory attached to other MCMs.

AIX 5L has optional support for organizing its memory management strategies around these affinity domains.

With memory affinity support enabled, AIX attempts to satisfy page faults from the memory closest to the processor that generated the page fault. This is of benefit to the application because it is now accessing memory that is local to the MCM rather than memory scattered among different affinity domains.

Memory affinity is enabled at the operating system level. Changes are not required at the application level to take advantage of it.

3.4.1 Memory configuration of pSeries 690

IBM @server pSeries 690 systems support four memory controllers per MCM. The memory subsystem is implemented using memory cards where each card contains two memory controllers, synchronous memory interfaces (SMIs), and DIMMs. Each controller can support up to 16 DIMMs.

To maximize memory performance on pSeries 690 systems, memory interleaving is employed. Memory is interleaved across controllers. Interleaving addresses is a function of the L3 cache controllers and the L3 cache to which the memory controllers are attached. It is implemented by the L3 cache controller on the POWER4 chip.

Figure 3-3 on page 53 illustrates the relationship between MCMs and memory subsystems from a logical point of view.

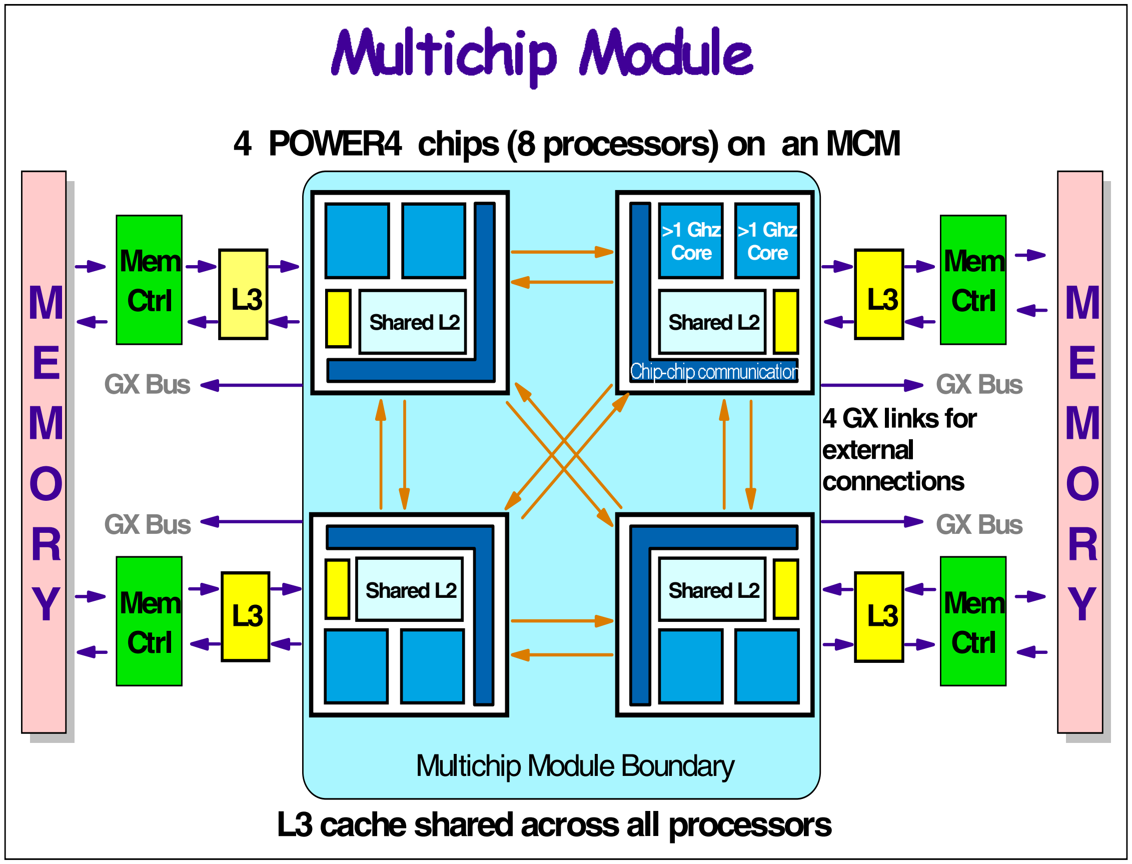

Multichip Module
4 POWER4 chips (8 processors) on an MCM
MEMORY
Mem Ctrl
L3
>1 Ghz Core
>1 Ghz Core
Shared L2
Chip-chip communication
GX Bus
4 GX links for external connections
Multichip Module Boundary
L3 cache shared across all processors

Figure 3-3 Logical view of MCM and memory

Memory configurations with performance consideration

- If an MCM has two memory cards of the same size installed, memory is interleaved in a round-robin fashion across the four memory controllers with 512-byte granularity. Referring to Figure 3-3, memory is interleaved in the following way:
  - The first 512-byte block of memory is in the memory card on the left of the MCM and is accessed by the L3 in the lower left corner of the MCM.
  - The second 512-byte block is accessed by the L3 in the upper left corner.
  - The third 512-byte block is accessed from the memory card on the right side of the MCM by the L3 in the upper right corner.
  - The fourth 512-byte block is accessed by the L3 in the lower right corner.

This continues throughout the memory range afforded by the two cards, and each L3 handles only one-fourth of the memory addresses for that MCM.

- If an MCM has one memory card attached (for example, in Figure 3-3 on page 53, if the memory card on the left is the only one present), then the memory is interleaved only on the single memory card by the two L3s on the left side of the MCM. The command queues associated with these two L3s must then process twice the traffic they would in the first case for the same application, and this reduces the available bandwidth for the MCM.
- If an MCM has two memory cards of different sizes attached, then the two cards are treated independently, with each card being two-way interleaved. For example, if an 8 GB and a 32 GB card has been installed in the memory slots of the MCMs, eighty percent of the data for an application will be handled by two of the L3s and twenty percent of it will be handled by the other two L3s. This reduces the effective bandwidth, as a result of uneven use of the L3 command queues.
- Memory size can also impact system performance. The 4 GB and 8 GB memory cards have one port between each memory controller and memory, whereas 16 GB and 32 GB cards have two ports between each memory controller and memory. Hence, the latter support a greater bandwidth.

For more detailed information about the memory subsystem of the pSeries 690, refer to the *IBM @server pSeries 690 System Handbook*, SG24-7040 and *The POWER4 Processor Introduction and Tuning Guide,* SG24-7041.

3.4.2 Enabling memory affinity

Memory affinity is a special purpose option for improving performance on multiple chip module (MCM) systems. It is intended to offer performance improvements to selected high performance computing applications. Therefore, it is neither recommended nor beneficial for general use.

Memory affinity is enabled through the -y flag of the **vmtune** command. The syntax for the flag is -y 0|1. A value of 1 turns on memory affinity, and a value of 0 turns it off. The default value of memory affinity is off. The **bosboot** and **shutdown -Fr** commands are required for the change to take effect.

If memory affinity is enabled, a memory pool is constructed for each affinity domain reported by the firmware at system boot time. The system will attempt to satisfy page faults from the memory pool closest to the processor that faulted.

Notes: Memory affinity is a performance tuning option only applicable to pSeries 690 and 670 systems. On platforms that do not support memory affinity, turning it on will have no effect.

Memory affinity is only available in Full System Partition (that is, SMP) mode. Turning it on in LPAR mode has no effect.

Memory affinity support is provided with AIX 5L Version 5.1, Recommended Maintenance Package 5100-02 or higher. You can use the `oslevel -r` command to determine your current AIX maintenance level.

3.4.3 Performance considerations for memory affinity

Enabling memory affinity may not always result in a performance benefit. This is because there is nothing preventing the AIX dispatcher from dispatching a thread to a different MCM for each time slice.

When memory affinity is enabled, the memory local to the MCM is allocated. So, the memory that the thread has allocated will have the lowest latency available at the time the thread requested it.

A problem arises if the thread is moved between MCMs. When a thread is moved from the MCM, the allocated memory will no longer be the lowest latency, and, in fact, will have higher latency. In other words, all of the previously allocated memory will now be remote to the MCM and will therefore have higher latency.

When memory affinity is turned off, memory will be allocated randomly. Therefore, some memory will be local to the MCM and other memory will be remote to the MCM. So, as the thread moves between MCMs, its memory latency remains a mixture of high and low with the result tending toward the average.

Thus, to obtain beneficial performance results that are consistent across multiple runs of an application, it is advisable to bind all the threads of an application to a single MCM. This can be done using the following methods:

- Through the `bindprocessor` command
- Through the bindprocessor application program interface (API)
- Through the AIX Workload Manager (WLM) by creating a class with resource sets and assigning the application to this class
- Through affinity logical partitions (ALPARs)

It is also possible to obtain a benefit with threads bound to different MCMs, but the application should be carefully written to avoid remote references. In particular, the memory accessed by individual threads should not overlap, and each thread should first reference its data area only after it has been bound to an MCM.

Some practical examples of when memory affinity will prove disadvantageous are the following. (They basically come down to not having all cooperating elements of an application running on the same MCM.)

- The application has one thread initializing its data area and other threads bound to other MCMs processing it. The data area will be faulted in on the MCM of the initial thread. The other threads will have remote references.
- The application uses technical large page segments and has threads bound to different MCMs accessing them. Technical large page segments are backed when the segment is created rather than when the virtual memory is referenced, so all the memory will come from the MCM on which the technical large page segments were created.
- The application uses shared memory and all the cooperating processes do not run on the same MCM. Unless the processes only references areas of shared memory that are non-overlapping, they will get remote references for some of their memory accesses.

3.4.4 Memory affinity with technical large page support

Memory affinity is supported on both 4 KB pages and 16 MB technical large pages. An application using technical large pages will get them preferentially allocated from the MCM where the application is running, if memory affinity is enabled. (Assuming of course, that there are technical large pages available in the memory pool associated with the MCM.)

One difference between 4 KB normal pages and 16 MB technical large pages has to do with when physical memory is allocated to back virtual memory. With 4 KB pages, physical memory is allocated when the segment containing the virtual memory is *referenced*. With 16 MB pages, physical memory is allocated when the segment containing the virtual memory is *created*. This can affect the performance gain or loss seen by an application when using memory affinity.

3.5 The 32-bit kernel versus the 64-bit kernel

The pSeries 690 running AIX 5L supports two different kernels: 32-bit and 64-bit. Features of the 64-bit kernel include:

- The 64-bit kernel supports both 32-bit applications and 64-bit applications, as does the 32-bit kernel.

- The 64-bit kernel maintains binary compatibility for 32-bit applications running on earlier versions of AIX on POWER-based systems. However, the 64-bit kernel maintains only source compatibility for 64-bit applications from earlier systems. These applications will have to be recompiled on the pSeries 690. This is due to the fact that the 64-bit application binary interface (ABI) was changed to make it more scalable, and, as a result, it is no longer compatible with the earlier 64-bit ABI. Additionally, existing 32-bit kernel extensions and device drivers used by 64-bit applications may have to be modified to support the new 64-bit ABI.
- The 64-bit kernel supports the increased size of virtual memory manager (VMM) data structures needed for larger memory configurations. The 32-bit kernel can address a maximum of 96 GB of physical memory. The 64-bit kernel has no such limitation.
- The 64-bit kernel provides the facility to scale kernel data types to more easily support greater than 32-bit addressability in the areas of 64-bit user address space, large files, number of inodes, device numbering, thread IDs, and so on.
- The 64-bit kernel supports the increased size and number of data structures in the global kernel address space that are required to support the possibility of thousands of physical and logical devices and their device drivers.

It is also very important to understand the difference between 64-bit kernel and 64-bit architecture. They are not synonymous. The 64-bit architecture is supported by the 32-bit kernel as well as the 64-bit kernel. A somewhat dated but still excellent whitepaper titled *The RS/6000 64-bit Solution*, which explains the 64-bit architecture, can be found at:

`http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/64bit6.html`

Your application mix will likely determine your choice of kernel. All of your applications and system software must support the 64-bit kernel prior to that becoming a viable option for you. However, even if the 64-bit kernel is not now a valid choice for your environment, it will likely be over time and, when it is, how will you decide which kernel to use?

Suppose you have a mix of 32-bit and 64-bit applications, but all are supported with either kernel. In general, if the main application or applications are 64-bit, then it is slightly better to use the 64-bit kernel. The same holds for 32-bit applications with the 32-bit kernel. However, the overhead of running 64-bit applications on the 32-bit kernel is handled by the kernel, and it is small. (The kernel has to remap the system calls to 32-bit calls and reshape the data structures for these calls.) The same holds true for 32-bit applications on the 64-bit kernel. So, given that your application mix will work on either kernel, our recommendation is to go with the 64-bit kernel.

Note: We recommend that you install your system with the 32-bit kernel, then switch to the 64-bit kernel. The reason for this recommendation is that if you install your system with the 64-bit kernel, you will also have to create rootvg as a JFS2 file system. If you install with the 32-bit kernel, then convert to the 64-bit kernel, rootvg can stay as JFS.

3.5.1 Selecting the 64-bit kernel

If your system has 64-bit processors, the 64-bit kernel is automatically installed, but not necessarily activated, with the base operating system. Activation of the 64-bit kernel during system installation is enabled by setting the Enable 64-bit Kernel and JFS2 option to yes at the start of the install process. Again, we recommend that you do not do this.

Enabling the 64-bit kernel

To switch from the 32-bit kernel to the 64-bit kernel after the system has been installed, do the following as root:

1. Run **`ln -fs /usr/lib/boot/unix_64 /unix`**.
2. Run **`ln -fs /usr/lib/boot/unix_64 /usr/lib/boot/unix`**.
3. Run **`bosboot -ad/dev/ipldevice`**.
4. Run **`shutdown -Fr`**.

Enabling the 32-bit kernel

To switch back to the 32-bit kernel, do the following as root:

1. Run **`ln -fs /usr/lib/boot/unix_mp /unix`**.
2. Run **`ln -fs /usr/lib/boot/unix_mp /usr/lib/boot/unix`**.
3. Run **`bosboot -ad/dev/ipldevice`**.
4. Run **`shutdown -Fr`**.

3.5.2 The 64-bit application environment

Enabling the 64-bit kernel is not the same thing as enabling the 64-bit application environment. The 64-bit application environment is what allows you to run 64-bit applications with the 32-bit kernel. It is enabled by running **`smitty`** and selecting **System Environments** -> **Enable 64-bit Application Environment**.

To determine if the 64-bit application environment has been enabled on your system, run:

```
[bd0101en][/]> grep load /etc/inittab
load64bit:2:wait:/etc/methods/cfg64 >/dev/console 2>&1 # Enable 64-bit execs
[bd0101en][/]>
```

If you see the load64bit line, then the 64-bit environment has been enabled.

Important: To enable the 64-bit application environment, use **`smitty`**. Do not manually add the `load64bit` line to /etc/inittab. It will not work.

By default, the 64-bit application environment is enabled on the pSeries 690.

3.6 Application performance tuning

While the focus of this book is tuning at the cluster level, the ultimate goal is to help you achieve better application performance. Consequently, some words concerning tuning at the application level are appropriate. This section is simply a reminder to consider application tuning as a means to improve performance. It gives you a brief overview of selected areas that you may wish to investigate.

The areas are:

- Application tuning guidelines and resources
- Compiler considerations
- Engineering and Scientific Subroutine Libraries
- The Mathematical Acceleration Subsystem (MASS) library
- Hostfile considerations for MPI performance
- Some final recommendations

3.6.1 Application tuning guidelines and resources

A detailed discussion of application coding practices for best performance on the pSeries 690 POWER4 architecture is beyond the scope of this book. Thankfully, this topic is covered in detail in guides, manuals, Redbooks, whitepapers, and Redpapers that are available on the Web. Some of these are listed below in Table 3-4 on page 60. Older references targeted at POWER3-based systems are also included, as these references still contain much relevant information on application tuning, including a wealth of examples and case studies.

Table 3-4 Selected application performance and tuning publications

Description	**Redpaper, redbook, whitepaper, or manual**
How to tune applications for the POWER4 processor.	*The POWER4 Processor Introduction and Tuning Guide*, SG24-7041 (redbook).
An older book, but an excellent source of tuning methodology.	*AIX Version 4, Optimization and Tuning Guide for Fortran, C, and C++*, SC09-1705 (guide)
Application tuning for the SP. Includes distributed memory, shared memory, and hybrid environments.	*Scientific Applications in RS/6000 SP Environments*, SG24-5611 (redbook)
Initial guide for application tuning on the POWER3 architecture.	*RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155 (redbook)
The XL Fortran User's Guide gives complete descriptions of XL Fortran compiler options.	*XL Fortran for AIX User's Guide Version 7.1*, SC09-2866
Introductory manual to VisualAge C++	*VisualAge C++, Version 5.0 Getting Started*
Contains both an overview of pSeries 690 hardware as it compares to previous POWER architectures and pSeries 690 performance measurements using the Gaussian computational chemistry code.	*Some Practical Suggestions for Performing Gaussian Benchmarks on a pSeries 690 System*, REDP0424 (Redpaper)
NCBI BLAST is one of the most widely used sequence alignment and database search applications in the area of BioInformatics. BLAST performance on the pSeries 690 is investigated.	*Some Practical Suggestions for Performing NCBI BLAST Benchmarks on a pSeries 690 System*, REDP0437 (Redpaper)
Detailed information on many tools useful in measuring system and application performance.	*AIX 5L Performance Tools Handbook*, SG24-6039 (redbook).

3.6.2 Compiler considerations

Where source code is available, significant performance gains can be realized by choosing the correct compiler options so that more optimal code is generated by the compiler. However, determining the correct combination of compiler options and their values can be a time consuming task.

Important: We *strongly* encourage you to follow these guidelines:

1. Be patient and willing to experiment.
2. Always check for correctness of results. Fast does not count for much if the results are wrong.
3. Read the compiler manuals to understand compiler option usage and resulting actions. Also refer to the Redbooks on POWER3 and POWER4 performance (noted in Table 3-4 on page 60) to gain a greater understanding of tuning for these platforms and view specific examples that may help your efforts.
4. This may seem obvious, but it is often overlooked: Talk with your colleagues. They may already have figured it out.
5. Iterate on Guideline 1.

Here is a short list of the more important compiler options to consider in trying to improve application performance. The list is by no means exhaustive; we encourage you to refer to the product manuals for XL Fortran and VisualAge C/C++ (see Table 3-4 on page 60). The options listed in Table 3-5 apply to both Fortran and VisualAge C/C++ unless noted otherwise.

Table 3-5 Important compiler options

Compiler option	**Description**
-O[level]	Specifies the level of optimization (if any) to be used during compilation. Levels range from 2 to 5 with higher values providing more aggressive optimization or automatic choices for other performance related compiler options. Currently, specifying -O is equivalent to specifying -O2. If optimization is not specified via -O, the default is to do *no* optimization.
-qarch	This option controls the instructions the compiler generates. Performance on a given architecture can be improved by targeting that architecture (For example: -qarch=pwr3 or -qarch=pwr4), but the resulting executable might only be able to run on specific machines. Note that executables created with -qarch=pwr3 may run on the POWER4 architecture of the pSeries 690. Choosing -qarch=auto causes the compiler to generate instructions specific to the machine on which the compilation is performed. So, specifying -qarch=auto on a pSeries 690 is equivalent to specifying -qarch=pwr4.

Compiler option	Description
-qtune	Tunes instruction selection, scheduling, and other implementation-dependent performance enhancements for a specific implementation of a hardware architecture. Note that while -qtune may affect the performance of a resulting executable on a given architecture implementation, it does not require the executable to be run only on the target architecture implementation. This is different from the results of some choices for -qarch.
-qstrict -qstrict_induction	By default, optimizations done by -O3, -qhot, and -qipa may rearrange code so that results or exceptions are different from those of unoptimized programs. The -qstrict flag ensures that -O3, -qhot, and -qipa do not alter the semantics of a program. The -qstrict_induction flag takes similar action, preventing the compiler from performing induction (loop counter) variable optimizations. Note that -qstrict and -qstrict_induction can cause reduced performance.
-qalign	Specifies the alignment of data objects in storage, which avoids performance problems due to misaligned data. Only aligns on 4 KB boundaries. Does not work with Fortran common blocks. It is more of an I/O optimization, but it can, in certain cases, be used for CPU optimization.
-Q	Controls whether procedures are to be inlined. Options allow a list of procedures to be named for inlining and identifying procedures that are *not* to be inlined. For VAC/C++, the -qinline option is used instead of -Q.
-qhot	Determines whether to perform high-order transformations on loops and array language during optimization.
-qipa	Enhances -O optimization by doing detailed analysis across procedures: interprocedural analysis or IPA. This option needs to be specified on the linker step as well.
-qmaxmem	Determines the amount of memory that the compiler can use while performing certain optimization. Setting -qmaxmem=-1 allows the compiler to use all the memory it needs without checking for limits. The default choices for -qmaxmem change with optimization levels, at -O3 the default setting for -qmaxmem is -1.

Additional information about the XL Fortran compiler can be found at:

http://www.ibm.com/software/ad/fortran/xlfortran/

Additional information about the VisualAge C/C++ compilers can be found at:

http://www.ibm.com/software/ad/vacpp/

3.6.3 Engineering and Scientific Subroutine Libraries

The Engineering and Scientific Subroutine Libraries (ESSL) family of products is a state-of-the-art collection of mathematical subroutines. (Parallel ESSL is the parallel version of ESSL.) Running on pSeries servers, RS/6000 workstations, RS/6000 servers, and SP systems, the ESSL family provides a wide range of high-performance mathematical functions for a variety of scientific and engineering applications.

The ESSL family includes:

- ESSL for AIX, which contains over 400 high-performance mathematical subroutines tuned for IBM UNIX hardware
- Parallel ESSL for AIX, which contains over 100 high-performance mathematical subroutines specifically designed to exploit the full power of RS/6000 SP and Cluster 1600 hardware with scalability of up to 512 nodes

ESSL provides a variety of mathematical functions, such as:

- Basic Linear Algebra Subprograms (BLAS)
- Linear Algebraic Equations
- Eigensystem Analysis
- Fourier Transforms

Both ESSL and Parallel ESSL have SMP-parallel capabilities. (The term *Parallel* in the Parallel ESSL product name refers specifically to the use of MPI message passing, usually across the Cluster 1600 interconnect). For SMP-parallel use within a stand-alone pSeries 690, Parallel ESSL is not required.

ESSL products are compatible with public domain subroutine libraries, such as Basic Linear Algebra Subprograms (BLAS), Scalable Linear Algebra Package (ScaLAPACK), and Parallel Basic Linear Algebra Subprograms (PBLAS). Thus, migrating applications to ESSL or Parallel ESSL is straightforward.

Additional information about ESSL and Parallel ESSL can be found at:

http://www.ibm.com/servers/eserver/pseries/library/sp_books/essl.html

3.6.4 The Mathematical Acceleration Subsystem (MASS) library

The Mathematical Acceleration Subsystem (MASS) library provides high-performance versions of a selected subset of Fortran intrinsic functions.

These versions sacrifice a small amount of accuracy to allow for faster execution. Compared to the standard mathematical library, libm.a, the MASS library differs, at most, only in the last bit. Thus, MASS results are sufficiently accurate in all but the most stringent conditions.

There are two basic types of function available for each operation:

- A single instance function
- A vector function

The *single instance function* simply replaces the libm.a call with a MASS library call. The *vector function* is used to produce a vector of results given a vector operand. The vector MASS functions may require coding changes, while the single instance functions do not.

Additional information about MASS, including performance measurements on POWER4 and POWER3 processors, can be found at:

`http://techsupport.services.ibm.com/server/mass?fetch=home.html`

The MASS software is also available for download from this site.

3.6.5 Hostfile considerations for MPI performance

Right about now you are probably thinking, “How can hostfiles affect performance?” Well, you will see that, in some cases, a beforehand knowledge of the communication patterns of your application combined with a careful reordering of the hosts in the hostfile can lead to improved performance.

For MPI parallel jobs, an application uses the hostfile to determine which hosts and processors the application uses to run its MPI tasks. The hostfile maps MPI tasks onto processors.

For example, assume an application is to be run on a 32-way pSeries 690 that has been partitioned into four, switch-attached, 8-way LPARs, lpar01 through lpar04. A four-way MPI parallel job to be run using one processor per LPAR would have this hostfile (mapped tasks are shown in parentheses and are not part of the hostfile):

```
lpar01    (task0)
lpar02    (task1)
lpar03    (task3)
lpar04    (task4)
```

By contrast, if the four-way job were to be run within a single LPAR (usually the best choice for performance), this would be the hostfile:

```
lpar01    (task0)
```

```
lpar01   (task1)
lpar01   (task2)
lpar01   (task3)
```

Four processors on the same LPAR would be used, and the application would run the job in *shared-memory MPI* mode.

In order to use processors both within and across LPARs for an eight-way job, this would be the hostfile:

```
lpar01   (task0)
lpar01   (task1)
lpar02   (task2)
lpar02   (task3)
lpar03   (task4)
lpar03   (task5)
lpar04   (task6)
lpar04   (task7)
```

Now consider a larger problem, an 8x8 process mesh to be run across two 32-way pSeries 690s. Each pSeries 690 has been partitioned into four, switch-attached, 8-way LPARs. Each element of the mesh will be assigned to a processor, meaning that a 64-way MPI parallel run will be made.

Example 3-1 shows the 64-element grid, assuming that communication between tasks is *nearest neighbor*. That is, communication follows the numerical order of the tasks: 0, 1, 2, 3, and so on. Thus, the bulk of the communication occurs across rows, as indicated by the horizontal lines in the example.

Example 3-1   8x8 process grid with nearest neighbor communication

```
56 --- 57 --- 58 --- 59 --- 60 --- 61 --- 62 --- 63    lpar08

48 --- 49 --- 50 --- 51 --- 52 --- 53 --- 54 --- 55    lpar07

40 --- 41 --- 42 --- 43 --- 44 --- 45 --- 46 --- 47    lpar06

32 --- 33 --- 34 --- 35 --- 36 --- 37 --- 38 --- 39    lpar05

24 --- 25 --- 26 --- 27 --- 28 --- 29 --- 20 --- 31    lpar04

16 --- 17 --- 18 --- 19 --- 20 --- 21 --- 22 --- 23    lpar03

 8 ---  9 --- 10 --- 11 --- 12 --- 13 --- 14 --- 15    lpar02

 0 ---  1 ---  2 ---  3 ---  4 ---  5 ---  6 ---  7    lpar01
```

Note: Communication does occur between tasks 7 & 8, 15 & 16, and so on.

Best performance is usually obtained by communicating within an LPAR as opposed to communicating across LPARs (that is, within a shared memory partition rather than across partitions via the interconnect). Consequently, it would be best to map MPI tasks 0 through 7 on one LPAR, 8 through 15 on a second, and so on. The numbering of the MPI tasks follows the numbering of processors: each row is contained in a single LPAR. This is shown in Example 3-1 on page 65 with the LPARs listed at the right side of the rows. The hostfile for this job assigns processors in numeric order and looks much like what you would expect:

```
lpar01   (task0)
    |  {6 entries of lpar01, (task1 through task6)}
lpar01   (task7)
lpar02   (task8)
    | {6 entries of lpar02, (task9 through task14)}
    |
lpar02   (task15)
{8 entries each for lpar03 through lpar08, (task16 through task63)}
```

Next, consider the same grid assuming a different communication pattern. Example 3-2 shows the case where communication occurs along columns of tasks rather than rows.

Example 3-2 8x8 process grid with columnar communication pattern

```
   56     57     58     59     60     61     62     63    lpar08
    |      |      |      |      |      |      |      |
   48     49     50     51     52     53     54     55    lpar07
    |      |      |      |      |      |      |      |
   40     41     42     43     44     45     46     47    lpar06
    |      |      |      |      |      |      |      |
   32     33     34     35     36     37     38     39    lpar05
    |      |      |      |      |      |      |      |
   24     25     26     27     28     29     20     31    lpar04
    |      |      |      |      |      |      |      |
   16     17     18     19     20     21     22     23    lpar03
    |      |      |      |      |      |      |      |
    8      9     10     11     12     13     14     15    lpar02
    |      |      |      |      |      |      |      |
    0      1      2      3      4      5      6      7    lpar01
```

Note: Communication does occur between tasks 56 & 1, 57 & 2, and so on.

If the hostfile from Example 3-1 on page 65 was used in this case, all communication between tasks would occur across LPARs over the switch rather than within a given LPAR. This ordering of processors in the hostfile would not take advantage of shared-memory MPI communication within an LPAR and performance would probably be less than optimal.

For this example, a reordering of the processor list in the hostfile would help to map tasks that communicate on the same LPAR. Since communication follows along tasks 0, 8, 16, 24, 32, 40, 48, and 56, it would be best to place these tasks on the same LPAR. Applying the same logic for the rest of the tasks results in this preferred ordering of the hostfile:

```
lpar01   (task 0)
lpar02   (task 1)
lpar03   (task 2)
lpar04   (task 3)
lpar05   (task 4)
lpar06   (task 5)
lpar07   (task 6)
lpar08   (task 7)
lpar01   (task 9)
    (repeat sequence as necessary)
lpar01   (task 56)
lpar02   (task 57)
lpar03   (task 58)
lpar04   (task 59)
lpar05   (task 60)
lpar06   (task 61)
lpar07   (task 62)
lpar08   (task 63)
```

The effect of the reordered hostfile on communication within and across LPARs is shown in Example 3-3. The situation is the reverse of Example 3-2 on page 66. Now, most communication occurs within an LPAR, taking full advantage of the *hybrid*, distributed, SMP architecture that the creation of LPARs allows.

Example 3-3 8x8 process grid with remapping of tasks

```
    7 --- 15 --- 23 --- 31 --- 39 --- 47 --- 55 --- 63    lpar08

    6 --- 14 --- 22 --- 30 --- 38 --- 46 --- 54 --- 62    lpar07

    5 --- 13 --- 21 --- 29 --- 37 --- 45 --- 53 --- 61    lpar06

    4 --- 12 --- 20 --- 28 --- 36 --- 44 --- 52 --- 60    lpar05

    3 --- 11 --- 19 --- 27 --- 35 --- 43 --- 51 --- 59    lpar04

    2 --- 10 --- 18 --- 26 --- 34 --- 42 --- 50 --- 58    lpar03

    1 ---  9 --- 17 --- 25 --- 33 --- 41 --- 49 --- 57    lpar02
```

```
     0 ---  8 --- 16 --- 24 --- 32 --- 40 --- 48 --- 56    lpar01
```

Note: Communication does occur between tasks 56 & 1, 57 & 2, and so on.

Tip: Assuming the communication pattern of an application is well understood and conducive to this approach, reordering the hostfile has the potential to improve performance without recoding the application.

Note: Under LoadLeveler, task to processor mapping is controlled with the task_geometry keyword.

3.6.6 Some final recommendations

Here are some final thoughts on the subject of application tuning.

Tip: Still confused? Start with these steps:

- Compile with these options:

  ```
  -O -qmaxmem=-1 -qarch=auto -qtune=auto
  ```

- Use ESSL and MASS wherever possible, as these are highly tuned for IBM AIX platforms.

Still want more? Try these next steps:

- Use more aggressive optimization, always checking for correctness.
- Profile the application in order to identify hotspots and bottlenecks in the code. It does no good to optimize a section of code in which little time is spent. Consult these references for profiling procedures and tuning examples:
  – *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041
  – *AIX Version 4, Optimization and Tuning Guide for Fortran, C, and C++*, SC09-1705
  – *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide*, SG24-5155

3.7 Network connectivity

High performance inter-node communication is a very important component of many cluster applications. The most basic measurements of the performance of the communication subsystems are *latency* and *bandwidth*. Latency is the overhead associated with sending data between two processors. Bandwidth is the rate at which data can be transmitted between two processors.

In this section, we provide some suggestions to help you get better performance out of your Cluster 1600 interconnect.

3.7.1 SP Switch2 PCI Attachment Adapter

The following functionality comes with the SP Switch2 PCI Attachement Adapter.

Aggregate IP

PSSP has a new function called *Aggregate IP*. The multilink, aggregate IP pseudo-device enables Cluster 1600 nodes to use one IP address to communicate over two SP Switch2 planes. The benefits are:

- Higher data throughput. The data is striped across the two SP Switch2 PCI Attachment Adapters.
- Higher availability. If one path has a problem, the other path is automatically used without interrupting the communication.

A dual-plane SP Switch2 is required to use the aggregate IP function. Additionally, each node that will use the aggregate IP function must have two SP Switch2 PCI Attachment Adapters, one for each plane. The aggregate IP function provides a virtual-device interface (that is, a third IP address configured as ml0), to enable IP messages to be transmitted in a more economical manner called striping. The striping technique provides the capability to transmit consecutive IP data across two fully operational adapters. It takes advantage of the combined bandwidth of both adapters. For example, when an IP message is sent between nodes and both nodes have access to both available switch networks, consecutive datagrams are sent in a pattern similar to css0, css1, css0, css1, and so on.

Once you have configured both css0 and css1 on a node, you can use the **spaggip** command or **smitty** to configure ml0.

Example 3-4 shows how to configure aggregate IP using the **smitty sp_agg_dialog** fast path.

Example 3-4 Aggregate IP configuration through smitty

```
[c37f1rp05][\]> smitty sp_agg_dialog
```

```
Aggregate IP Information

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  Start Frame                                        []
  Start Slot                                         []
  Node Count                                         []

  OR

  Node Group                                         []

  OR

  Node List                                          [1]

* Starting Node's Aggregate IP Address               [192.168.5.1]
* Netmask                                            [255.255.255.0]
* Adapter List                                       [css0,css1]
  Update Interval                                    []
  Update Threshold                                   []
  Skip IP Addresses for Unused Slots?                 no

Esc+1=Help          Esc+2=Refresh       Esc+3=Cancel        Esc+4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

The Update Interval specifies the time interval in seconds between the aggregate network route table refreshes. The value must be between 3 and 10 inclusive. The default is 3.

The Update Threshold specifies the number of missed refresh updates before the network connection is dropped. The value must be between 10 and 400 inclusive. The default is 10.

Figure 3-5 shows how to configure aggregate IP using the **spaggip** command.

Example 3-5 Aggregate IP configuration with the spaggip command

```
# /usr/lpp/ssp/bin/spaggip -l 1 -i css0,css1 192.168.5.1 255.255.255.0
```

Tunables

For the SP Switch2 adapters, the following parameters can affect the performance of the communication:

Note: The SP Switch adapters are also affected by the same tunable parameters. For more information, see *RS/6000 SP System Performance Tuning Update,* SG24-5340.

win_poolsize

Also referred to as *device memory*, win_poolsize is the total, maximum amount of pinned system memory (in bytes) that can be used as interface network FIFO buffers for SP Switch2 adapter windows.

win_minsize

The guaranteed minimum amount of device memory (in bytes) per SP Switch2 adapter window; win_minsize ensures that all tasks in a job have the minimum required device memory to run.

win_maxsize

The maximum amount of device memory (in bytes) per SP Switch2 adapter window, win_maxsize internally limits device memory usage for each window; win_maxsize is further bounded by available device memory and by any job-scheduler specified limit.

rpoolsize

Size of the IP receive buffer pool (in bytes).

spoolsize

Size of the IP send buffer pool (in bytes).

These parameters can be changed with the **chgcss** command. For more information, see the **chgcss** man page.

Notes: The maximum values for rpoolsize and spoolsize in the SP Switch2 PCI Attachment Adapter are 32 MB (33554432) each. This doubles the maximum values for the previous switch adapters, and the values can be changed on the fly without a reboot.

The available number of *switch adapter windows* in the SP Switch2 PCI Attachment Adapter is 68 (64 US + 1 VSD + 1 GPFS + 1 IP + 1 SVC)

The number of US adapter windows has a direct effect over the MPI threads. You can now increase the number of MPI threads (up to 64 per adapter) in a pSeries 690 node (LPAR).

Example 3-6 shows what the output from the **`lsattr -El css0`** looks like when run against a SP Switch2 PCI Attachment Adapter.

Example 3-6 Output of lsattr command on SP Switch2 PCI Attachment Adapter

```
[c37f1rp08][/]> lsattr -El css0

adapter_memory 0xf1000000      Adapter memory address        False
adapter_size   0x00800000      Adapter memory size           False
sdram_start    0xf0000000      SDRAM memory address          False
sdram_size     0x01000000      SDRAM memory size             False
TOD_address    0xf1800000      TOD address                   False
win_poolsize   1107296256     Total window memory pool size True
win_maxsize    16777216       Maximum window memory size    True
win_minsize    524288         Minimum window memory size    True
int_priority   3              Interrupt priority            False
int_level      2853           Bus interrupt level           False
spoolsize      2097152        Size of IP send buffer        True
rpoolsize      2097152        Size of IP receive buffer     True
khal_spoolsize 524288         Size of KHAL send buffer      True
khal_rpoolsize 524288         Size of KHAL receive buffer   True
adapter_status css_ready      Configuration status          False
diags_prog                    Diagnostic program            True
ucode_version  1              Micro code version            True
ucode_name     /etc/microcode/cor_ucode Micro code name     True
window0-15     VSD   AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
   AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners         True
window16-31    AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
   AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners         True
window32-47    AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
   AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners         True
window48-63    AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
   AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners         True
window64-65    AVAIL AVAIL             window owners         True
driver_debug   0               Device Driver Debug           True
```

```
ip_chksum       off             if_cl checksum             True
ip_debug        0               if_cl debug level          True
proto_debug     off             proto debug                True
```

3.7.2 EtherChannel configurations

EtherChannel configurations can improve performance, but all of the normal things that impact traditional Ethernet performance (adapter placement, speed, system resources, disk read speed, application efficiency, tuning options, and so forth) also impact throughput on the EtherChannel. The EtherChannel imposes very little overhead so throughput is generally in line with running the adapters individually. An example of a four-link EtherChannel is shown in Figure 3-4.

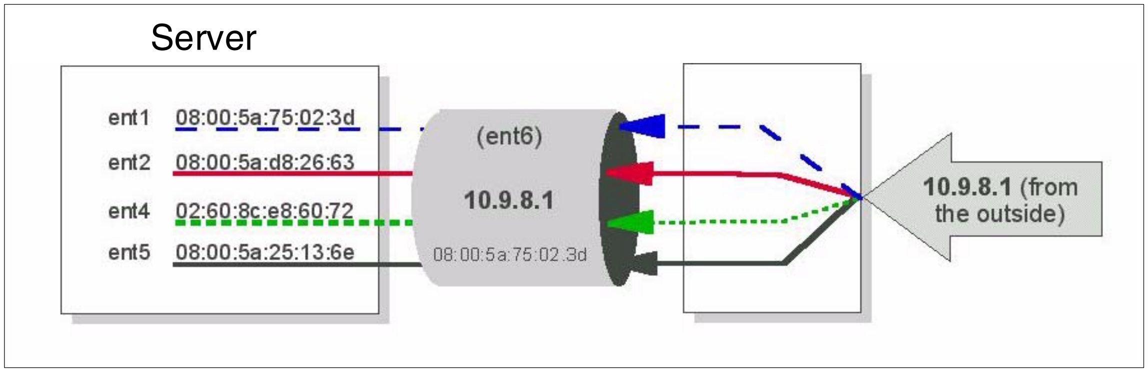


Figure 3-4 Four-link EtherChannel schema

In EtherChannel environments, however, traffic distribution can also impact throughput. Ideally, traffic should be spread evenly across the adapters in the EtherChannel so as not to saturate one adapter while leaving another one idle. Incoming packets are distributed as configured by the EtherChannel switch administrator, so the AIX system administrator cannot control this. You can have an idea of traffic distribution on the channel using **netstat -v** ***ent#***, where ent# is the EtherChannel device. Statistics for incoming packets show the distribution determined by the switch. The outgoing statistics reflects the AIX control over the packet distribution between the member adapters of the EtherChannel.

In Gigabit Ethernet environments, there are more variables (CPU speed, capacity, I/O bus limits, and so on) that can cause bottlenecks compared with fast Ethernet. In optimized lab tests, a two-link GigaChannel can provide 1.7 times the throughput of a single Gigabit link and a four-link GigaChannel can provide 2.5 to 3 times the throughput of a single Gigabit link.

AIX has two user-configurable options (modes) for distribution of outgoing packets. In standard mode, the distribution is based upon the last byte of the destination IP address (for TCP/IP packets) or MAC (for non-IP traffic) address divided by the number of adapters in the EtherChannel. The remainder of the calculation determines the adapter selected. A remainder of zero indicates the first adapter in the channel, one indicates the second, and so forth. The more clients, the more chances that the random nature of the addresses will result in fairly even traffic distribution. In environments of one or two clients, the distribution will likely be skewed because all traffic between a host pair goes out over the same adapter. In this situation, round-robin distribution mode is a better choice. Round-robin mode distributes the outgoing traffic evenly across all of the adapters in the EtherChannel, but there may be a performance penalty due to the potential for out-of-order packets.

3.7.3 Internet Protocol (IP) and User Space (US) switch windows

MPI tasks can communicate across the switch with either Internet Protocol (IP) protocol or User Space (US) protocol. Typically, US is the protocol of choice, but there are certain situations where IP may be the better choice.

IP uses a single window and avoids the polling. However, use of this single window requires that IP multiplex its traffic through the kernel. So IP is trading context switching for polling. Additionally, IP can also use much larger packets than US. These packets, known as *Jumbo Packets*, can be almost 60 times larger than the largest US packet. For a given amount of data being sent by either IP or US, these IP packets require fewer writes to the adapter and fewer handshake messages on the receive side than their US counterparts.

In contrast, US works by opening and reserving a switch adapter window (which is a special location on the adapter) to provide an independent communication path through the adapter for each MPI task running on the node. This provides a direct path from the user process to the switch adapter and bypasses the need for a kernel context switch. A polling algorithm in the switch adapter microcode determines if messages are present in these windows and need to be serviced. As the number of MPI tasks increase, so do the number of windows that need to be polled.

As you can already guess, there is a break-even point between polling and context switching. On one side of the point, US performs better than IP, and on the other side, IP performs better than US.

Here are the characteristics from each side of the point:

- US performs better than IP for applications:
  - That use small messages
  - That have a small number of MPI tasks per adapter

  - That tend to be CPU constrained
- IP performs better than US for applications:
  - That use very large messages
  - That have a large number of MPI tasks per adapter
  - That tend to be adapter bus constrained (rather than CPU or memory constrained)

The determining factor is the amount of overhead spent on packet management. As the number of tasks increases, the overhead required to manage multiple adapter windows becomes the limiting factor for US performance.

The IP and US bandwidths for 16 MB message size in a single-plane SP Switch2 with an SP Switch2 PCI Attachment Adapter are shown in Table 3-6. These test results are from a fully-loaded, 32-way pSeries 690 running in Full System Partition (SMP) mode. Note that for a single task, US performs better than IP, even at the very large (16 MB) message size. However, when the number of tasks is increased to 32, IP performs better than US.

Table 3-6 16 MB message single-plane bandwidths

Number of tasks	IP (bytes/sec)	US (bytes/sec)
1	205	217
32	218	207

The IP and US bandwidths for 16 MB message size in a dual-plane SP Switch2 with two SP Switch2 PCI Attachment Adapters are shown in Table 3-7. Again, these test results are from a fully-loaded, 32-way pSeries 690 running in Full System Partition (SMP) mode. As you can see, the bandwidth for either protocol can be improved by adding an additional SP Switch2 plane with an SP Switch2 PCI Attachment Adapter to the system, but even in this environment, the same performance pattern can be observed.

Table 3-7 16 MB message dual-plane bandwidths

Number of Tasks	IP (bytes/sec)	US (bytes/sec)
1	221	375
32	382	364

Even though the bandwidth significantly improves with the addition of another adapter in this configuration, the PCI bus is still the limiting factor, not the CPUs. IP maintains a 5 percent bandwidth edge for the 32 task case, even when the second adapter is added.

Additional adapters can effectively be added by increasing the number of LPARs. This enables the system to make better use of the overall bandwidth of the interconnect, but at the potential cost of increasing the amount of communications required between MPI tasks. Smaller LPARs reduce the differences in performance between US and IP because of the inherent decrease in tasks per adapter. Also, with fewer CPUs to drive the adapters, the IP path soon becomes CPU limited, and US becomes the clear performance winner for all message sizes.

The SP Switch2 PCI Attachment Adapter does not introduce any different choices than other generations of adapters have had. The big difference is that with the pSeries 690, server applications that were previously CPU bound may now become adapter bound, and IP may now be an appropriate choice for these applications.

The IBM Parallel Environment MPI Library provides three transport protocols:

- Shared memory
- User Space (US)
- Internet Protocol (IP)

These transports are selected at run time by the user, and the appropriate libraries are loaded. The applications have no awareness of which transport is being used, providing maximum flexibility. Shared memory has lower latency and higher bandwidth than the adapter. It can be used in combination with either US or IP for inter-node transports. (Shared memory MPI use is turned on via the MP_SHARED_MEMORY environment variable.) For inter-node transport, US provides better performance for applications that use short messages, or for which the additional CPU required to implement IP protocol is not available. IP is a better choice for applications that are dominated by very long messages, and there are many CPUs and tasks all sharing a single adapter.

3.8 What is next

This completes the conceptual portion of the book. In Chapter 4, “Investigations” on page 77, we take you through our experiments with the features and concepts that you have just finished reading about here. As you will see, we give you the blow-by-blow details of what we did, what we found, and where we went wrong. The idea is not to give you a cookbook, but rather to show you how we approached testing these new things out. Some of the tests worked, while others did not. However, either way, we came away the wiser for it. We hope you will too.

4

Investigations

For us, this chapter was where the real work occurred. This chapter contains our findings from the experiments we performed in an attempt to better understand the concepts presented in the first set of chapters. As you will see, some of our experiments were successful, while others were not so successful. The point, however, is that we learned something in each of them, and we hope that you will as well. If we can help you avoid some dead ends, then our mission has been accomplished. We hope that you have fun reading it. We certainly had fun doing it.

4.1 Technical large page investigation

For this investigation, we wanted to see if we could see any differences between running the AIX sort program in normal mode, in large memory mode, and in technical large page mode. The AIX sort program seemed like a good choice for readily available code that is fairly memory intensive. These series of investigations was run on a 4-way, 4 GB normal LPAR with the software levels shown in Example 4-1.

Example 4-1 Software levels for technical large page investigation

```
**************************************************
Software Levels As Of: Sat Jun  8 10:22:43 EDT 2002
**************************************************
AIX:                 bos.mp             5.1.0.27
PSSP:                ssp.basic          3.4.0.7
LoadLeveler:
GPFS:
RVSD:
VSD:
RSCT:                rsct.basic.rte     2.2.1.0
POE:
PESSL:
ESSL:
FORTRAN:
PERL:                perl.rte           5.6.0.0
C:                   xlC.rte            5.0.2.1
```

4.1.1 Setting up the environment for technical large page

First, we had to prepare the test environment. This involved:

- Preparing the environment for technical large page usage
- Creating a non-root user with technical large page capabilities
- Creating modified AIX sort executables
- Preparing the input dataset

Preparing the environment for technical large page usage

The steps required to prepare the environment were as follows:

1. Run **`vmtune`** to set aside memory for technical large pages.
2. Run **`bosboot`** to update the boot logical volume.
3. Run **`shutdown`** to reboot.

Since our LPAR had 4 GB of total memory, we used 1 GB of that for technical large page. Remember, this is pinned memory. The steps to enable technical large page are illustrated in Example 4-2.

Example 4-2 Preparing the environment for technical large page

```
[c37f1rp05][/]> /usr/samples/kernel/vmtune -g 16777216 -L 64

vmtune:  current values:

...
        -s                -n         -S          -L          -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0                 0          0           0      16777216        0
...

number of valid memory pages = 1048573  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 269042    numperm=26.4% of real memory
number of compressed memory pages = 0   compressed=0.0% of real memory
number of client memory pages = 0       numclient=0.0% of real memory
# of remote pgs sched-pageout = 0       maxclient=79.9% of real memory

vmtune:  new values:

...
        -s                -n         -S          -L          -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0                 0          0          64      16777216        0
...

number of valid memory pages = 1048573  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 538084    numperm=52.8% of real memory
number of compressed memory pages = 0   compressed=0.0% of real memory
number of client memory pages = 0       numclient=0.0% of real memory
# of remote pgs sched-pageout = 0       maxclient=79.9% of real memory

[c37f1rp05][/]> bosboot -ad /dev/ipldevice

bosboot: Boot image is 13276 512 byte blocks.

[c37f1rp05][/]> shutdown -Fr
```

Creating a non-root user with technical large page capabilities

Next, we created a non-root test user (tlp) for technical large page. We set this user's limits to unlimited and enabled the account for technical large page usage. These steps are illustrated in Example 4-3 on page 80.

Example 4-3 Enabling a non-root user account for technical large page

```
[c37f1rp05][/]> grep -p tlp /etc/security/limits

tlp:
        fsize = 2097151
        core = 2097151
        cpu = -1
        data = -1
        rss = -1
        stack = -1
        nofiles = 2000

[c37f1rp05][/]> chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE tlp

[c37f1rp05][/]>
```

Creating modified AIX sort executables

We then created three versions of the AIX sort program:

- Normal version: sort.normal
- Large memory enabled version: sort.lmem
- Large page enable version: sort.lpage

The steps for modifying each of the executables is shown in Example 4-4.

Example 4-4 Modifying the executables for large memory and large page

```
[c37f1rp05][/]> cd ~tlp
# Create sort.small
[c37f1rp05][/home/tlp]> cp /usr/bin/sort ./sort.normal
# Create sort.lmem
[c37f1rp05][/home/tlp]> cp /usr/bin/sort ./sort.lmem
[c37f1rp05][/home/tlp]> echo '\0200\0\0\0' | \
> dd of=./sort.lmem bs=4 count=1 seek=19 conv=notrunc
1+0 records in.
1+0 records out.
# Create sort.lpage
[c37f1rp05][/home/tlp]> cp /usr/bin/sort ./sort.lpage
[c37f1rp05][/home/tlp]> /usr/ccs/bin/ldedit -blpdata ./sort.lpage
/usr/ccs/bin/ldedit:  File ./sort.lpage has been updated.
# Give them to tlp
[c37f1rp05][/home/tlp]> chown tlp.staff sort*
```

Preparing the input dataset

The final step in the preparation was to create an input dataset. We needed something large enough to cause the sort program to run for a long enough period of time so that we could examine the system while it was running. On the other hand, it could not be so large as to cause the sort program to crash due to lack of system resources. So, we turned to the *Melville MD10 dataset* as the perfect choice.

4.1.2 Creating the Melville MD10 dataset for technical large page

The Melville MD10 dataset is but one of a whole family of datasets in the *Melville MD series*. The advantage to the MD10 dataset is that it is just about the perfect size for the sort tests, coming in at 19924620 bytes and 9962310 lines.

Credit for the initial research into the Melville MD datasets belongs to Richard E. Weingarten, IBM Boulder, who used early versions in his investigations of *Distributed Computing Environment (DCE) Cross-Cell Propagation Issues*. We are indebted to him for this pioneering work.

Example 4-5 is a listing of a script that we used to generate the MD10 dataset for our technical large page investigation.

Example 4-5 Creation of the Melville MD10 dataset

```
[c37f1rp05][/install/marc]> cat mkmd10.ksh
#!/usr/bin/ksh
#desc creates the melville md10 dataset
#
# Purpose: This script creates the Melville MD10 dataset
#          for use in the redbook investigations.
# Inputs:  mobydick.txt
#
# Modification History
# 06/07/02 Marc Genty (NCAR) created
#
IFILE=/install/marc/mobydick.txt
OFILE=/install/marc/md10

cp ${IFILE} ${OFILE}

/usr/bin/perl -i.bak -pe "s/\s//g" ${OFILE}
/usr/bin/perl -i.bak -pe "s/(.)/\1\n/g" ${OFILE}

for i in 1
do
  for j in 1 2 3 4 5 6 7 8 9 10
  do
```

```
    cat ${OFILE} >> ${OFILE}.lpage
  done
done

/usr/bin/rm ${OFILE}.bak
```

For input, the script requires the mobydick.txt file. This file is the ASCII version of Herman Melville's classic novel, *Moby Dick*. It is available for download at:

ftp://ibiblio.org/pub/docs/books/gutenberg/etext01/moby10b.txt

The mkmd10.ksh script takes this file, strips out all whitespace and newline characters, separates each of the remaining characters onto their own lines, and concatenates this output ten times into the final md10.lpage dataset.

4.1.3 Running the tests for technical large page

We first did an initial test run to verify that the sort executables were modified correctly. Prior to running any of the tests, we also set the LD_CNTRL environment variable in the tlp account to force mandatory use of technical large pages. Remember that modifying the executable for technical large page only sets it for advisory mode. If you want mandatory mode, you have to use the LD_CNTRL environment variable.

Example 4-6 shows the test run for the sort.normal program.

Example 4-6 Test run of sort.normal

```
[c37f1rp05][/]> su - tlp
$ export LD_CNTRL=LARGE_PAGE_DATA=M
$ timex ./sort.normal -o /dev/null /install/marc/md10.lpage

# In a second root window...

[c37f1rp05][/]> ps -ef|grep sort|grep -v grep
     tlp 12354 14598   0 12:40:00  pts/0  0:00 timex ./sort.normal -o /dev/null
/install/marc/md10.lpage
     tlp 14980 12354  89 12:40:00  pts/0  0:03 ./sort.normal -o /dev/null
/install/marc/md10.lpage
[c37f1rp05][/]> svmon -P 14980 -r | more
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   14980 sort.normal      13217     2068      792     8343      N     N     N

     PageSize       Inuse        Pin       Pgsp     Virtual
         4 KB       13217       2068        792        8343
        16 MB           0          0          0           0
```

```
    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Vrtual
       0         0 work kernel seg                   -   4951  2066  792  4951
                   Addr Range: 0..25279
   64e19         - pers large file /dev/lvinstall:16 -   4865     0    -     -
                   Addr Range: 0..4864
   7401d         d work shared library text          -   2245     0    0  2245
                   Addr Range: 0..60123
   5ce97         2 work process private              -   1125     2    0  1125
                   Addr Range: 0..1167 : 65310..65535
   405d0         f work shared library data          -     22     0    0    22
                   Addr Range: 0..1817
   545f5         1 pers code,/dev/hd1:2061           -      9     0    -     -
                   Addr Range: 0..8
   68e5a         - pers /dev/hd9var:6541             -      0     0    -     -
```

Notice that the LPage column is set to N, that the Pagesize section reports only 4 KB pages in use, and that Segment 2 has both heap (0..1167) and stack (65310..65535). This is what we were expecting to see.

We repeated the same test with the sort.lmem program. Example 4-7 shows that test run.

Example 4-7 Test run of sort.lmem

```
[c37f1rp05][/]> su - tlp
$ export LD_CNTRL=LARGE_PAGE_DATA=M
$ timex ./sort.leme -o /dev/null /install/marc/md10.lpage

# In a second root window...

[c37f1rp05][/]> ps -ef|grep sort|grep -v grep
     tlp 12374 14598   0 12:52:11  pts/0  0:00 timex ./sort.lmem -o /dev/null
/install/marc/md10.lpage
     tlp 15004 12374 102 12:52:11  pts/0  0:04 ./sort.lmem -o /dev/null
/install/marc/md10.lpage
[c37f1rp05][/]> svmon -P 15004 -r|more
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   15004 sort.lmem        13356     2069      792     8343      N     N     N

     PageSize       Inuse        Pin       Pgsp    Virtual
         4 KB       13356       2069        792       8343
        16 MB           0          0          0          0

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Vrtual
       0         0 work kernel seg                   -   4951  2067  792  4951
                   Addr Range: 0..25279
   64e19         - pers large file /dev/lvinstall:16 -   4865     0    -     -
```

```
                Addr Range: 0..4864
   7401d       d work shared library text          -   2245     0    0  2245
                Addr Range: 0..60123
   44e11       3 work shmat/mmap                   -   1110     0    0  1110
                Addr Range: 0..1167
   60eb8       - pers /dev/hd9var:6512             -    139     0    -     -
                Addr Range: 0..138
   405d0       f work shared library data          -     22     0    0    22
                Addr Range: 0..1817
   40e10       2 work process private              -     15     2    0    15
                Addr Range: 65310..65535
   5c5f7       1 pers code,/dev/hd1:2063           -      9     0    -     -
                Addr Range: 0..8
   4ceb3       5 work shmat/mmap                   -      0     0    0     0
   28eaa       9 work shmat/mmap                   -      0     0    0     0
   5ce97       7 work shmat/mmap                   -      0     0    0     0
   40e70       6 work shmat/mmap                   -      0     0    0     0
   3ceaf       a work shmat/mmap                   -      0     0    0     0
   24ea9       4 work shmat/mmap                   -      0     0    0     0
   48eb2       8 work shmat/mmap                   -      0     0    0     0
```

Notice that the LPage column is still set to N, that the Pagesize section still reports only 4 KB pages in use, but that Segment 2 has only stack (65310..65535), and Segment 3 now has the heap (0..1167). This is also what we were expecting to see.

Finally, we repeated the same test with the sort.lpage program. Example 4-8 shows that test run.

Example 4-8 Test run of sort.lpage

```
[c37f1rp05][/]> su - tlp
$ export LD_CNTRL=LARGE_PAGE_DATA=M
$ timex ./sort.leme -o /dev/null /install/marc/md10.lpage

# In a second root window...

[c37f1rp05][/]> ps -ef|grep sort|grep -v grep
     tlp 12388 15020 120 13:00:45  pts/0  0:05 ./sort.lpage -o /dev/null
/install/marc/md10.lpage
     tlp 15020 14598   0 13:00:45  pts/0  0:00 timex ./sort.lpage -o /dev/null
/install/marc/md10.lpage
[c37f1rp05][/]> svmon -P 12388 -r | more
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   12388 sort.lpage       77737    67604      792    72779      N     N     Y

     PageSize      Inuse        Pin       Pgsp    Virtual
```

```
       4 KB       12201       2068        792       7243
      16 MB          16         16          0         16

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Vrtual
   40e10         3 work shmat/mmap                   Y  65536 65536    0 65536
       0         0 work kernel seg                   -   4951  2066  792  4951
                   Addr Range: 0..25279
   64e19         - pers large file /dev/lvinstall:16 -   4865     0    -     -
                   Addr Range: 0..4864
   7401d         d work shared library text          -   2245     0    0  2245
                   Addr Range: 0..60123
   48eb2         - pers /dev/hd9var:6530             -     84     0    -     -
                   Addr Range: 0..83
   405d0         2 work process private              -     25     2    0    25
                   Addr Range: 0..13 : 65310..65535
   28eaa         f work shared library data          -     22     0    0    22
                   Addr Range: 0..1817
   585f6         1 pers code,/dev/hd1:2062           -      9     0    -     -
                   Addr Range: 0..8
```

Now notice that the LPage column is set to Y, the Pagesize section now reports both 4 KB and 16 MB pages in use, and that Segment 2 has primarily stack (0..13 : 65310..65535), and all of Segment 3 has been allocated to LPage for heap (65536). The big surprise for us was that the svmon listing looks more like the normal listing than the large memory one. We tried modifying the sort.lpage executable to also use large memory (via the **echo dd** command), but it had no effect. The output from that test run was identical to the sort.lpage one.

We should also mention that we ran the **ps**, **vmstat**, and **topas** commands during these test runs and found no new information in any of them.

Having validated the three executables, we were then ready to perform the experiment. As with the test runs, we used timex as the measurement tool. The test runs consisted of 40 individual runs of each program back-to-back with the LPAR quiesced and dedicated for the runs. Example 4-9 is a listing of the script that we used to do the runs.

Example 4-9 Test run script for sort executable variants

```
[c37f1rp05][/home/tlp]> cat sortem.ksh
#!/usr/bin/ksh

echo " "
echo "Normal Sort"
echo " "
for j in 1 2 3 4
do
  for i in 1 2 3 4 5 6 7 8 9 10
  do
    timex sort.normal -o /dev/null /install/marc/md10.lpage
  done
done

echo " "
echo "Large Memory Sort"
echo " "
for j in 1 2 3 4
do
  for i in 1 2 3 4 5 6 7 8 9 10
  do
    timex sort.lmem -o /dev/null /install/marc/md10.lpage
  done
done

echo " "
echo "Large Page Sort"
echo " "
for j in 1 2 3 4
do
  for i in 1 2 3 4 5 6 7 8 9 10
  do
    timex sort.lpage -o /dev/null /install/marc/md10.lpage
  done
done
```

To trap the output, we ran sortem.ksh from within the **`script`** command. Table 4-1 summarizes the results.

Table 4-1 Technical large page AIX sort results

Program: AIX sort				
sort.normal				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	115.63	116.18	115.43	0.17
User	114.70	115.03	114.48	0.14
Sys	0.39	0.65	0.27	0.09
sort.lmem				
Real	115.58	115.95	115.44	0.11
User	114.66	114.99	114.43	0.13
Sys	0.37	0.56	0.24	0.07

Program: AIX sort				
sort.lpage				
Real	115.70	115.96	115.52	0.13
User	113.85	114.01	113.69	0.07
Sys	1.18	1.39	1.05	0.08

Notice that the overall run times for all three show no significant differences. However, also notice that for this test, it appears that using technical large page with this executable trades system time for user time as can be seen in the second two rows of each run.

4.1.4 Conclusions from the first attempt at technical large page

So, why did we not see any significant differences in the overall times between the runs of the three modified sort executables? Well, it is kind of like the missing $1.00 in Chapter 1, “Introduction” on page 1. The answer was right in front of us, and, once we saw it, we could not understand why we did not see it all along. Let us go back to the initial test run output from the sort.small program. Example 4-10 is a repeat of that listing.

Example 4-10 Test run of sort.normal (repeated)

```
[c37f1rp05][/]> su - tlp
$ export LD_CNTRL=LARGE_PAGE_DATA=M
$ timex ./sort.normal -o /dev/null /install/marc/md10.lpage

# In a second root window...

[c37f1rp05][/]> ps -ef|grep sort|grep -v grep
     tlp 12354 14598   0 12:40:00  pts/0  0:00 timex ./sort.normal -o /dev/null
/install/marc/md10.lpage
     tlp 14980 12354  89 12:40:00  pts/0  0:03 ./sort.normal -o /dev/null
/install/marc/md10.lpage
[c37f1rp05][/]> svmon -P 14980 -r | more
-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   14980 sort.normal      13217     2068      792     8343      N     N     N

     PageSize      Inuse        Pin       Pgsp    Virtual
         4 KB      13217       2068        792       8343
        16 MB          0          0          0          0

    Vsid      Esid Type Description              LPage  Inuse   Pin Pgsp Vrtual
       0         0 work kernel seg                   -   4951  2066  792  4951
```

```
                Addr Range: 0..25279
  64e19       - pers large file /dev/lvinstall:16 -   4865     0    -     -
                Addr Range: 0..4864
  7401d       d work shared library text          -   2245     0    0  2245
                Addr Range: 0..60123
  5ce97       2 work process private              -   1125     2    0  1125
                Addr Range: 0..1167 : 65310..65535
  405d0       f work shared library data          -     22     0    0    22
                Addr Range: 0..1817
  545f5       1 pers code,/dev/hd1:2061           -      9     0    -     -
                Addr Range: 0..8
  68e5a       - pers /dev/hd9var:6541             -      0     0    -     -
```

Notice the very last line of the svmon output, the one with the /dev/hd9var in it. Therein lies the reason we did not see any significant differences in the runs of the three sort executables. If you run **sort** and watch /var/tmp, you will see something much like what is illustrated in Example 4-11.

Example 4-11 Directory listing from /var/tmp while sort is running

```
[c37f1rp05][/var/tmp]> ls -l
total 20983
-rw-r--r--   1 root     system        7205 Jun 06 15:00 dpid2.log
p---rw----   1 root     system           0 Jun 06 15:01 em_trap.9.114.189.61
p---rw----   1 root     system           0 Jun 06 15:01 errlog_entry
-rw-r--r--   1 root     system       45061 Jun 06 15:00 hostmibd.log
-rw-r--r--   1 root     system       38996 Jun 06 15:02 snmpd.log
-rw-------   1 tlp      staff      1198370 Jun 08 13:31 stm15076aaaaa
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaab
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaac
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaad
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaae
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaaf
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaag
-rw-------   1 tlp      staff      1198370 Jun 08 13:32 stm15076aaaah
-rw-------   1 tlp      staff      1044480 Jun 08 13:32 stm15076aaaai
```

The AIX sort program does not do in-core sorting. Instead, it uses temporary files in /var/tmp. Because it is not doing the sort in memory (via something like the malloc system call), it is not using the heap area for sorting. Therefore, enabling technical large page for sort has no effect, because this only enables those kinds of pages for the heap/data area of the program, and not for the persistent segments through which file data passes.

4.1.5 The second attempt at technical large page

For our next test of technical large page, we continued with the sort idea, but this time with our own program that did in-core sorting of a large set of randomly-generated numbers. The source code for the program is shown in Example 4-12.

Example 4-12 The gabisort program

```
[c37f1rp05][/home/tlp]> cat gabisort.c
/*
**
** Program: gabisort.c
**
** Author:  Gabriel Radu (IBM Romania)
**
** Notes:   Quick and dirty implementation of bubblesort and
**          quicksort for use in the redbook investigations.
**
*/

#include <stdio.h>

#define RND random()

int main(void)
{
  int COUNTER=246841000;

  int *ptr,i,*BASE;
  int *ptra,*ptrb,*ptrc;
  void sort_bubble(int *sir, int LONG);
  void afiseaza_sir(int *sir, int LONG);
  void quick_sort(int *p,int *s, int *d);
  void int_swap(int *,int *);
  int * set_sir(int LONG);

  printf("\n");
  printf("Generating %d numbers\n",COUNTER);
  ptr=set_sir(COUNTER);
  BASE=ptr;
  printf("Sorting -> %d numbers\n",COUNTER);

  /*
  afiseaza_sir(ptr,COUNTER);
  */

  printf("\n");
  printf("Starting Sort\n");
  /*
  sort_bubble(ptr,COUNTER);
  */
  ptra=BASE;
  ptrb=BASE;
  ptrc=BASE+COUNTER;
  quick_sort(ptra,ptrb,ptrc);
  ptr=BASE;
  /*
  afiseaza_sir(ptr,COUNTER);
  */
  printf("Finished Sort\n");
  printf("\n");

  free(BASE);
  exit(0);
}
/* End Main */

void int_swap(int *pax,int *pbx)
{
  int temp;
  temp=*pbx;
  *pbx=*pax;
  *pax=temp;
}

void sort_bubble(int *sir, int LONG)
{
  int j=LONG,test_sort=1,i,temp;
  int h=0;
  while(test_sort==1) {
    test_sort=0;
    for(i=0;i<j;i++) {
      if(sir[i] > sir[i+1]) {
        int_swap((sir+i),(sir+i+1));
        test_sort=1;
      }
    }
    j--;
  }
}

void afiseaza_sir(int *sir, int LONG)
{
  int i;
  for (i=0; i<=LONG;i++) {
    printf("%d \t  %d \n",i,*(sir+i));
  }
```

```
}

int * set_sir(int LONG)
{
  int *aa,i,*bbb;
  srandom(100);
  aa=(int *)malloc(LONG*sizeof(int));
  bbb=aa;
  if ( aa == NULL ) {
    printf("Allocation Error\n");
    exit(1);
  }
  for (i=0;i<=LONG;i++) {
    *aa=RND;
    aa=aa+1;
    /*
    printf("%d\n",i);
    */
  }
  return bbb;
}

void quick_sort(int *p, int *s, int *d)
{
  int *sant;
  int *pi;
  pi=p;
  if(s>=d) {
    return;
  }
  sant=s;
  for(pi=s+1;pi<=d;pi++) {
    if(*pi<*s) {
      int_swap(++sant,pi);
    }
  }
  int_swap(s,sant);
  quick_sort(p,s,sant-1);
  quick_sort(p,sant+1,d);
}
```

We used the quicksort version of the code. The gabiqsort executable was compiled with **`cc -g gabisort.c -o gabiqsort`** on a system with software levels illustrated in Example 4-13.

Example 4-13   Software levels of the system used to compile gabisort

```
[bd0101en][/]> ~mgenty/pmrinfo
```

```
*************************************************
Software Levels As Of: Sat Jun  8 11:54:47 MDT 2002
*************************************************
AIX:                  bos.mp              5.1.0.25
PSSP:
LoadLeveler:          LoadL.full          3.1.0.2
GPFS:
RVSD:
VSD:
RSCT:                 rsct.basic.rte      2.2.1.0
POE:                  ppe.poe             3.2.0.2
PESSL:                pessl.rte.smp43     2.3.0.1
ESSL:                 essl.rte.smp43      3.3.0.0
FORTRAN:              xlfrte              7.1.1.2
PERL:                 perl.rte            5.6.0.0
C:                    xlC.rte             5.0.2.1
```

Following the same model that we had used with the AIX sort executable, we made modified copies of the gabiqsort executable. In this case, we only made the gabiqsort.lmem and gabiqsort.lpage versions, because the input set was too large for the gabiqsort.normal to handle.

We performed the initial test runs of each executable to verify that the **svmon** output was as we were expecting, and it was.

Finally, we modified the sortem.ksh script to perform the 40 test runs of each of the two gabiqsort executables. The results of those runs are summarized in Table 4-2.

Table 4-2 Technical large page gabiqsort results

Program: gabiqsort				
gabiqsort.lmem				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	671.37	672.70	670.90	0.56
User	667.76	668.39	667.32	0.27
Sys	2.54	3.55	2.12	0.38
gabiqsort.lpage				
Real	668.68	670.22	668.07	0.61
User	664.63	665.43	664.22	0.26
Sys	3.52	4.47	3.26	0.40

For these sets of tests, it appears that technical large page does produce a statistically significant difference in run times. The mean run time for gabiqsort.lmem is over four (4.41) standard deviations (0.61) away from the mean run time for gabiqsort.lpage. However the change in magnitude in the mean run time is only approximately four tenths of one percent. As with the AIX sort tests, we see that system time is higher, but user time is lower for the technical large page code.

4.1.6 Running both tests again within an affinity LPAR (ALPAR)

The AIX sort and gabiqsort tests were run in a normal LPAR. Would the numbers have been any different if the tests had been run in an affinity LPAR (ALPAR)? We decided to find out. We reconfigured the 4-way LPAR as a 4-way ALPAR, and allocated 1 GB of the memory to technical large pages. Unfortunately, we were not able to hold the physical memory constant at 4 GB because of the memory book configuration in the p690. It had two 16 GB memory books, so we had to go with 16 GB in the ALPAR. We then repeated the same sequence of 40 tests with both the AIX sort program and the gabiqsort program.

The results of the AIX sort test runs are summarized in Table 4-3.

Table 4-3 Technical large page (with ALPAR) AIX sort results

Program: AIX sort				
sort.normal				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	117.99	118.75	117.44	0.40
User	117.03	117.72	116.45	0.36
Sys	0.42	0.58	0.28	0.08
sort.lmem				
Real	118.02	118.84	117.49	0.46
User	117.08	117.84	116.56	0.40
Sys	0.40	0.61	0.24	0.11
sort.lpage				
Real	117.94	118.47	117.64	0.30
User	116.14	116.68	115.80	0.31
Sys	1.13	1.24	1.04	0.05

These results look very similar to those from the runs done in the normal LPAR (see Table 4-1 on page 86). Again, we see that the overall run times for all three runs show no significant differences. We also see the same pattern of increased system time and decreased user time for the technical large page runs.

So, running in an ALPAR did not change the relative performance between the three different executables. However, the run times for all three are higher than they were for their counterparts running in an LPAR. Does this mean that performance is worse when using ALPARs? Not necessarily. Remember that the ALPAR had 16 GB of memory, while the LPAR had only 4 GB. So, the difference in run times may be due to the change in memory size rather than the change in partition type. Let us see what happened with the next set of runs.

The results of the gabiqsort test runs are summarized in Table 4-4.

Table 4-4 Technical large page (with ALPAR) gabiqsort results

Program: gabiqsort				
gabiqsort.lmem				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	671.61	672.75	670.59	0.71
User	667.83	668.44	667.13	0.31
Sys	2.71	3.68	2.00	0.49
gabiqsort.lpage				
Real	668.93	670.36	668.14	0.82
User	664.80	665.70	664.34	0.39
Sys	3.62	4.44	3.21	0.49

What we see here is that these numbers are almost identical to those of the corresponding runs done in the normal LPAR (see Table 4-2 on page 92). This would seem to indicate that for this particular code, affinity LPARs provide no benefit over normal LPARs. It is also interesting that the additional memory in the ALPAR did not make a difference either.

4.1.7 Three steps forward, one step back

What still remains is the nagging question of why the run times changed for the AIX sort when running in an affinity LPAR as opposed to a normal LPAR. We eliminated the additional memory as a factor in the equation by reconfiguring the normal LPAR to match the affinity LPAR (that is, 4-way and 16 GB) and rerunning the tests.

The results of the AIX sort test runs are summarized in Table 4-5.

Table 4-5 Technical large page (with 16 GB) AIX sort results

Program: AIX sort				
sort.normal				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	117.71	118.80	117.30	0.46
User	116.77	117.77	116.33	0.43
Sys	0.39	0.60	0.27	0.08
sort.lmem				
Real	118.00	118.73	117.31	0.46
User	117.08	117.78	116.33	0.46
Sys	0.38	0.56	0.20	0.08
sort.lpage				
Real	118.14	118.82	117.57	0.47
User	116.27	117.00	115.64	0.43
Sys	1.20	1.45	1.00	0.12

These results look very similar to those from the runs done in the ALPAR (see Table 4-3 on page 93). Again, we see that the overall run times for all three show no significant differences. We also see the same pattern of increased system time and decreased user time for the technical large page runs.

From these results, it appears that the observed differences in the run times between the first runs in the normal LPAR and the second runs in the affinity LPAR were due to the increase in memory from 4 GB to 16 GB rather than the change of partition type form LPAR to ALPAR. Let us see what happened with the next set of runs.

The results of the gabiqsort test runs are summarized in Table 4-6 on page 96.

Table 4-6 Technical large page (with 16 GB) gabiqsort results

Program: gabiqsort				
gabiqsort.lmem				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	671.64	672.54	670.75	0.58
User	667.91	668.43	667.27	0.27
Sys	2.68	3.51	2.06	0.45
gabiqsort.lpage				
Real	668.42	670.24	668.00	0.40
User	664.60	665.46	664.25	0.22
Sys	3.30	4.28	3.22	0.23

What we see here is that these numbers are almost identical to those of the corresponding runs done in the other two test scenarios (see Table 4-2 on page 92 and Table 4-4 on page 94). So, again, it appears that for this particular code, there is no benefit to running it in an ALPAR.

However, the nagging question still remains about why the numbers from the first run of the AIX sort are so different from those from the subsequent runs. We tried one more test. This time we set the LPAR back to the original 4 GB to see if we could reproduce the first numbers.

The results of the AIX sort test runs are summarized in Table 4-7.

Table 4-7 Technical large page (with 4 GB) AIX sort results

Program: AIX sort				
sort.normal				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	117.75	118.50	117.37	0.34
User	116.80	117.44	116.39	0.31
Sys	0.40	0.62	0.23	0.11
sort.lmem				
Real	117.78	118.54	117.41	0.36
User	116.84	117.58	116.43	0.31

Program: AIX sort				
Sys	0.40	0.64	0.28	0.09
sort.lpage				
Real	117.99	118.49	117.59	0.24
User	116.12	116.42	115.76	0.20
Sys	1.19	1.36	1.06	0.07

The results from these runs did not reproduce the results from the first ones (see Table 4-1 on page 86). In fact, they look very much like the results from the latter runs. So, what is going on here?

Before discussing the AIX sort results further, let us take a quick look at the gabiqsort results. They are summarized in Table 4-8.

Table 4-8 Technical large page (with 4 GB) gabiqsort results

Program: gabiqsort				
gabiqsort.lmem				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	671.24	672.52	670.89	0.36
User	667.80	668.37	667.37	0.22
Sys	2.37	3.35	2.02	0.29
gabiqsort.lpage				
Real	668.70	670.06	668.05	0.62
User	664.69	665.35	664.27	0.28
Sys	3.50	4.33	3.22	0.39

No surprises here. The numbers are as we expected.

So why were we not able to reproduce the numbers from the first run of the AIX sort tests, and why were the numbers different between that first run and subsequent ones? The quick answer is that we do not know, but here are some possible explanations. Remember that the AIX sort does most of its work with temporary files in /var/tmp. So there is a whole disk I/O piece that could be confounding things. The second possibility is that the first LPAR had very good locality of reference between the processors and memory that were allocated to the LPAR. However, we would have expected to see these same results (if not better results) with the ALPAR. In our opinion, the first possibility seems far more reasonable than the second, but, without further testing, it is nothing more than a somewhat educated guess.

4.1.8 Memory affinity

For the final set of test runs, we wanted to see if turning *memory affinity* on in an LPAR had any effect on the performance of either of the two sort codes. Memory affinity is enabled through the **vmtune** command, and, like technical large page, it requires the **bosboot** and **shutdown** commands afterwards (reboot). These steps are illustrated in Example 4-14.

Example 4-14 Preparing the environment for memory affinity

```
[c37f1rp05][/]> /usr/samples/kernel/vmtune -y 1

vmtune:  current values:

...
        -s               -n         -S         -L          -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0                0          0          64      16777216        0
...
    -Z                   -q                    -Q                -y
j2_nBufferPer  j2_minPageReadAhead  j2_maxPageReadAhead   memory_affinity
      512              2                    8                 0
...
number of valid memory pages = 1048573  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 16630     numperm=2.2% of real memory
number of compressed memory pages = 0   compressed=0.0% of real memory
number of client memory pages = 0       numclient=0.0% of real memory
# of remote pgs sched-pageout = 0       maxclient=79.9% of real memory

vmtune:  new values:

...
        -s               -n         -S         -L          -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0                0          0          64      16777216        0
...
    -Z                   -q                    -Q                -y
j2_nBufferPer  j2_minPageReadAhead  j2_maxPageReadAhead   memory_affinity
      512              2                    8                 1
...
number of valid memory pages = 1048573  maxperm=79.9% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 33260     numperm=4.4% of real memory
number of compressed memory pages = 0   compressed=0.0% of real memory
number of client memory pages = 0       numclient=0.0% of real memory
# of remote pgs sched-pageout = 0       maxclient=79.9% of real memory

[c37f1rp05][/]> bosboot -ad /dev/ipldevice

bosboot: Boot image is 13276 512 byte blocks.

[c37f1rp05][/]> shutdown -Fr
```

After the reboot, we verified (with the **vmtune** command) that both memory affinity and technical large page were still turned on, then launched our test suite for one final time.

The results of the AIX sort test runs are summarized in Table 4-9.

Table 4-9 Technical large page (with memory affinity) AIX sort results

Program: AIX sort				
sort.normal				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	117.72	118.01	117.25	0.15
User	116.79	117.11	116.35	0.17
Sys	0.39	0.73	0.29	0.10
sort.lmem				
Real	117.71	117.96	117.28	0.16
User	116.78	117.06	116.42	0.17
Sys	0.38	0.57	0.28	0.07
sort.lpage				
Real	117.84	118.02	117.52	0.14
User	115.95	116.21	115.69	0.14
Sys	1.22	1.44	1.05	0.11

These results look very similar to the ones that were generated from running in the same LPAR without memory affinity (see Table 4-7 on page 96). Let us see what happened with the gabiqsort runs. They are summarized in Table 4-10 on page 100.

Table 4-10 Technical large page (with memory affinity) gabiqsort results

Program: gabiqsort				
gabiqsort.lmem				
Time	**Mean**	**Max**	**Min**	**Std. Dev.**
Real	671.87	672.75	670.89	0.59
User	667.96	668.33	667.46	0.22
Sys	2.85	3.66	2.13	0.50
gabiqsort.lpage				
Real	668.43	669.89	668.05	0.38
User	664.56	665.21	664.26	0.19
Sys	3.35	4.19	3.25	0.22

Again, no significant changes here compared with the previous run in the same LPAR without memory affinity turned on (see Table 4-8 on page 97).

Like ALPARs, memory affinity does not appear to affect the performance of these codes either positively or negatively. No big surprises here.

The reason that the results were no big surprise is that it was really a trick question. Even though you can use **`vmtune -y 1`** to enable memory affinity in an LPAR, it has no effect. In other words, it does nothing. Here is why.

AIX requires processor and memory topology data from the hardware to support memory affinity. The hardware topology data is not available on all machines or in all environments. On a pSeries 690, topology data is available when the system is configured in Full System Partition (SMP) mode, but it is not available when the system is configured in LPAR mode. That is why we saw no differences in the results of these test runs compared to those run in an LPAR without memory affinity turned on.

4.1.9 Technical large page investigation conclusions

Our conclusions from this investigation are as follows:

- For codes that make heavy use of the heap and data segments, technical large page seems to provide some improvement in performance.
- None of the tests showed any advantage to using ALPARs over LPARs.
- None of the tests showed any advantage to having memory affinity turned on in LPAR mode. Note that memory affinity is not supported in LPAR mode.

- For a mixed workload with applications that tax different components of the system (for example, memory, CPU, disk, and network), it probably does not make sense to use either ALPARs or technical large page.
- Our technical large page recommendation is that you really need to have a compelling reason (and data to back it) for using it. It is not a panacea for performance problems. Rather, it is a very specialized feature targeted for specific applications in limited application domains.
- Our ALPAR recommendation is that if you are planning to configure your pSeries 690 with either 4-way or 8-way homogeneous LPARs, then go ahead and use ALPARs. At the very least, they will save you some administrative time during configuration.

We believe the primary value of this investigation comes not from the final conclusions but more from the process itself. To that end, here are our final thoughts on the process itself:

- Performance and tuning is an iterative process, and designing valid performance tests is non-trivial. We began with a simple plan. It seemed so easy and so straightforward on paper. Unfortunately, as you have seen, in practice there were an amazing amount of blind alleys and missteps, each requiring a revision to the plan and yet another iteration of the tests.
- Setting up and executing performance tests is very time consuming. This investigation took one person a full week of dedicated work to set up and execute. Each set of runs alone took over 19 hours to complete, and this was with very trivial codes.
- Each iteration provided 40 data points for each class of time (real, user, and sys). Is this a large enough sample from which to draw valid conclusions? Well, it is better than using one or two data points, but we would have been more comfortable with something like 100 data points per class per iteration.
- Even when you try to change just one variable at a time, it is terribly easy for confounding variables to sneak into your well-planned tests. Once there, they wreak havoc on the validity of your results.

So, finally, begin by trusting the performance experts until you can prove otherwise, and start with the recommendations for tuning and feature settings from IBM and the software vendors first. Use these as your baseline for further investigations.

4.2 Tivoli Storage Manager (TSM) investigations

Note: This is not a book about the performance of the Tivoli Storage Manager (TSM). Rather, TSM was chosen for these investigations because it is a commonly-used commercial application.

We wanted to see if the TSM (server and client) could take advantage of technical large page through the shared memory communication protocol during backup operations. The use of shared memory protocol required that the server and client were on the same LPAR.

The test environment consisted of a single 1-way LPAR with 1.7 GB of memory, one 18 GB hard disk, and the software levels shown in Example 4-15. The TSM server was configured to use only disk storage pools.

Example 4-15   Installed software on TSM environment

```
****************************************************
Software Levels As Of: Tue Jun 18 10:11:24 EDT 2002
****************************************************
AIX:                 bos.mp            5.1.0.27
PSSP:                ssp.basic         3.4.0.7
LoadLeveler:
GPFS:
RVSD:
VSD:
RSCT:                rsct.basic.rte    2.2.1.0
POE:
PESSL:
ESSL:
FORTRAN:
PERL:                perl.rte          5.6.0.0
C:                   xlC.rte           5.0.2.1

# lslpp -l tivoli.tsm.server.com
Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  tivoli.tsm.server.com      5.1.1.0  COMMITTED  Tivoli Storage Manager Server
                                                 common services
Path: /etc/objrepos
  tivoli.tsm.server.com      5.1.1.0  COMMITTED  Tivoli Storage Manager Server
                                                 common services
# lslpp -l tivoli.tsm.client.ba.aix51.64bit.common
Fileset                      Level  State      Description
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
```

```
  tivoli.tsm.client.ba.aix51.64bit.common
                               5.1.1.0  COMMITTED  TSM Client - Backup/Archive
                                                   Common Files
```

We performed the following set of tests:

- With technical large page disabled:
  - Backed up one big file (total 1 GB)
  - Backed up 2048 files (total 1 GB)
- With technical large page enabled (with 32 and 64 pages):
  - Backed up one big file (total 1 GB)
  - Backed up 2048 files (total 1 GB)

The two metrics of interest from these tests were:

- The aggregate data transfer rate (KB/sec) from the **`dsmc`** command. (The **`dsmc`** command is the command line backup/archive client interface.)
- The real, user, and sys times from the **`timex`** command.

4.2.1 TSM environment without large page

The data from the first series of tests is shown in Table 4-11.

Table 4-11 TSM backup with vmtune -L 0

Test results without technical large page (`vmtune -L 0`)			
Aggregate data transfer rate (KB/sec)	**`timex`**		
	real	user	sys
Backup of one big file (1 GB)			
8,508.13	123.71	1.06	4.07
20,964.62	50.44	0.98	1.23
19,778.15	53.54	1.03	1.41
21,392.93	49.54	0.81	1.15
20,158.5	53.57	0.84	1.54
Backup of many small files (1023 * 1 MB + 1024 * 1 KB)			
8,571.30	123.41	1.50	5.08
11,252.91	94.75	1.52	3.12

Test results without technical large page (`vmtune -L 0`)			
12,781.43	82.46	1.32	1.58
12,047.29	88.61	1.33	1.46
13,974.16	76.47	1.29	1.62

In both cases the first part of the backup took more time. This was due to the fact that these initial file pages had to be loaded from disk. After that, they were in the cache memory used by the JFS. Example 4-16 shows a portion of the **`vmtune`** output taken after the backup had completed. Notice the high value for the numperm parameter.

Example 4-16 The vmtune listing from after the backups

```
...
number of valid memory pages = 458749   maxperm=79.8% of real memory
maximum pinable=80.0% of real memory    minperm=20.0% of real memory
number of file memory pages = 284065    numperm=64.1% of real memory
number of compressed memory pages = 0   compressed=0.0% of real memory
number of client memory pages = 0       numclient=0.0% of real memory
# of remote pgs sched-pageout = 0       maxclient=79.8% of real memory
```

4.2.2 TSM environment with technical large page

The TSM server process uses the root account by default. Rather than change TSM, we opted instead to change the root account to use technical large page. The steps used to change the account are shown in Example 4-17.

Example 4-17 Enabling root to use technical large page

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE root

# grep -p root /etc/security/limits

root:
        fsize = 2097151
        core = 2097151
        cpu = -1
        data = -1
        rss = -1
        stack = -1
        nofiles = 2000
```

For the first test, we set up the LPAR with 32 technical large pages (using **`vmtune -L 32`**). The procedure we used is shown in Example 4-18 on page 105.

Example 4-18 Enabling LPAR with 32 technical large pages

```
# /usr/samples/kernel/vmtune -g 16777216 -L 32
...
        -s               -n         -S         -L         -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0                0          0          32      16777216        0
...
# bosboot -ad /dev/ipldevice
# shutdown -Fr
```

After the reboot, we created a modified TSM server binary (dsmserv) to use technical large page, set the LDR_CNTRL environment variable to mandatory mode, and fired up the server, as shown in Example 4-19.

Example 4-19 Enabling TSM to use technical large page

```
# cd /usr/tivoli/tsm/server/bin
# cp dsmserv dsmserv.lp
# usr/ccs/bin/ldedit -blpdata dsmserv.lp
# export LDR_CNTRL=LARGE_PAGE_DATA=M
# ./dsmserv.lp
```

With the modified TSM server running, we used **svmon** to verify that the modified TSM server was really using technical large pages. This is shown in Example 4-20.

Example 4-20 Modified TSM server verification with svmon

```
# ps -ef | grep dsm
    root 11996  9934   0 14:33:32  pts/0  0:01 ./dsmserv.lp
# svmon -P 11996 -r

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
   11996 dsmserv.lp       75860    67597      791    73141      N     Y     Y

     PageSize      Inuse        Pin       Pgsp    Virtual
         4 KB      10324       2061        791       7605
        16 MB         16         16          0         16

    Vsid      Esid Type Description              LPage Inuse   Pin Pgsp Virtual
   345da         3 work shmat/mmap                   Y  65536 65536    0 65536
       0         0 work kernel seg                   -   4862  2018  791  4862
                   Addr Range: 0..25224
   2c016         d work shared library text          -   2204     0    0  2204
                   Addr Range: 0..60123
   343fa         - pers /dev/tivolilv:17             -   1298     0    -     -
                   Addr Range: 0..24832
```

```
2a5d5         1 pers code,/dev/hd2:163919        -   1155     0     -     -
                Addr Range: 0..2284
205d0         2 work process private             -    310     2     0   310
                Addr Range: 0..293 : 65304..65535
263f3         - pers /dev/tivolilv:18            -    265     0     -     -
                Addr Range: 0..25600
2c5d6         - work                             -    140    41     0   140
                Addr Range: 0..49377
325d9         f work shared library data         -     89     0     0    89
                Addr Range: 0..1955
305d8         - pers /dev/hd2:163993             -      1     0     -     -
                Addr Range: 0..0
265d3         4 work shmat/mmap                  -      0     0     0     0
1844c         - pers /dev/hd2:163960             -      0     0     -     -
26453         - pers /dev/tivolilv:21            -      0     0     -     -
22451         - pers /dev/hd2:163961             -      0     0     -     -
24452         - pers /dev/tivolilv:20            -      0     0     -     -
```

Table 4-12 shows the results from the 32 technical large page tests.

Table 4-12 Test results with 32 technical large pages

Tests results with technical large page (`vmtune -g 16777216 -L 32`)			
Aggregate data transfer rate (KB/sec)	**`timex`**		
	real	user	sys
Backup of one big file (1 GB)			
8,437.54	125.94	0.77	5.03
17,179.08	62.67	0.58	2.51
19,411.73	56.58	0.79	2.40
21,392.23	51.55	0.73	2.24
19,777.96	54.52	0.78	2.28
Backup of many small files (1023 * 1 MB + 1024 * 1 KB)			
8,587.28	123.82	1.04	6.20
13,790.12	77.49	1.21	2.64
12,627.75	85.52	1.10	2.54
12,330.53	86.58	1.08	2.57
13,789.84	78.49	1.25	2.54

Before discussing the results from the tests with 32 technical large pages, let us take a look at the tests with 64 technical large pages. Example 4-21 shows the procedure that was used to reconfigure the LPAR with 64 technical large pages.

Example 4-21 Enabling LPAR for 64 technical large pages

```
# /usr/samples/kernel/vmtune -g 16777216 -L 64
...
       -s              -n        -S        -L         -g          -h
sync_release_ilock  nokilluid  v_pinshm  lgpg_regions  lgpg_size strict_maxperm
        0               0         0         64       16777216        0
...
# bosboot -ad /dev/ipldevice
# shutdown -Fr
```

The results of the backup tests in the 64 technical large page environment are shown in Table 4-13.

Table 4-13 Test results with 64 technical large pages

Test results with technical large page (`vmtune -g 16777216 -L 64`)			
Aggregate data transfer rate (KB/sec)	**`timex`**		
	real	user	sys
Backup of one big file (1 GB)			
9,613.37	110.67	0.84	5.05
8,379.64	127.03	0.82	5.01
8,718.95	122.27	0.65	5.52
8,437.70	126.12	0.73	5.15
9,156.07	116.58	0.77	5.00
Backup of many small files (1023 * 1 MB + 1024 * 1 KB)			
8,734.29	121.88	1.03	5.90
8,116.88	131.31	1.15	5.50
7,762.78	137.15	1.35	5.78
8,733.32	121.91	1.00	5.95
7,993.72	133.26	1.21	5.81

4.2.3 TSM and technical large page conclusions

Example 4-20 on page 105 showed that the TSM server was using technical large pages for a portion of its process space, and Example 4-22 shows that the TSM client was also using technical large pages for a portion of its process space.

Example 4-22 Modified TSM client verification with svmon

```
svmon -P 5476 -r

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd LPage
    5476 cli             220046    67568      791    72041      Y     Y     Y

     PageSize       Inuse        Pin       Pgsp    Virtual
         4 KB      154510       2032        791       6505
        16 MB          16         16          0         16

    Vsid      Esid Type Description              LPage  Inuse Pin Pgsp Virtual
   20670         - pers /dev/lvtest:19               - 147560     0    -     -
                   Addr Range: 0..147559
   28634        11 work text data BSS heap           Y  65536 65536    0 65536
       0         0 work kernel seg                   -   4862  2019  791  4862
                   Addr Range: 0..25224
...
```

Neither the TSM server nor the TSM client performed better with technical large pages. Remember that the LPAR only had 1.7 GB of memory, so 64 technical large pages took up approximately 60 percent of the available memory. Also, remember that technical large pages use pinned memory, which means that this memory is not available for anything else. TSM performance is dominated by I/O, not by computation, and a large amount of memory is needed for file pages, as we saw in the numperm output in Example 4-16 on page 104. By configuring such a large pool for technical large pages, we were effectively starving the system of file pages, and the VMM performance suffered because it had to continually search for free pages to be used to cache file pages. This can be seen in the **vmstat** output shown in Example 4-23.

Example 4-23 The vmstat output during backup

```
# vmstat 5
kthr     memory             page              faults        cpu
----- ----------- ------------------------ ------------ -----------
 r  b    avm   fre  re  pi  po   fr    sr  cy  in   sy   cs us sy id wa
 1  1 314412   779   0   0   0  767  2018   0 378 1496  435  2  9 54 35
 1  0 314905   128   0   0   0   28    41   0 261  586  128  0 19 78  3
 0  1 315198   120   0   0   0 2372 28508   0 692 1873 1266  4 18  0 78
```

```
0   1 315198   120   0   0   0 2134 11373   0 653 1756 1215  3 16  0 81
1   1 315198   128   0   0   0 2175  3416   0 664 1769 1271  3 13  0 84
2   1 315198   120   0   0   0 2428  3038   0 691 1886 1376  3 13  0 83
0   1 315198   128   0   0   0 1724  2153   0 586 1542 1046  1  7  0 92
...
```

So, for these tests, enabling technical large pages not only did not help TSM performance, it actually hurt it. Our recommendation is that you do not use technical large pages with TSM (server or client).

4.2.4 TSM and SP Switch2 communication

TSM environments typically use the network to communicate between the server and the client. For the next set of tests, we created two LPARs, one for the TSM server and the other for the TSM client, and we configured them to use the SP Switch2 for their communication path. Both LPARs were 2-way with 4 GB of memory, and the software levels were the same as what was shown in Example 4-15 on page 102.

For the first 1 GB backup test, we used the default settings for the dsm.sys configuration file as well as the css0 tunables. As you can see in Table 4-14, the performance was terrible. (The data shown is an average from a series of **dsmc** command outputs.)

Table 4-14 Backup performance for 1 GB with default settings

Size of data	Network transfer rate (KB/sec)	Aggregate transfer rate (KB/sec)	Elapsed time (minutes)
1 GB	320	315	54:00

The SP Switch2 is a high speed network with a large Maximum Transmission Unit (MTU) size. (The default for css0 is 65504). Because of this, systems (LPARs) that use the SP Switch2 as an interconnect can spend a lot of time buffering data prior to sending it. That is what happened here.

So, for our next test, we wanted to change this buffering behavior. We could do this by changing the tcp_nagle_limit to 0 with the **no** command, but this is a global setting that applies to all networks. Instead, we choose to use the TCPNodelay parameter in the dsm.sys file so that we could limit the scope. The modified dsm.sys file is shown in Example 4-24.

Example 4-24 The dsm.sys configuration file

```
SErvername  server_a
   NODEName          c37f2rp06
   COMMmethod         TCPip
   TCPPort            1500
   TCPServeraddress   9.114.189.85
   PASSWORDAccess     GENERATE

   TCPNodelay         Yes
```

Setting TCPNodelay to Yes forces the client to send all transactions to the server without buffering them first. In other words, the client does not have to wait until the TCP send buffer is full before transmitting the data.

The **dsmc** average transfer rates for this test are shown in Table 4-15.

Table 4-15 Backup performance for 1 GB with TCPNodelay Yes

Size of data	Network transfer rate (KB/sec)	Aggregate transfer rate (KB/sec)	Elapsed time (minutes)
1 GB	20900	20100	00:52

In an environment like ours with ample memory and a fast interconnect, you can use higher values for the tcpbuffsize and tcpwindowsize. We tried this, as shown in Example 4-25.

Example 4-25 The dsm.sys file with modified TCP buffer size

```
SErvername  server_a
   NODEName           c37f2rp06
   * COMMmethod       Sharedmem
   COMMmethod         TCPip
   TCPPort            1500
   TCPServeraddress   9.114.189.85
   PASSWORDAccess     GENERATE

   TCPNodelay         Yes
   tcpbuffsize        512
   tcpwindowsize      2048
```

The **dsmc** average transfer rates for this test are shown in Table 4-16.

Table 4-16 Backup performance for 1 GB with larger tcpbuffsize

Size of data	Network transfer rate (KB/sec)	Aggregate transfer rate (KB/sec)	Elapsed time (minutes)
1 GB	21300	22300	00:49

Further analysis of these test results showed that the bottleneck was now the storage pool used by TSM. As you will remember, the TSM storage pool was configured on the internal 18 GB disk in the TSM server.

This is not a book about TSM performance. It is a book about the performance of the pSeries 690 in a Cluster 1600. With that in mind, we set out to eliminate the storage pool bottleneck, and relief came from a somewhat unusual avenue.

AIX supports the creation and use of ramdisks. A ramdisk is a portion of memory that is made to look and act like a physical disk. These virtual disks can be very useful for applications (like sort routines and compilers) that are I/O intensive to short-lived, temporary files. Production TSM does not fit in this category, but this is not production TSM. The point here is to eliminate the storage pool as the bottleneck, and use of a ramdisk seems like a logical way to do that.

So, for our next test, we created a ramdisk, put a file system on it, and modified the TSM server to use it as the storage pool. The procedure we used can be found in Appendix A, "Scripts" on page 147. We also created a ramdisk and a file system on the TSM client and used it to house the data that was to be backed up. The css0 parameters and dsm.sys file remained the same as they were in the previous test.

The **dsmc** average transfer rates for this test (using memory-to-memory across the interconnect) are shown in Table 4-17.

Table 4-17 Backup performance for 1 GB with ramdisks

Size of data	Network transfer rate (KB/sec)	Aggregate transfer rate (KB/sec)	Elapsed time (minutes)
1 GB	143000	115000	00:09

Remember, this is by no means a production TSM configuration. We simply wanted to figure out the maximum performance that we could expect across the SP Switch2 with the SP Switch2 PCI Attachment Adapter. In a real TSM production environment with many tape drives, many paths or multiple storage pools on separate disks, and many clients doing backups at the same time, you can expect your performance bottleneck to be I/O and not communications.

For our next test, we changed the css0 spoolsize and rpoolsize tunable parameters with the **chgcss** command to their maximums, which are now 32 MB (33554432) each. The test results did not show any significant improvement and will not be shown here. We suspect that these parameter changes would have helped performance if many clients were trying to back up large amounts of data at the same time.

Finally, we tried reducing the tcpwindowsize in the dsm.sys file. If you remember, we had set it and the tcpbuffsize to their maximum values (that is, `tcpbuffsize=512` and `tcpwindowsize=2048`) for a previous test run. (See Table 4-16 on page 110.) We wanted to see if setting the tcpwindowsize to 1024 would make a difference.

The **dsmc** average transfer rates for this test are shown in Table 4-18.

Table 4-18 Backup performance for 1 GB with tcpwindowsize at 1024

Size of data	Network transfer rate (KB/sec)	Aggregate transfer rate (KB/sec)	Elapsed time (minutes)
1 GB	180000	131000	00:08

This dsm.sys change had no significant impact on the performance from the previous test. We also did additional tests with different dsm.sys values (which will not be shown here), and none of them produced any significant performance improvements. In other words, it appears that we have found the maximum throughput performance numbers for this investigation.

4.2.5 TSM and SP Switch2 conclusions

The performance of the TSM backup across the SP Switch2 interconnect is limited by the I/O to the storage device on the TSM server. In our lab environment, we had an 18 GB internal disk on which we were able to achieve a maximum transfer rate of 20 MB/sec with the following parameter settings:

- Set the tcpbuffsize to 512 in the dsm.sys file.
- Set the tcpwindowsize to 1024 in the dsm.sys file.
- Set css0 spoolsize to 33554432 (32 MB) with the **chgcss** command.
- Set css0 rpoolsize to 33554432 (32 MB) with the **chgcss** command.

Important: Even though it may be tempting to use a ramdisk in a production TSM environment, do not do it. We used one in our investigations for the sole purpose of trying to find the maximum throughput rate across the interconnect. We would never think to use one in a production TSM environment. If and when the system goes down, all data on the ramdisk is lost. This is but one of a number of RAS problems associated with their use.

4.3 IP vs. US investigation

> **Note:** This investigation is primarily of interest for folks from the scientific and technical community. Few if any business applications use US protocol across the interconnect, so folks from the commercial community may want to skip this section.

For this investigation, we wanted to see if we could see any differences between running MPI codes with US communication over the SP Switch2 compared to these same codes running with IP communication over the SP Switch2. The MPI codes that we used were the three parallel sample programs from the *Parallel Environment for AIX: Hitchhiker's Guide*, SA22-7424. This guide and a tar file with the sample programs can be downloaded from:

http://www.ibm.com/servers/eserver/pseries/library/sp_books/pe.html

We picked these codes for our tests because they are readily available for anyone to use, are well documented, and are large enough to be useful on a small system.

The test system consisted of two 4-way, 8 GB normal LPARs on two separate pSeries 690s in a Cluster 1600. The interconnect fabric was a single-plane SP Switch2, and each of the LPARs had a single SP Switch2 PCI Attachment Adapter. Each of the MPI codes used nine MPI tasks with five on one LPAR and four on the other LPAR. The software levels for the two LPARs are shown in Example 4-26.

Example 4-26 Software levels for IP vs. US investigation

```
***************************************************
Software Levels As Of: Thu Jun 20 08:41:42 EDT 2002
***************************************************
AIX:                 bos.mp              5.1.0.27
PSSP:                ssp.basic           3.4.0.8
LoadLeveler:         LoadL.full          3.1.0.6
GPFS:
RVSD:
VSD:
RSCT:                rsct.basic.rte      2.2.1.1
POE:                 ppe.poe             3.2.0.6
PESSL:
ESSL:
FORTRAN:             xlfrte              7.1.1.2
PERL:                perl.rte            5.6.0.0
C:                   xlC.rte             5.0.2.1
```

4.3.1 Setting up the environment for IP vs. US testing

First, we had to prepare the test environment. This involved:

- Increasing the switch send pool and receive pool sizes
- Checking the other network options for unusual settings
- Preparing the POE and LoadLeveler environments
- Creating the MPI test code executables
- Creating the wrapper and driver scripts for the tests

Increasing the switch send pool and receive pool sizes

The send pool (spool) and the receive pool (rpool) on the SP Switch2 with the SP Switch2 PCI Attachment Adapter can now each be 32 MB in size. This is double the size of what was available on the previous SP Switch fabric. Additionally, these size changes can now be made on the fly. To change them, use the **chgcss** command, as shown in Example 4-27.

Example 4-27 Changing the spoolsize and the rpoolsize

```
/usr/lpp/ssp/css/chgcss -l css0 -a spoolsize=33554432 -a rpoolsize=33554432
```

You can then verify the changes with the **lsattr -El css0** command, as shown in Example 4-28.

Example 4-28 Verifying the spoolsize and the rpoolsize

```
[c37f1rp08][/]> lsattr -El css0
adapter_memory 0xf1000000              Adapter memory address       False
adapter_size   0x00800000              Adapter memory size          False
sdram_start    0xf0000000              SDRAM memory address         False
sdram_size     0x01000000              SDRAM memory size            False
TOD_address    0xf1800000              TOD address                  False
win_poolsize   1107296256              Total window memory pool size True
win_maxsize    16777216                Maximum window memory size    True
win_minsize    524288                  Minimum window memory size    True
int_priority   3                       Interrupt priority           False
int_level      2853                    Bus interrupt level          False
spoolsize      33554432                Size of IP send buffer        True
rpoolsize      33554432                Size of IP receive buffer     True
khal_spoolsize 524288                  Size of KHAL send buffer      True
khal_rpoolsize 524288                  Size of KHAL receive buffer   True
adapter_status css_ready               Configuration status         False
diags_prog                             Diagnostic program            True
ucode_version  1                       Micro code version            True
ucode_name     /etc/microcode/cor_ucode Micro code name              True
window0-15     VSD AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
```

```
 AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners              True
window16-31  AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
 AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners              True
window32-47  AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
 AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners              True
window48-63  AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL
 AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL AVAIL window owners              True
window64-65  AVAIL AVAIL                  window owners              True
driver_debug   0                        Device Driver Debug          True
ip_chksum      off                      if_cl checksum               True
ip_debug       0                        if_cl debug level            True
proto_debug    off                      proto debug                  True
```

Checking the other network options for unusual settings

The other network settings were left at default and are shown in Example 4-29.

Example 4-29 Network settings for IP vs. US investigation

```
[c37f1rp08][/]> no -a
         extendednetstats = 0
                  thewall = 1048576
               sockthresh = 85
                   sb_max = 1310720
                somaxconn = 1024
      clean_partial_conns = 0
        net_malloc_police = 0
                  rto_low = 1
                 rto_high = 64
                rto_limit = 7
               rto_length = 13
           inet_stack_size = 16
              arptab_bsiz = 7
                arptab_nb = 25
               tcp_ndebug = 100
                   ifsize = 8
                 arpqsize = 12
                 ndpqsize = 50
             route_expire = 1
       send_file_duration = 300
                 fasttimo = 200
          routerevalidate = 0
         dgd_packets_lost = 3
           dgd_retry_time = 5
            dgd_ping_time = 5
              passive_dgd = 0
                  sodebug = 0
                nbc_limit = 786432
            nbc_max_cache = 131072
```

```
                nbc_min_cache = 1
                     nbc_pseg = 0
               nbc_pseg_limit = 4194296
                     strmsgsz = 0
                     strctlsz = 1024
                     nstrpush = 8
                    strthresh = 85
                    psetimers = 20
                  psebufcalls = 20
                   strturncnt = 15
                 pseintrstack = 12288
                    lowthresh = 90
                    medthresh = 95
                     psecache = 1
              subnetsarelocal = 1
                       maxttl = 255
                    ipfragttl = 60
              ipsendredirects = 1
                 ipforwarding = 1
                      udp_ttl = 30
                      tcp_ttl = 60
                   arpt_killc = 20
                tcp_sendspace = 65536
                tcp_recvspace = 65536
                udp_sendspace = 32768
                udp_recvspace = 65536
           tcp_bad_port_limit = 0
           udp_bad_port_limit = 0
               rfc1122addrchk = 0
               nonlocsrcroute = 1
                tcp_keepintvl = 150
                 tcp_keepidle = 14400
                    bcastping = 0
                     udpcksum = 1
                  tcp_mssdflt = 1448
              icmpaddressmask = 0
                 tcp_keepinit = 150
   ie5_old_multicast_mapping = 0
                      rfc1323 = 1
             pmtu_default_age = 10
    pmtu_rediscover_interval = 30
            udp_pmtu_discover = 0
            tcp_pmtu_discover = 0
                    ipqmaxlen = 100
           directed_broadcast = 0
            ipignoreredirects = 0
               ipsrcroutesend = 1
               ipsrcrouterecv = 1
            ipsrcrouteforward = 1
           ip6srcrouteforward = 1
                   ip6_defttl = 64
                    ndpt_keep = 120
               ndpt_reachable = 30
                 ndpt_retrans = 1
                   ndpt_probe = 5
                    ndpt_down = 3
                ndp_umaxtries = 3
                ndp_mmaxtries = 3
                    ip6_prune = 2
                ip6forwarding = 0
                  multi_homed = 1
                     main_if6 = 0
                   main_site6 = 0
                  site6_index = 0
                     maxnip6q = 20
              llsleep_timeout = 3
                 tcp_timewait = 1
            tcp_ephemeral_low = 32768
           tcp_ephemeral_high = 65535
            udp_ephemeral_low = 32768
           udp_ephemeral_high = 65535
                     delayack = 0
                delayackports = {}
                         sack = 0
                     use_isno = 1
                  tcp_newreno = 1
              tcp_nagle_limit = 65535
                      rfc2414 = 0
              tcp_init_window = 0
                      tcp_ecn = 0
         tcp_limited_transmit = 1
            icmp6_errmsg_rate = 10
                 tcp_maxburst = 0
```

Preparing the POE and LoadLeveler environments

Even though we were not using LoadLeveler to submit these test runs in batch mode, it is still necessary when running parallel codes through POE interactively. The LoadLeveler setup that we used was basically the default one, the setup for which is well documented in the *LoadLeveler for AIX: Installation Memo*, GI11-2819. The only point that we want to emphasize here is the importance of running the **`llextSDR`** command to create the machine and adapter stanzas for the LoadL_admin file. Do not just take a LoadL_admin file from another system and change the names. The switch type (css_type) changes with the SP Switch2 PCI Attachment Adapter, and, if it is not correct in your LoadL_admin file, you end up with these very strange 0028-601 error messages when you try to run the codes. Example 4-30 shows the LoadL_admin machine and adapter stanzas for one of our test LPARs.

Example 4-30   LoadL_admin machine and adapter stanzas

```
c37f1rp08.ppd.pok.ibm.com: type = machine
        central_manager = true
        schedd_host = true
#       machine_mode = batch
#       resources = ConsumableCpus(8)
        adapter_stanzas = c37sn08.ppd.pok.ibm.com c37f1rp08.ppd.pok.ibm.com
        spacct_excluse_enable = false
        alias = c37sn08.ppd.pok.ibm.com

c37sn08.ppd.pok.ibm.com: type = adapter
        adapter_name = css0
        network_type = switch
        interface_address = 9.114.189.72
        interface_name = c37sn08.ppd.pok.ibm.com
        switch_node_number = 7
        css_type = SP_Switch2_PCI_Attachment_Adapter

c37f1rp08.ppd.pok.ibm.com: type = adapter
        adapter_name = en0
        network_type = ethernet
        interface_address = 9.114.189.8
        interface_name = c37f1rp08.ppd.pok.ibm.com
```

For the POE setup, we created an initial hostfile (shown in Example 4-31).

Example 4-31   POE hostfile for IP vs. US investigations

```
[c37f1rp08][/ptmp/mgenty]> cat hostfile
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
```

We also made sure there were no /etc/poe.limits files in either of the LPARs.

Finally, we created a shared file system between the two LPARs. The lab system was not set up for GPFS, so we went with NFS instead. We created the local file system on a third LPAR and NFS-mounted it to the other two. This was done in an attempt to keep the relative performance of the two “compute” LPARs as similar as possible.

Creating the MPI test code executables

We used three programs for this investigation, as shown in Table 4-19.

Table 4-19 IP vs. US test codes

Type	Name
MPI	inverse_parallel_enabled.c
MPI	inverse_parallel.c
MPI	series_parallel.c

The MPI codes were compiled with separate make files. The source code and make files are listed in Appendix B, “MPI sample programs” on page 159.

Creating the wrapper scripts for the tests

Next, we prepared the scripts for running the tests. We began by creating a series of three wrapper scripts for the test codes:

- One for all of the parallel (MPI) codes using IP to communicate across the switch
- One for all of the parallel (MPI) codes using US to communicate across the switch

Parallel IP wrapper

The wrapper script for the parallel (MPI) codes using IP is shown in Example 4-32.

Example 4-32 Parallel code (IP) wrapper script

```
[c37f1rp08][/ptmp/mgenty]> cat hgip.sh
#!/usr/bin/ksh
#
# Hitchhiker's Guide - IP.
#
export LOADL_INTERACTIVE_CLASS=parallel
export MP_INFOLEVEL=1
export MP_HOSTFILE=hostfile
export MP_CPU_USE=multiple
export MP_PGMMODEL=SPMD
export MP_ADAPTER_USE=shared
export MP_EUIDEVICE=css0
export MP_EUILIB=ip
export MP_PROCS=9
export MP_NODES=2
export MP_RESD=yes
#
echo " "
echo "Inverse_Parallel_Enabled"
echo " "
timex poe inverse_parallel_enabled
echo " "
echo "Inverse_Parallel"
echo " "
timex poe inverse_parallel
echo " "
echo "Series_Parallel"
echo " "
timex poe series_parallel
```

Parallel US wrapper

The wrapper script for the parallel (MPI) codes using US is shown in Example 4-33.

Example 4-33 Parallel code (US) wrapper script

```
[c37f1rp08][/ptmp/mgenty]> cat hgus.sh
#!/usr/bin/ksh
#
# Hitchhiker's Guide - US.
#
export LOADL_INTERACTIVE_CLASS=parallel
export MP_INFOLEVEL=3
export MP_HOSTFILE=hostfile
export MP_CPU_USE=multiple
export MP_PGMMODEL=SPMD
export MP_ADAPTER_USE=shared
export MP_EUIDEVICE=css0
export MP_EUILIB=us
export MP_PROCS=9
export MP_NODES=2
export MP_RESD=yes
#
echo " "
echo "Inverse_Parallel_Enabled"
echo " "
timex poe inverse_parallel_enabled
echo " "
echo "Inverse_Parallel"
echo " "
timex poe inverse_parallel
```

```
echo " "
echo "Series_Parallel"
echo " "
timex poe series_parallel
```

Notice that the only difference between the two parallel (MPI) code wrapper scripts is that the environment variable MP_EUILIB has been changed from `ip` to `us`. That is literally all that is needed to switch these codes from using IP protocol to using US protocol. In other words, the codes do not know nor do they care about how the communication is being handled across the interconnect.

Creating the driver scripts for the tests

Following the same model used in Section 4.1, "Technical large page investigation" on page 78, we went with a sample size of 40 runs of each code, and, as was also the case with that investigation, we were only interested in run times and not the actual output from the codes. To gather run times and discard code outputs, we created one driver script to run the tests.

Parallel driver

The driver script for the parallel (MPI) tests is shown in Example 4-34.

Example 4-34 Parallel (MPI) code driver script

```
[c37f1rp08][/ptmp/mgenty]> cat usip.sh
#!/usr/bin/ksh
#
# US vs IP Test Runs.
#

#
# US.
#
echo " "
echo "US Test Runs"
echo " "
for i in 1 2 3 4
do
  for j in 1 2 3 4 5 6 7 8 9 10
  do
    /ptmp/mgenty/hgus.sh | grep -E "Parallel|real|user|sys"
  done
done
#
# IP.
#
echo " "
echo "IP Test Runs"
echo " "
for i in 1 2 3 4
do
  for j in 1 2 3 4 5 6 7 8 9 10
  do
    /ptmp/mgenty/hgip.sh | grep -E "Parallel|real|user|sys"
  done
done
```

4.3.2 Running the tests for IP vs. US

Prior to running the runs, we made sure that there was nothing else running on the LPARs and that the Cluster 1600 itself was not being heavily used.

We also did a couple of single-iteration test runs of each code and each wrapper script to make sure that things were set up correctly. To check that the proper communications protocol was really being used, we changed the MP_INFOLEVEL environment variable to 3 in the parallel (MPI) wrapper scripts and examined the output. (That output was very verbose and will not be included here.)

One other thing we discovered during these initial test runs was that, unlike the technical large page investigation results, the only value of interest from the **timex** command for these runs was the real time. (This is where the communications happened.) The user and sys times provided no interesting data.

Important: The run-time results shown in all of the tables in this investigation represent the real time component of the **timex** command output for the runs.

The results of the parallel (MPI) code test runs using IP protocol are shown in Table 4-20.

Table 4-20 Results from the parallel (MPI) IP test runs: 4-way/8 GB

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	13.34	13.59	13.18	0.10
inverse_parallel	12.55	13.20	12.39	0.17
series_parallel	12.25	12.40	12.19	0.09

The results of the parallel (MPI) code test runs using US protocol are shown in Table 4-21 on page 123.

Table 4-21 Results from the parallel (MPI) US test runs: 4-way/8 GB

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	15.68	16.39	13.59	0.56
inverse_parallel	14.88	15.40	12.60	0.45
series_parallel	14.36	15.20	12.20	0.65

First, notice that there is a statistically significant difference between the run times of the parallel (MPI) codes going from IP to US. The IP run times are better, which is likely due to the new Jumbo Packet feature of IP over the switch that was covered in Chapter 3, “Features relevant to performance” on page 35. The other thing to notice is that the variation in the run times is much less with IP than it is with US. At this point, we do not know if this is significant or just random.

So IP is the better choice, right? Not always. Here is a complication. For our next test in this 4-way, 8 GB environment, we took a parallel code that is used by the Supercomputer Systems Group at the National Center for Atmospheric Research (NCAR) in Boulder, Colorado to identify and isolate CPUs that are exhibiting numerical instability. The code is a legitimate (but old) climate code known as the *Parallel Climate Model (PCM), Version 1*. The version that we used is the four MPI task version. The executable was built on a POWER3 system on May 15, 2000. So, needless to say, there were no optimizations in it for POWER4 or the Cluster 1600. We set it up to run 40 times with two tasks on each LPAR. The results are shown in Table 4-22.

Table 4-22 Results from the pcm_04 IP and US test runs: 4-way/8 GB

Code	Mean	Max	Min	Std. Dev.
pcm_04 (IP)	218.93	234.45	185.27	19.11
pcm_04 (US)	178.74	182.40	172.56	1.99

Notice that US outperformed IP by a significant margin on these test runs. Also notice that the variation in run times switched, again in favor of US. So, which should you use, IP or US? The answer really depends on the nature of your code. This may sound like bad news at first, but remember that the switch from IP to US is simply an environment variable setting change. Try both, measure the results (from multiple iterations), and then decide.

4.3.3 IP vs. US on larger LPARs

Next, we increased both LPARs from 4-way with 8 GB of memory to 8-way with 16 GB of memory. In the previous tests, one of the LPARs was running five MPI tasks, but that LPAR only had four CPUs. We wanted to see if there would be any improvement in run times if we used larger LPARs and did not oversubscribe the CPUs.

The results from the parallel (MPI) code runs using IP protocol are shown in Table 4-23.

Table 4-23 Results from the parallel (MPI) IP test runs: 8-way/16 GB

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	13.24	19.20	12.79	1.09
inverse_parallel	12.74	23.80	12.19	1.83
series_parallel	12.42	18.40	11.99	1.04

The results from the parallel (MPI) code runs using US protocol are shown in Table 4-24.

Table 4-24 Results from the parallel (MPI) US test runs: 8-way/16 GB

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	15.25	30.80	14.20	2.80
inverse_parallel	14.35	15.60	14.00	0.59
series_parallel	14.58	25.21	13.80	2.07

From these results, it appears that increasing the size of the LPARs did not significantly improve the run times over those from the smaller LPARs. However, a new oddity showed up. Take a look at the Max column in both tables. On most of the runs, there was at least one very large output. We do not have a good explanation for this. It may be due to the increase in size of the LPARs, or it may be due to other work that was occurring at the same time on the Cluster 1600. Additional data is needed to further isolate the cause or causes.

4.3.4 IP vs. US with different hostfile

Next, we wanted to see if rearranging the hostfile would have any effect on the run times. This was discussed in Chapter 3, “Features relevant to performance” on page 35. We did not change the LPAR definitions from the previous runs. They were both left as 8-way, 16 GB LPARs. The original and reorganized hostfiles are shown in Example 4-35.

Example 4-35 IP vs. US hostfiles

```
#
# Original hostfile
```

```
#
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com

#
# Reorganized hostfile
#
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
c37f2rp08.ppd.pok.ibm.com
```

The results from the parallel (MPI) code runs using IP protocol are shown in Table 4-25.

Table 4-25 Results from the parallel (MPI) IP test runs new hostfile

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	12.89	13.19	12.79	0.11
inverse_parallel	12.44	12.80	12.39	0.10
series_parallel	12.22	12.40	12.19	0.07

The results from the parallel (MPI) code runs using US protocol are shown in Table 4-26.

Table 4-26 Results from the parallel (MPI) US test runs new hostfile

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	13.60	14.60	12.79	0.83
inverse_parallel	13.26	14.40	12.40	0.81

Code	Mean	Max	Min	Std. Dev.
series_parallel	13.12	14.00	12.19	0.79

It appears that reorganizing the hostfile did make an improvement, especially in the US runs. Additionally, the differences between the run times for IP and US have narrowed quite a bit. Notice also that the large variation within the run times has once again disappeared. This lends credibility to the supposition that the anomalies were due to other things occurring on the Cluster 1600 rather than to the change in LPAR definitions.

4.3.5 IP vs. US with MP_SHARED_MEMORY=yes

Next, we wanted to find out if turning shared memory on would have any effect on the run times. By default, shared memory is turned off for POE jobs. To enable it, you simply have to set the MP_SHARED_MEMORY environment variable to yes. We did this in the IP and US wrapper scripts.

With shared memory turned on, the intra-LPAR MPI tasks use shared memory for communication. However, the inter-LPAR MPI tasks still use the interconnect (either IP or US) for communication.

The results from the parallel (MPI) code runs using IP protocol are shown in Table 4-27.

Table 4-27 The parallel (MPI) IP test runs with shared memory on

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	12.89	13.07	12.79	0.10
inverse_parallel	12.45	12.80	12.39	0.10
series_parallel	12.26	12.40	12.20	0.09

The results from the parallel (MPI) code runs using US protocol are shown in Table 4-28.

Table 4-28 The parallel (MPI) US test runs with shared memory on

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	14.55	14.99	12.79	0.42
inverse_parallel	14.07	14.40	12.40	0.41
series_parallel	13.69	14.00	12.20	0.50

Oddly enough, turning on shared memory does not appear to have helped, and, in the case of US, it may even have hurt. (However, load on the Cluster 1600 may again have been a factor with the US anomaly.) Perhaps affinity logical partitions (ALPARs) would have been a better choice in this situation.

4.3.6 IP vs. US with single LPAR using shared memory

For our final set of test runs, we ran the codes on a single LPAR with shared memory for all of the MPI communications. We changed the definition of one of our LPARs from 8-way to 10-way, but left the memory at 16 GB. (This also required that we change our NFS server LPAR from 4-way to 2-way to free up the needed CPUs, but we left the memory at 4 GB.) We also had to modify the hostfile, as shown in Example 4-36.

Example 4-36 The hostfile for the shared memory runs

```
[c37f1rp08][/ptmp/mgenty]> cat hostfile
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
c37f1rp08.ppd.pok.ibm.com
```

For this series of tests, we ran both the IP and the US wrapper scripts. This was for validation purposes. Given that shared memory was being used for the MPI communications, we expected to see no significant differences between the run times of the two different versions.

The results from the IP wrapper script parallel (MPI) code runs are shown in Table 4-29.

Table 4-29 Shared memory runs with the IP wrapper script

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	12.84	13.39	12.79	0.11
inverse_parallel	12.31	12.40	12.19	0.10
series_parallel	12.01	12.20	11.99	0.04

The results from the US wrapper script parallel (MPI) code runs are shown in Table 4-30 on page 128.

Table 4-30 Shared memory runs with the US wrapper script

Code	Mean	Max	Min	Std. Dev.
inverse_parallel_enabled	12.99	13.59	12.79	0.17
inverse_parallel	12.67	17.60	12.39	0.81
series_parallel	12.35	12.60	12.20	0.13

These results are a bit surprising for a number of reasons:

- The second set of runs all have consistently higher average run times than the first. This very well could just be random variation, and additional runs are needed to prove or disprove this.
- Several of the runs had single, large outputs. For example, the inverse_parallel US run had one with a value of 17.60 seconds. This is roughly five seconds longer than any of the other test runs, and this data point occurred in the middle of the series of runs, rather than at the beginning or end. This is another case where additional runs are needed. We have no good explanation for this anomaly, especially considering that (by chance) we had dedicated use of the Cluster 1600 for these runs.
- Notice that the run times for both sets of runs are fairly close to the run times for the IP runs with the reorganized hostfile (Table 4-25 on page 125) and also the IP runs with shared memory and the reorganized hostfile (Table 4-27 on page 126). This is even more evidence that these codes perform better with IP communications than with US communications.
- We were expecting the shared memory numbers to be significantly better than any of the previous runs, and they were not. The next logical step here would be to run the codes through a profiler (such as xprofiler) to see where the codes are spending the bulk of their time. Is the majority of the time being spent in computation or communication? We suspect the former, and a run through a profiler would enable us to prove it.
- These shared memory tests were run in a normal LPAR. Would their run times improve in an ALPAR? We suspect so, and, at the very least, it certainly seems like a logical place to use ALPARs.

4.3.7 IP vs. US investigation conclusions

Our conclusions from this investigation are as follows:

- The choice between IP or US is not clearly delineated. Some codes do better with IP, some with US, and some do not seem to care either way. Because it is so easy to switch between the two on a code-by-code basis, we recommend that you try both, and pick whichever works best for the given code. If it is a tie, we recommend that you use US, since the IP window is a shared resource.

- We also recommend that, if possible, you use ALPARs to help improve the locality of reference for the memory subsystem. In the best case, it will help. In the worst case, it will do nothing.
- If your codes are at all complex, use tools like xprofiler and PE Benchmarker (Appendix C, “Parallel tools” on page 173) to better understand their performance both on the pSeries 690 and within the Cluster 1600. These tools can help you make better performance and tuning configuration choices based upon your specific application types and mixes.

As much as we would have liked to have had nice, tidy results from these experiments with clear cut answers that option X is better than option Y, the real world rarely, if ever, works that way. As you have seen, one set of experiments typically leads to another, and each brings with it a whole new set of questions. This is yet another place where the 80/20 rule seems to apply. Absolute answers are hard to come by in the area of performance and tuning, but you can usually get eighty percent of what you need with about twenty percent of the effort it would take to reach certainty. Going with what the eighty percent suggests is, at the very least, a good starting point, and, in many cases, is a good ending point as well. However, you always need to keep in mind that performance and tuning is an iterative process. Nothing remains static, and this applies to performance and tuning settings as much as it applies to applications and application mixes.

4.4 CHARMm IP vs. US investigation

Note: This investigation is primarily of interest for folks from the scientific and technical community. Few, if any, business applications use US protocol across the interconnect, so folks from the commercial community may want to skip this section.

For this investigation, we used CHARMm, which is a popular molecular dynamics code used in the Life Sciences discipline. A series of parallel MPI runs (2-way, 4-way, 8-way, and 16-way) were made to see if the relative performance of the IP and US protocols changed as the number of tasks handled by the adapter increased. We were particularly interested in seeing if we could observe the effects discussed in Section 3.7.3, “Internet Protocol (IP) and User Space (US) switch windows” on page 74. We were expecting CHARMm to be more dependent on message passing bandwidth than on latency because it sends large messages with low frequency rather than small messages with high frequency. This behavior should favor IP over US.

The test system consisted of two normal LPARs in a single pSeries 690. Each LPAR was 8-way with 16 GB of memory. Each LPAR was connected to the single-plane SP Switch2 switch with a single SP Switch2 PCI Attachment Adapter. All of the parallel runs in this investigation were set up to communicate across the interconnect, and the MPI tasks were distributed equally among the two LPARs. (The environment is more fully described in Section 4.4.1, “Setting up the environment for IP vs. US testing” on page 130.) The software levels for the two LPARs were the same as for Section 4.3, “IP vs. US investigation” on page 113.

4.4.1 Setting up the environment for IP vs. US testing

Preparing the test environment involved these steps:

- Increasing the switch send pool and receive pool sizes
- Checking the other network options for unusual settings
- Preparing the POE and LoadLeveler environments
- Setting up the executables and hostfiles
- Creating the scripts for the tests

The first three steps were performed as in Section 4.3.1, “Setting up the environment for IP vs. US testing” on page 114.

Tip:

To view US switch adapter window, use:

```
/usr/bin/st_status <hostname>
```

The output shows the status of the switch table as well as the activity for each of the US adapter windows. Example 4-37 on page 130 shows a sample output.

Note that the **st_status** command does not provide information about IP switch adapter window use. This is because there is a single IP switch adapter window that is shared by all IP traffic across the interconnect.

Example 4-37 Output from the st_status command during a 16-way US run

```
$ st_status c37f1rp08
**************************************************************
Node c37f1rp08 adapter /dev/css0 window 0 returned ST_SWITCH_NOT_LOADED.
**************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
```

```
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 1
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 2
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 3
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 4
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 5
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 6
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 7
***************************************************************
Status from node: c37f1rp08   User: tarkus
Load request from: c37f1rp08 Pid: 15594 Uid: 12112
```

```
Job Description: 1812735943
Time of request: Thu_Jun_27_10:57:11_EDT_2002
Adapter: /dev/css0 Memory Requested: 1048576
Window id: 8
**************************************************************
Node c37f1rp08 adapter /dev/css0 window 9 returned ST_SWITCH_NOT_LOADED.
**************************************************************
Node c37f1rp08 adapter /dev/css0 window 10 returned ST_SWITCH_NOT_LOADED.
**************************************************************
                                       |
             Windows 11-64 were also not loaded, output not shown.
                                       |
**************************************************************
Node c37f1rp08 adapter /dev/css0 window 65 returned ST_SWITCH_NOT_LOADED.
```

Setting up the executables and hostfiles

For this investigation, we used CHARMm Version 27b3. Some of the more pertinent points with respect to the build are:

- The CHARMm executables were created on a pSeries 690 running AIX 5L Version 5.1 Recommended Maintainence Package 5100-02 using XL Fortran Version 7.1.1.2 and VAC/VAC++ Version 5.0.2. (The bulk of the code is Fortran, but there is also some C.) The -qarch and -qtune options were specified as auto. Optimization levels varied from routine to routine. The bulk of the code was compiled with -O and -qmaxmem=-1, but certain routines were compiled with -O3.
- The MASS Version 3.0 library was used.
- There were separate 32-bit executables for the parallel and serial versions.

The test system did not have a high-performance, shared file system, such as GPFS. Rather than use NFS, we opted instead to install the CHARMm executables and input files in duplicate directory structures on each LPAR. (Remember, our goal was to measure communications performance not file system performance.) The output files for each job submission were written to the local file system on the LPAR from which the job was originally submitted.

As discussed in Section 3.6.5, “Hostfile considerations for MPI performance” on page 64, the distribution of tasks for an MPI parallel job can be controlled through the hostfile mechanism. For this investigation, we created hostfiles to distribute tasks evenly between the two LPARs.

Two examples of hostfiles for an 8-way parallel job are shown in Example 4-38 on page 133.

Example 4-38 Sample hostfiles for IP vs. US investigation

```
$ cat host.08_f1f2
c37f1rp08
c37f1rp08
c37f1rp08
c37f1rp08
c37f2rp08
c37f2rp08
c37f2rp08
c37f2rp08
$
$ cat host.08_f1f2_sw
c37sn08
c37sn08
c37sn08
c37sn08
c37sn24
c37sn24
c37sn24
c37sn24
$
```

Unless otherwise noted, the hostfiles used for the runs favored *nearest neighbor* communications (see Section 3.6.5, “Hostfile considerations for MPI performance” on page 64). However, for comparison purposes, we also created hostfiles to force sequential tasks to alternate between the two LPARs (that is, c37f1rp08, c37f2rp08, c37f1rp08, c37f2rp08, and so on).

Important:

You may be wondering why we created two different hostfiles for the 8-way job across the two LPARs.

The two hostfiles differ as follows:

- The first, host.08_f1f2, uses the en0 names (such as c37f1rp08 and cc3f2rp08).
- The second, host.08_f1f2_sw, uses the css0 names (such as c37sn08 and c37sn24).

The output of **`netstat -i`** shows the distinction:

```
$ netstat -i
Name  Mtu   Network     Address          Ipkts Ierrs  Opkts      Oerrs  Coll
en0   1500  link#2      0.2.55.6a.ae.5e  36423029     0 35746140     0     0
en0   1500  9.114.189   c37f1rp08        36423029     0 35746140     0     0
css0  65504 link#3                       30961280     0 31038412     0     0
css0  65504 9.114.189.6 c37sn08          30961280     0 31038412     0     0
lo0   16896 link#1                       839718       0 840340       0     0
lo0   16896 27          localhost        839718       0 840340       0     0
lo0   16896 ::1                          839718       0 840340       0     0
$
```

The reason for having two is:

- Runs that use US over the switch are best made with hostfiles that use en0 (host) names. Use of the switch is *predetermined* by the MP_EUILIB=us environment variable setting.
- Runs that use IP over the switch are best made with hostfiles that use css0 names. This ensures that the switch will be used and eliminates the need for setting the MP_EUIDEVICE environment variable.

The approach outlined here is merely a recommendation. How you set things up will be determined partly by how your system is configured and partly by personal preference. We will have more to say about this subject later in this section.

Creating the run scripts

We created three *very* basic CHARMm run scripts to do the following:

- Set the necessary environment variables
- Select the correct hostfiles
- Submit the runs in the proper sequence

The scripts we created are by no means production CHARMm run scripts. They do no checking for path settings, proper arguments, return codes, and so on. They are meant specifically for these investigations and nothing more.

The three CHARMm run scripts are:

run_this	The master wrapper script. It calls the run_charmm script to submit the various CHARMm jobs.
run_charmm	The script to submit a given CHARMm job. It takes three arguments: number of processors, communication protocol to be used, and input file.
run_solo	The single-run CHARMm test script. It takes the same arguments as the run_charmm script.

Example 4-39 is a listing of all three run scripts.

Example 4-39 Run scripts for CHARMm IP vs. US investigation

```
$ cat run_this

#!/bin/ksh
#
# This script serves as the master for job submission.
# Environment variables which must be set for all job submissions
# are done at the beginning of the script. Environment variables which
# must be set for specific jobs (that is, IP vs. US) are set just before the
# invocation of those jobs.
#

#
# Set "general" environment variables
#

export LOADL_INTERACTIVE_CLASS=parallel

#
# Ensure IP uses same mode as US.
#
export MP_WAIT_MODE=poll
#
# Ensures use of shared mem within node/LPAR when running across nodes/LPARs.
#
export MP_SHARED_MEMORY=yes
#
# Generate additional messages. US will give host info.
#
export MP_INFOLEVEL=2
export AIXTHREAD_SCOPE=S
export AIXTHREAD_MUTEX_DEBUG=OFF
export AIXTHREAD_COND_DEBUG=OFF
export AIXTHREAD_RWLOCK_DEBUG=OFF

#
# Make CHARMm runs.
#
cd /user/tarkus/rundir/charmm

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.16_f1f2_sw
/user/tarkus/rundir/charmm/run_charmm 16 ip 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 16 ip 5cb_B.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.16_f1f2
/user/tarkus/rundir/charmm/run_charmm 16 us 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 16 us 5cb_B.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.08_f1f2_sw
/user/tarkus/rundir/charmm/run_charmm 8 ip 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 8 ip 5cb_B.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.08_f1f2
/user/tarkus/rundir/charmm/run_charmm 8 us 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 8 us 5cb_B.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.04_f1f2_sw
/user/tarkus/rundir/charmm/run_charmm 4 ip 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 4 ip 5cb_B.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.04_f1f2
/user/tarkus/rundir/charmm/run_charmm 4 us 5cb_A.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 4 us 5cb_B.inp ; wait

#
# Try different hostfile for fun
#

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.16_alt_sw
/user/tarkus/rundir/charmm/run_charmm 16 ip 5cb_AA.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 16 ip 5cb_BB.inp ; wait

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.16_alt
/user/tarkus/rundir/charmm/run_charmm 16 us 5cb_AA.inp ; wait
/user/tarkus/rundir/charmm/run_charmm 16 us 5cb_BB.inp ; wait

$ cat run_charmm

#!/bin/ksh
```

```
#
# The three arguments are:
#   The number of processors,
#   The euilib choice,
#   The input file.
#

NUM_PROCS=$1
LIB_CHOICE=$2
INPUT=$3

echo "CHARMM run to be submitted. `date` "
echo " "
export MP_PROCS=$NUM_PROCS
echo "The number of processors is MP_PROCS =" $MP_PROCS

export MP_EUILIB=$LIB_CHOICE
echo "The choice of communication protocol is MP_EUILIB =" $MP_EUILIB

export chmstdin=$INPUT

timex /user/tarkus/hold/c27b3/exec/ibmsp/charmm >
${INPUT}_${MP_PROCS}_${MP_EUILIB}.out 2>&1
wait
echo "CHARMm run complete. `date` "
echo " "
echo " "

$ cat run_solo

#!/bin/ksh
#
# The three arguments are:
#   The number of processors,
#   The euilib choice,
#   The input file.
#

#
# Note that the executable chosen here is the parallel version.
#
NUM_PROCS=$1
LIB_CHOICE=$2
INPUT=$3

echo " "
export MP_PROCS=$NUM_PROCS
echo "The number of processors is MP_PROCS =" $MP_PROCS
```

```
export MP_EUILIB=$LIB_CHOICE
echo "The choice of communication protocol is MP_EUILIB =" $MP_EUILIB

#
# Set "general" environment variables
#

export LOADL_INTERACTIVE_CLASS=parallel

#
# Ensure IP uses same mode as US.
#
export MP_WAIT_MODE=poll
#
# Ensures use of shared mem within node/LPAR when running across nodes/LPARs.
#
export MP_SHARED_MEMORY=yes
#
# Generate additional messages. US will give host info.
#
export MP_INFOLEVEL=2

export AIXTHREAD_SCOPE=S
export AIXTHREAD_MUTEX_DEBUG=OFF
export AIXTHREAD_COND_DEBUG=OFF
export AIXTHREAD_RWLOCK_DEBUG=OFF

export MP_HOSTFILE=/user/tarkus/rundir/hostfiles/host.16_f1f2_sw

echo "The input is " $INPUT
echo " "
#
# CHARMm Ver C27B3 requires this.
#
export chmstdin=$INPUT
echo "CHARMM run to be submitted. `date` "
echo " "

timex /user/tarkus/hold/c27b3/exec/ibmsp/charmm >
${INPUT}_${MP_PROCS}_${MP_EUILIB}.out 2>&1
wait

echo "CHARMm run complete. `date` "
echo " "
```

Before proceeding, let us discuss some of the environment variables used in the scripts. For illustration purposes, we will use the run_solo script, because it has no dependencies on the other scripts, and it contains all of the environment variables that we will be discussing here.

The MP_EUILIB environment variable controls which communications protocol will be used: US or IP.

When MP_EUILIB is set to `us`, communications will always occur over the switch adapters present in the LPARs. Settings for the MP_EUDEVICE will be ignored because the US protocol is only supported over the switch.

When MP_EUILIB is set to `ip`, communications can occur over either the switch adapters or Ethernet adapters present in the LPARs. There are a number of ways in which to specify the choices and various interactions can occur with each. The controlling factors are:

- The settings for MP_EUIDEVICE, MP_RESD, and MP_RMPOOL
- Whether or not a hostfile is used
- Personal preference

The interaction of these factors is described in Table 1 on page 14 in the *Parallel Environment for AIX: Operations and Use, Volume 1*, SA22-7425. For the IP runs, we found it most convenient to use a hostfile comprised of the switch names.

All runs were made with MP_SHARED_MEMORY set to `yes` and MP_WAIT_MODE set to `poll`. The latter ensured that the behavior of a blocked task was the same for both US and IP. (The default for US is `poll`, and the default for IP is `sleep`.)

4.4.2 Running the tests for IP vs. US

The following conditions were observed for the test runs:

- All tests were run on the same two LPARs, and no other work was running at the time.
- The switch was used for all but the serial runs.
- The same input was used for all runs, but was copied and renamed for convenience.
- To help check consistency of results, two runs for each case (that is, a given number of processors and a given choice of IP or US) were performed. (In one case, an outlying data point caused us to perform another run as a check.)

The results for all of the runs are shown in Table 4-31 on page 140.

Important: The run-time results shown in all of the tables in this investigation represent the real time component of the `timex` command output for the runs. The real time component is expressed in seconds.

Table 4-31 Comparison of IP vs. US for CHARMm across two 8-way LPARs

Run	Processors	Protocol	Real (sec)
A	1 (serial version)	not applicable	1420.20
B	1 (serial version)	not applicable	1420.20
A	2	ip	1358.46
B	2	ip	1362.02
A	2	us	1359.60
B	2	us	1359.28
A	4	ip	738.16
B	4	ip	736.82
A	4	us	737.27
B	4	us	740.69
A	8	ip	448.88
B	8	ip	608.78
C	8	ip	449.33
A	8	us	438.53
B	8	us	439.36
A	16	ip	347.41
B	16	ip	347.25
A	16	us	320.29
B	16	us	320.97
A	16 (alt)	ip	369.15
B	16 (alt)	ip	372.58
A	16 (alt)	us	349.68
B	16 (alt)	us	351.39

The results from the runs are:

- IP and US performance was roughly the same for the 2-way and 4-way jobs. US provided just slightly better performance for the 8-way runs. The improvement was about two percent, which may be too close to the noise level to be considered a real improvement. For the 16-way runs, the margin between IP and US jobs grew. The US jobs were about ten percent faster.
- Reordering the hostfile from *nearest neighbor* to alternating between LPARs decreased performance by about six percent.

These results were somewhat surprising because, as we mentioned at the beginning of this investigation, we were expecting CHARMm to do better with IP than with US. Our initial assumption was that CHARMm communication is more bandwidth driven than latency driven. This was not the case, and, in fact, US performed better than IP even on the 16-way runs.

The observation that US performs better than IP for the 16-way runs might be explained by a change in the communication pattern as the number of tasks is increased. For the same input, the message sizes for a 16-way job will be smaller than those for a 2-way job or a 4-way job. Also, communication between tasks will be more frequent as more tasks are being run. These trends favor US communication over IP communication, and this was observed in the relative performance of IP vs. US over the series of 2-way, 4-way, 8-way, and 16-way runs.

Note: This is strictly speculation on our part. Yes, it is reasonable. Yes, it makes sense. However, it has not been proven by measurement.

Unfortunately, time did not allow additional runs and measurements to be made. The next step would have been to use tools like PE Benchmarker and MPI Trace (Appendix C, “Parallel tools” on page 173) to actually measure message frequencies and sizes for the different runs. Only then could we begin to really understand which factors were affecting the performance of the application.

4.4.3 CHARMm IP vs. US investigation conclusions

Perhaps it is no surprise that the general conclusions here are the same as those in Section 4.3.7, “IP vs. US investigation conclusions” on page 128. However, it is important to add that the choice of IP or US for a given application could conceivably change as the problem type, size, and number of tasks change. We encourage you to experiment with both and to not assume that because one worked better than the other this time for this application that this will not necessarily always be the case. Also, do not forget that there are tools available to help you further understand the communications patterns within the application.

5

Summary

Good performance and tuning begins with understanding. You would not think to try to tune the performance of your automobile engine without first understanding what a timing belt is or how a spark plug works. The same holds true for the pSeries 690 in a Cluster 1600. You first need to understand things like what an MCM is, what an LPAR is, how the memory subsystem works, and so on. That is why we spent the first three chapters covering concepts. Our fundamental belief is that if you do not know how it works, you cannot do a good job of tuning it. Time spent in learning the internals is repaid many times over when it comes time to solving a performance and tuning problem.

The pSeries 690 in a Cluster 1600 presents some very unique challenges in the area of performance and tuning. If you think about it, what you are really dealing with is a cluster within a cluster within a cluster:

- The Cluster 1600 is made up of multiple nodes, some or all of which can be pSeries 690 LPARs.
- The pSeries 690 LPAR is made up of one or more MCMs, and multiple MCMs can be viewed as a cluster from a performance and tuning standpoint.
- The pSeries 690 MCM is made up of a number of POWER4 chips, caches, and interconnects.

In this environment, it is absolutely imperative that you first identify the performance and tuning problem as either isolated or systemic. If you attack an isolated case as though it is systemic or vice versa, you will create more problems than you solve. For isolated problems, start at the lowest level and work outwards. For systemic problems, start at the highest level and work inwards. Here once again are the recommendations that we provided at the beginning of this book:

- Know your system, and know your workload.
- Document before and after you make a change.
- Change one thing at a time, and have a backout plan.
- Know how to measure what you are changing.
- When possible, measure with more than one tool.
- Distinguish between systemic versus isolated performance issues.
- For isolated (node) problems, look for bottlenecks in the following order: CPU, memory, disk I/O, and network. For systemic (cluster) problems, look for bottlenecks in the opposite order.

We have tried to accomplish three things with this book:

1. We have tried to give you a better understanding of the pSeries 690 architecture as well as that of the new SP Switch2 with the SP Switch2 PCI Attachment Adapter interconnect fabric.
2. We have tried to give you a better understanding of the performance-relevant features of both of these items.
3. We have tried to give you some ideas about how to test and measure these features in your own environment.

Along the way, we have seen some things (for example, ALPARs) that look useful for many environments and others things (for example, memory affinity) that look useful only in very specific cases. We also hope that we have shown how messy and time consuming this kind of work can be, and, to that end, we go back to a comment we made earlier. Begin by trusting the experts. Do not try to reinvent the wheel. Many very smart people have spent many hours looking at this stuff. Build on what they have done. Use it as your baseline, and move forward from there.

Finally, we would like to suggest that in your quest for performance and tuning knowledge, you be open to sources outside of the I/T world. The field of queueing theory is another tangential area that has a wealth of relevant information for performance and tuning of the pSeries 690 in a Cluster 1600. There are also a number of writings in the field of manufacturing that are surprisingly germane to this work as well. For example, the novel *The Goal* by Eliyahu Goldratt provides some very good insights into how to find and work with bottlenecks in a system, be it manufacturing (which is the subject of the book) or Cluster 1600s.

Oh yeah, what about that missing $1.00? The friends initially paid $30.00 for the room. The proprietor kept $25.00 and gave $5.00 to his son. The son kept $2.00 and returned $3.00 to the friends. So, each friend paid $9.00 for the room, and three times $9.00 is $27.00, *minus* the $2.00 the son kept leaves the proprietor with the $25.00 that he kept. Once you see it, you wonder why you did not see it all along. The same holds true when tracking down and fixing performance and tuning problems with the pSeries 690 in a Cluster 1600. Happy hunting.

A

Scripts

Contained within this appendix are the scripts that we developed during the course of our residency or ones that we brought with us and thought might be of use to others.

The pmrinfo tool

The pmrinfo tool is a script that was developed at the National Center for Atmospheric Research (NCAR) to quickly gather the software version information that is needed when opening a problem management record (PMR) with IBM.

Sample output

Example A-1 shows an example of the output from a pmrinfo run. The first column is the software component, the second column is the fileset that was used to derive the release level, and the third column is the release level.

Example: A-1 Output from a pmrinfo run

```
[bf0915en][/]> pmrinfo


****************************************************
NCAR SOFTWARE LEVELS: Thu Jun  6 09:34:47 MDT 2002.
****************************************************


AIX:                  bos.mp              4.3.3.79
PSSP:                 ssp.basic           3.2.0.16
LoadLeveler:          LoadL.full          2.2.0.15
GPFS:                 mmfs.gpfs.rte       1.3.0.12
RVSD:                 vsd.rvsd.rvsdd      3.2.0.8
VSD:                  vsd.vsdd            3.2.0.13
RSCT:                 rsct.basic.rte      1.2.1.9
POE:                  ppe.poe             3.1.0.15
PESSL:                pessl.rte.smp43     2.2.0.0
ESSL:                 essl.rte.smp43      3.2.0.0
FORTRAN:              xlfrte              7.1.1.2
PERL:                 perl.rte            5.5.3.75
C:                    xlC.rte             5.0.2.1


[bf0915en][/]>
```

Script listing

Example A-2 is a listing of the pmrinfo script.

Example: A-2 The pmrinfo tool

```
#!/bin/ksh
#desc reports software levels for use in pmrs.
#
# Purpose: This script reports most if not all of
#          the software levels needed when opening
#          a PMR with IBM
#
# Modification History
# 09/28/01 Marc Genty (NCAR) created

print "\n"
print "***************************************************"
print "Software Levels As Of: $(date)"
print "***************************************************"
print "\n"
#
# AIX
#
_f="bos.mp"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"              "$2}')
print "AIX:                  $_v"
#
# PSSP
#
_f="ssp.basic"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"           "$2}')
print "PSSP:                 $_v"
#
# LoadLeveler
#
_f="LoadL.full"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"          "$2}')
print "LoadLeveler:          $_v"
#
# GPFS
#
_f="mmfs.gpfs.rte"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"       "$2}')
print "GPFS:                 $_v"
#
# RVSD
#
_f="vsd.rvsd.rvsdd"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"      "$2}')
print "RVSD:                 $_v"
#
# VSD
#
_f="vsd.vsdd"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"            "$2}')
print "VSD:                  $_v"
#
# RSCT
```

```
#
_f="rsct.basic.rte"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"      "$2}')
print "RSCT:                 $_v"
#
# POE
#
_f="ppe.poe"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"             "$2}')
print "POE:                  $_v"
#
# PESSL
#
_f="pessl.rte.smp43"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"     "$2}')
print "PESSL:                $_v"
#
# ESSL
#
_f="essl.rte.smp43"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"      "$2}')
print "ESSL:                 $_v"
#
# FORTRAN
#
_f="xlfrte"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"              "$2}')
print "FORTRAN:              $_v"
#
# PERL
#
_f="perl.rte"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"            "$2}')
print "PERL:                 $_v"
#
# C
#
_f="xlC.rte"
_v=$(lslpp -l|grep $_f|grep $_f|head -1|awk '{print $1"             "$2}')
print "C:                    $_v"

print "\n"
```

The mkllqwcoll tool

The mkllqwcoll tool is also a script that was developed at the National Center for Atmospheric Research (NCAR). It is used in conjunction with LoadLeveler. The purpose of the tool is to build a temporary *working collective* (WCOLL) that consists of the nodes on which a parallel job is currently executing. This WCOLL file can then be used to further isolate and troubleshoot performance problems on the nodes on which the job is running.

Sample output

Example A-3 shows an example of how mkllqwcoll can be used.

Example: A-3 An example mkllqwcoll session

```
[bf0915en][/]> llq|more
Id                       Owner      Submitted   ST PRI Class        Running On
------------------------ ---------- ----------- -- --- ------------ -----------
bf0913en.106214.0        dennis      6/6  09:21 R  1   all_spec     bf2212en
bf0915en.113473.0        shields     6/6  06:23 R  50  csl_spec     bf1207en
bf0913en.106215.0        mindeje     6/6  09:21 R  50  csl_pr       bf0108en
bf0911en.106603.0        ecco        6/4  14:07 R  50  com_reg      bf1215en
bf0909en.106495.0        olson       6/6  05:57 R  50  csl_reg      bf1914en
bf0909en.106502.0        olson       6/6  06:35 R  50  csl_reg      bf2015en
bf0915en.113476.0        olson       6/6  06:41 R  50  csl_reg      bf2213en
bf0909en.106506.0        kutz2       6/6  06:59 R  50  csl_reg      bf1512en
bf0911en.107054.0        shields     6/6  07:02 R  50  com_reg      bf2105en
...

[bf0915en][/]> mkllqwcoll bf0915en.113473.0
bf1207en
bf0811en
bf0810en
bf0809en
bf0807en
bf0806en
bf0805en
bf0804en
bf0802en
bf0801en
bf0607en
bf0602en
bf0601en
bf0416en
bf0401en
bf0314en
bf0301en

Working Collective File Is: /tmp/llqwcoll.root
```

```
[bf0915en][/]> export WCOLL=/tmp/llqwcoll.root

[bf0915en][/]> dsh 'vmstat 1 2|tail -n 1'
bf1207en:  1  1 99681 388613  0  0  0  0  0  0 445 15279 1305 66  4 30  0
bf0811en:  1  2 93764 384373  0  0  0  0  0  0 591 17021  806 64  6 25  6
bf0810en:  4  1 94138 384944  0  0  0  0  0  0 422 10013 1306 56  2 42  0
bf0809en:  4  1 94436 370292  0  0  0  0  0  0 420 7560  1318 53  2 46  0
bf0807en:  1  1 94756 388558  0  0  0  0  0  0 420 7728  1095 52  0 47  0
bf0806en:  4  1 94735 371050  0  0  0  0  0  0 452 17559 1097 57  3 37  2
bf0805en:  4  1 94285 384634  0  0  0  0  0  0 428 9034   885 63  2 36  0
bf0804en:  4  1 94267 410088  0  0  0  0  0  0 419 6850   968 51  0 48  0
bf0802en:  2  1 94202 371507  0  0  0  0  0  0 426 9375  1170 54  1 45  0
bf0801en:  1  2 96331 367737  0  0  0  0  0  0 432 12538 1381 52  3 44  0
bf0607en:  3  1 94570 369968  0  0  0  0  0  0 419 9505   807 54  1 45  0
bf0602en:  1  1 95166 401388  0  0  0  0  0  0 427 7360  1180 54  1 45  0
bf0601en:  4  2 119925 361201 0  0  0  0  0  0 450 1574   137 99  0  0  0
bf0416en:  4  1 100554 401178 0  0  0  0  0  0 426  269    72 99  0  0  0
bf0401en:  4  2 93256 364999  0  0  0  0  0  0 437  383    91 99  0  0  0
bf0314en:  4  1 121628 354610 0  0  0  0  0  0 426 4287    76 99  0  0  0
bf0301en:  1  2 113328 354935 0  0  0  0  0  0 422 3635   246 29  1 70  0
[bf0915en][/]>
```

Script listing

Example A-4 is a listing of the mkllqwcoll script.

Example: A-4 The mkllqwcoll tool

```
#!/usr/local/bin/perl
#desc creates a working collective from an llq -l listing
#
# Purpose: This script takes a job name as input and
#          produces a working collective file of the
#          nodes on which the job is running.
#
# Modification History
# 12/20/01 Marc Genty (NCAR) created
#
$username        = `/usr/bin/whoami`; chomp $username;
$nodename        = `/usr/bin/hostname`; chomp $nodename;
($system_prefix) = ($nodename =~ /(^..).*/);
$wcollfile       = "/tmp/llqwcoll.${username}";
$llq             = "/usr/lpp/LoadL/full/bin/llq -l ";
#
# Check For Command Line Options.
#
$args_supplied  = 0;
$jobid_supplied = 0;
```

```
while (@ARGV) {
  $arg = shift @ARGV;
  #
  # Get First Letter Of Command Line Option.
  #
  ($arg_check) = ($arg =~ /^(..).*/);
  #
  # Check If Verbose Output Is Being Requested.
  #
  if ($args_supplied lt 1) {
    $args_supplied++;
    if ($arg_check eq $system_prefix) {
      $jobid_supplied = 1;
      $jobid = $arg;
    } else {
      print "\nUsage: mkllqwcoll \<jobid\>\n\n";
      exit(1);
    }
  } else {
    print "\nUsage: mkllqwcoll \<jobid\>\n\n";
    exit(1);
  }
}
if (!$jobid_supplied) {
  print "\nUsage: mkllqwcoll \<jobid\>\n\n";
  exit(1);
}
#
# Get The llq -l Output And Pull Out The Node Names.
#
open(PIPE, "$llq $jobid |") || die "Llq Pipe Open Failed: $!.\n";
@llqlong = <PIPE>; close(PIPE);

@output = ();

foreach $llq_line (@llqlong)
{
  if ($llq_line =~ m/.*bf.*en::.*/) {
    $llq_line =~ s/.*(bf.*en)::.*/$1/;
    push @output, sprintf("$llq_line");
    print "$llq_line";
  }
}
#
# Create The WCOLL File.
#
$outref = \@output;
@tosub = ($wcollfile,$outref);
make_wcoll(@tosub);
```

```
#
# That's All.
#
exit(0);

# ############################################################################
# subroutine definitions.
# ############################################################################

# ----------------------------------------------------------------------------
#  void make_wcoll(@tosub)
# ----------------------------------------------------------------------------
sub make_wcoll {
  return unless @_;
  my ($wcollfile,$outerf) = @_;
  my (@output) = @$outref;
  my $out_line;
  unless (-e $wcollfile) {
    system "touch $wcollfile";
  }
  open(WCOLL ,">$wcollfile") || die "Could Not Open ${wcollfile}: $!.\n";
  foreach $out_line (@output) {
    print WCOLL $out_line;
  }
  close(WCOLL);
  print "\nWorking Collective File Is: ${wcollfile}\n\n";
}
```

Note: You will need to change the llq_line parsing to map to your node naming convention.

The fix_nlspath tool

If you go to run parallel code through POE or LoadLeveler and receive a bunch of error message about the NLSPATH not being correctly set, try running this script as root. It may fix the problem. If it does, you will need to rerun it after each system reboot.

Script listing

Example A-5 is a listing of the fix_nlspath script.

Example: A-5 The fix_nlspath tool

```
#!/usr/bin/ksh
#
# Descrption: Workaround Script To Fix The NLSPATH.
```

```
#
# Location:  /ssg/bin/fix_nlspath
#
# Modification History
# --------------------
# 02/20/02  mgenty  Created.
#
chnlspath
/usr/lib/nls/msg/en_US/%N:/usr/lib/nls/msg/en_US/%N.cat:/usr/dt/lib/nls/msg/en_
US/%N:/usr/dt/lib/nls/msg/en_US/%N.cat
#
# NOTE: The "chnlspath /usr/lib/nls/..." above is all one continuous line!
#
```

Important: You will need to rerun this script each time the system or LPAR is rebooted. You may want to add it to one of your system startup files, like /etc/rc.local.

The mk_temp_dir

This script (Example A-6) creates a temporary directory mounted on a ramdisk. For input, the script needs the size of the ramdisk and the mount directory. It then creates a ramdisk, formats it, creates a file system mount point, and mounts it.

Example: A-6 mk_temp_dir listing

```
#!/bin/ksh
echo "Please input the size of ramdisk"
echo "Default size is 4194304"
read NEW_SIZE
T="txt"
if [ "$NEW_SIZE$T" = "$T" ]
then
RAMDISK_SIZE=4194304
else
RAMDISK_SIZE=$NEW_SIZE
fi
echo "Please input the mount directory"
echo "Default is /tmp/mnt"
read NEW_MOUNTDIR
if [ "$NEW_MOUNTDIR$T" = "$T" ]
then
MOUNT_DIR=/tmp/mnt
else
MOUNT_DIR=$NEW_MOUNTDIR
fi
echo "Creating ramdisk ..."
```

```
NAME_RAMDISK=`mkramdisk $RAMDISK_SIZE`
if [ $? -ne 0 ]
then
echo "RAM_DISK_ADD failed: Not enough space"
exit 1
fi
NUMBER_RAMDISK=`echo $NAME_RAMDISK|cut -c14-`
echo "Creating file system ..."
mkfs -V jfs /dev/ramdisk$NUMBER_RAMDISK
if [ $? -ne 0 ]
then
echo "Some Error occurred ..."
exit 2
fi
echo "Creating directory $MOUNT_DIR ..."
mkdir -p $MOUNT_DIR
if [ $? -ne 0 ]
then
echo "Some Error occurred ..."
exit 3
fi
echo "Mounting  /dev/ramdisk$NUMBER_RAMDISK in $MOUNT_DIR"
mount -V jfs -o nointegrity /dev/ramdisk$NUMBER_RAMDISK $MOUNT_DIR
if [ $? -ne 0 ]
then
echo "Some Error occurred ..."
exit 4
fi
```

The rm_temp_dir

This script (Example A-7) removes the temporary directory mounted on a ramdisk and destroys the ramdisk. The script first shows the mounted file systems and the ramdisks, then requests the number of the ramdisk to be deleted.

Example: A-7 rm_temp_dir listing

```
#!/bin/ksh
K="txt"
NR=`df -k|grep -c ramdisk`
if [ $NR -eq 0 ]
then
echo "There is no ramdisk file system mounted"
exit 1
fi
df -k |grep ramdisk
```

```
echo "Which file system mounted on a ramdisk do you want to remove? "
echo "ex:0, 1, 2,..."
read NAME_RAMDISK
if [ $NAME_RAMDISK$K = $K ]
then
echo "ok .... BYE ..."
exit 0
fi
MOUNT_POINT=`df -k |grep -w ^/dev/ramdisk$NAME_RAMDISK | awk '{ print $7 }'`
echo "Unmounting $MOUNT_POINT"
umount -f $MOUNT_POINT
if [ $? -ne 0 ]
then
echo "Some Error occurred ..."
exit 2
fi
echo "Removing /dev/ramdisk$NAME_RAMDISK ..."
rmramdisk /dev/ramdisk$NAME_RAMDISK
if [ $? -ne 0 ]
then
echo "Some Error occurred ..."
exit 2
fi
```

B

MPI sample programs

The following programs are taken from Chapter 2 of the *Parallel Environment for AIX: Hitchhiker's Guide*, SA22-7424. They were used for the IP vs. US investigation in Chapter 4, “Investigations” on page 77. These programs and associated documentation are available for download at:

`http://www.ibm.com/servers/eserver/pseries/library/sp_books/pe.html`

There were three programs used for the investigation. They are:

- The inverse_parallel_enabled.c MPI program
- The inverse_parallel.c MPI program
- The series_parallel.c MPI program

The MPI codes were compiled with separate make files, which are also shown here.

The inverse_parallel_enabled.c MPI program

The inverse_parallel_enabled.c source code is listed in Example B-1.

Example: B-1 The inverse_parallel_enabled.c source code

```
/************************************************************************
 *
 * Hitchhiker's Guide to the IBM PE
 * Matrix Inversion Program - serial version enabled for parallel environment
 * Chapter 2 - The Answer is 42
 *
 * To compile:
 * mpcc -g -o inverse_parallel_enabled inverse_parallel_enabled.c
 *
 ************************************************************************/

#include<stdlib.h>
#include<stdio.h>
#include<assert.h>
#include<errno.h>
#include<mpi.h>

float determinant
      (float **matrix,int size, int * used_rows, int * used_cols, int depth);
float coefficient(float **matrix,int size, int row, int col);
void print_matrix(FILE * fptr,float ** mat,int rows, int cols);
float test_data[8][8] =  {
                           {4.0, 2.0, 4.0, 5.0, 4.0, -2.0, 4.0, 5.0},
                           {4.0, 2.0, 4.0, 5.0, 3.0, 9.0, 12.0, 1.0 },
                           {3.0, 9.0, -13.0, 15.0, 3.0, 9.0, 12.0, 15.0},
                           {3.0, 9.0, 12.0, 15.0, 4.0, 2.0, 7.0, 5.0 },
                           {2.0, 4.0, -11.0, 10.0, 2.0, 4.0, 11.0, 10.0 },
                           {2.0, 4.0, 11.0, 10.0, 3.0, -5.0, 12.0, 15.0 },
                           {1.0, -2.0, 4.0, 10.0, 3.0, 9.0, -12.0, 15.0 } ,
                           {1.0, 2.0, 4.0, 10.0, 2.0, -4.0, -11.0, 10.0 } ,
};
#define ROWS 8

int me, tasks, tag=0;

int main(int argc, char **argv)
{

  float **matrix;
  float **inverse;
  int rows,i,j;
  float determ;
  int * used_rows, * used_cols;
```

```
MPI_Status status[ROWS];              /* Status of messages */
MPI_Request req[ROWS];                /* Message IDs */

MPI_Init(&argc,&argv);                /* Initialize MPI */
MPI_Comm_size(MPI_COMM_WORLD,&tasks); /* How many parallel tasks are there?*/
MPI_Comm_rank(MPI_COMM_WORLD,&me);    /* Who am I? */

rows = ROWS;

/* Allocate markers to record rows and columns to be skipped */
/* during determinant calculation                             */
used_rows = (int *)    malloc(rows*sizeof(*used_rows));
used_cols = (int *)    malloc(rows*sizeof(*used_cols));

/* Allocate working copy of matrix and initialize it from static copy */
matrix  =   (float **) malloc(rows*sizeof(*matrix));
inverse =   (float **) malloc(rows*sizeof(*inverse));
for(i=0;i<rows;i++)
  {
    matrix[i]  = (float *) malloc(rows*sizeof(**matrix));
    inverse[i] = (float *) malloc(rows*sizeof(**inverse));
    for(j=0;j<rows;j++)
      matrix[i][j] = test_data[i][j];
  }

/* Compute and print determinant */
printf("The determinant of\n\n");
print_matrix(stdout,matrix,rows,rows);
determ=determinant(matrix,rows,used_rows,used_cols,0);
printf("\nis %f\n",determ);
fflush(stdout);

for(i=0;i<rows;i++)
  {
    for(j=0;j<rows;j++)
      {
        inverse[j][i] = coefficient(matrix,rows,i,j)/determ;
      }
  }

printf("The inverse is\n\n");
print_matrix(stdout,inverse,rows,rows);

/* Wait for all parallel tasks to get here, then quit */
MPI_Barrier(MPI_COMM_WORLD);
MPI_Finalize();

return 0;
```

```
}

float determinant
      (float **matrix,int size, int * used_rows, int * used_cols, int depth)
  {
    int col1, col2, row1, row2;
    int j,k;
    float total=0;
    int sign = 1;

    /* Find the first unused row */
    for(row1=0;row1<size;row1++)
      {
        for(k=0;k<depth;k++)
          {
            if(row1==used_rows[k]) break;
          }
        if(k>=depth)  /* this row is not used */
          break;
      }
    assert(row1<size);

    if(depth==(size-2))
      {
        /* There are only 2 unused rows/columns left */

        /* Find the second unused row */
        for(row2=row1+1;row2<size;row2++)
          {
            for(k=0;k<depth;k++)
              {
                if(row2==used_rows[k]) break;
              }
            if(k>=depth)  /* this row is not used */
              break;
          }
        assert(row2<size);

        /* Find the first unused column */
        for(col1=0;col1<size;col1++)
          {
            for(k=0;k<depth;k++)
              {
                if(col1==used_cols[k]) break;
              }
            if(k>=depth)  /* this column is not used */
              break;
          }
        assert(col1<size);
        /* Find the second unused column */
        for(col2=col1+1;col2<size;col2++)
          {
            for(k=0;k<depth;k++)
              {
                if(col2==used_cols[k]) break;
              }
            if(k>=depth)  /* this column is not used */
              break;
          }
        assert(col2<size);

        /* Determinant = m11*m22-m12*m21 */
        return matrix[row1][col1] * matrix[row2][col2] - matrix[row2][col1] *
               matrix[row1][col2];
      }

    /* There are more than 2 rows/columns in the matrix being processed  */
    /* Compute the determinant as the sum of the product of each element */
    /* in the first row and the determinant of the matrix with its row   */
    /* and column removed                                                */
    total = 0;

    used_rows[depth] = row1;
    for(col1=0;col1<size;col1++)
      {
        for(k=0;k<depth;k++)
          {
            if(col1==used_cols[k]) break;
          }
        if(k<depth)  /* This column is used */
          continue;
        used_cols[depth] = col1;
        total += sign * matrix[row1][col1] *
                 determinant(matrix,size,used_rows,used_cols,depth+1);
        sign=(sign==1)?-1:1;
      }
    return total;
  }

void print_matrix(FILE * fptr,float ** mat,int rows, int cols)
{
  int i,j;
  for(i=0;i<rows;i++)
    {
      for(j=0;j<cols;j++)
        {
          fprintf(fptr,"%10.4f ",mat[i][j]);
        }
      fprintf(fptr,"\n");
    }
  fflush(fptr);
}

float coefficient(float **matrix,int size, int row, int col)
{
  float coef;
  int * ur, *uc;

  ur = malloc(size*sizeof(matrix));
  uc = malloc(size*sizeof(matrix));
  ur[0]=row;
  uc[0]=col;
  coef = (((row+col)%2)?-1:1)*determinant(matrix,size,ur,uc,1);
  return coef;
}
```

The make file for the inverse_parallel_enabled program is shown in Example B-2.

Example: B-2 The inverse_parallel_enabled make file

```
CC = mpcc
CFLAGS = -g
LIBDIR = -L/usr/lpp/ppe.poe/lib
LIBS = -lmpi
INCLUDE = -I/usr/lpp/ppe.poe/include

inverse_parallel_enabled: inverse_parallel_enabled.c
    $(CC) -o inverse_parallel_enabled $(INCLUDE) $(CFLAGS) $(LIBDIR) $(LIBS)
inverse_parallel_enabled.c
```

The inverse_parallel.c MPI program

The inverse_parallel.c source code is listed in Example B-3.

Example: B-3 The inverse_parallel.c source code

```
/************************************************************************
 *
 * Hitchhiker's Guide to the IBM PE
 * Matrix Inversion Program - First parallel implementation
 * Chapter 2 - The Answer is 42
 *
 * To compile:
 * mpcc -g -o inverse_parallel inverse_parallel.c
 *
 ************************************************************************/

#include<stdlib.h>
#include<stdio.h>
#include<assert.h>
#include<errno.h>
#include<mpi.h>

float determinant
      (float **matrix,int size, int * used_rows, int * used_cols, int depth);
float coefficient(float **matrix,int size, int row, int col);
void print_matrix(FILE * fptr,float ** mat,int rows, int cols);

float test_data[8][8] =  {
                           {4.0, 2.0, 4.0, 5.0, 4.0, -2.0, 4.0, 5.0},
                           {4.0, 2.0, 4.0, 5.0, 3.0, 9.0, 12.0, 1.0 },
                           {3.0, 9.0, -13.0, 15.0, 3.0, 9.0, 12.0, 15.0},
                           {3.0, 9.0, 12.0, 15.0, 4.0, 2.0, 7.0, 5.0 },
                           {2.0, 4.0, -11.0, 10.0, 2.0, 4.0, 11.0, 10.0 },
                           {2.0, 4.0, 11.0, 10.0, 3.0, -5.0, 12.0, 15.0 },
                           {1.0, -2.0, 4.0, 10.0, 3.0, 9.0, -12.0, 15.0 } ,
                           {1.0, 2.0, 4.0, 10.0, 2.0, -4.0, -11.0, 10.0 } ,
};
#define ROWS 8
int me, tasks, tag=0;

int main(int argc, char **argv)
{
  float **matrix;
  float **inverse;
  int rows,i,j;
  float determ;
  int * used_rows, * used_cols;

  MPI_Status status[ROWS];              /* Status of messages */
  MPI_Request req[ROWS];                /* Message IDs */

  MPI_Init(&argc,&argv);                /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD,&tasks); /* How many parallel tasks are there?*/
  MPI_Comm_rank(MPI_COMM_WORLD,&me);    /* Who am I? */

  rows = ROWS;

  /* We need exactly one task for each row of the matrix plus one task */
  /* to act as coordinator.  If we don't have this, the last task      */
  /* reports the error (so everybody doesn't put out the same message  */
  if(tasks!=rows+1)
    {
      if(me==tasks-1)
        fprintf(stderr,"%d tasks required (matrix rows plus one\n",rows+1);
      exit(-1);
    }

  /* Allocate markers to record rows and columns to be skipped */
  /* during determinant calculation                            */
  used_rows = (int *)    malloc(rows*sizeof(*used_rows));
  used_cols = (int *)    malloc(rows*sizeof(*used_cols));

  /* Allocate working copy of matrix and initialize it from static copy */
  matrix = (float **) malloc(rows*sizeof(*matrix));
  for(i=0;i<rows;i++)
    {
      matrix[i] = (float *) malloc(rows*sizeof(**matrix));
      for(j=0;j<rows;j++)
        matrix[i][j] = test_data[i][j];
    }

  /* Everyone computes the determinant (to avoid message transmission */
  determ=determinant(matrix,rows,used_rows,used_cols,0);

  if(me==tasks-1)
    { /* The last task acts as coordinator */
      inverse = (float **) malloc(rows*sizeof(*inverse));
      for(i=0;i<rows;i++)
        {
          inverse[i] = (float *) malloc(rows*sizeof(**inverse));
        }
      /* Print the determinant */
      printf("The determinant of\n\n");
      print_matrix(stdout,matrix,rows,rows);
      printf("\nis %f\n",determ);
      /* Collect the rows of the inverse matrix from the other tasks */
      /* First, post a receive from each task into the appropriate row */
      for(i=0;i<rows;i++)
        {
          MPI_Irecv(inverse[i],rows,MPI_REAL,i,tag,MPI_COMM_WORLD,&(req[i]));
        }
      /* Then wait for all the receives to complete */
      MPI_Waitall(rows,req,status);
      printf("The inverse is\n\n");
      print_matrix(stdout,inverse,rows,rows);
    }
  else
    { /* All the other tasks compute a row of the inverse matrix */
      int dest = tasks-1;
      float *one_row;
      int size = rows*sizeof(*one_row);
```

```
      one_row = (float *) malloc(size);
      for(j=0;j<rows;j++)
        {
          one_row[j] = coefficient(matrix,rows,j,me)/determ;
        }
      /* Send the row back to the coordinator */
      MPI_Send(one_row,rows,MPI_REAL,dest,tag,MPI_COMM_WORLD);
    }

  /* Wait for all parallel tasks to get here, then quit */
  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Finalize();

}

float determinant
      (float **matrix,int size, int * used_rows, int * used_cols, int depth)
  {
    int col1, col2, row1, row2;
    int j,k;
    float total=0;
    int sign = 1;

    /* Find the first unused row */
    for(row1=0;row1<size;row1++)
      {
        for(k=0;k<depth;k++)
          {
            if(row1==used_rows[k]) break;
          }
        if(k>=depth)  /* this row is not used */
          break;
      }
    assert(row1<size);

    if(depth==(size-2))
      {
        /* There are only 2 unused rows/columns left */

        /* Find the second unused row */
        for(row2=row1+1;row2<size;row2++)
          {
            for(k=0;k<depth;k++)
              {
                if(row2==used_rows[k]) break;
              }
            if(k>=depth)  /* this row is not used */
              break;
          }
        assert(row2<size);

        /* Find the first unused column */
        for(col1=0;col1<size;col1++)
          {
            for(k=0;k<depth;k++)
              {
                if(col1==used_cols[k]) break;
              }
            if(k>=depth)  /* this column is not used */
              break;
          }
        assert(col1<size);

        /* Find the second unused column */
        for(col2=col1+1;col2<size;col2++)
          {
            for(k=0;k<depth;k++)
              {
                if(col2==used_cols[k]) break;
              }
            if(k>=depth)  /* this column is not used */
              break;
          }
        assert(col2<size);

        /* Determinant = m11*m22-m12*m21 */
        return matrix[row1][col1] * matrix[row2][col2] - matrix[row1][col2] *
               matrix[row2][col1];
      }

    /* There are more than 2 rows/columns in the matrix being processed  */
    /* Compute the determinant as the sum of the product of each element */
    /* in the first row and the determinant of the matrix with its row   */
    /* and column removed                                                */
    total = 0;

    used_rows[depth] = row1;
    for(col1=0;col1<size;col1++)
      {
        for(k=0;k<depth;k++)
          {
            if(col1==used_cols[k]) break;
          }
        if(k<depth)  /* This column is used -- skip it*/
          continue;
        used_cols[depth] = col1;
        total += sign * matrix[row1][col1] *
                 determinant(matrix,size,used_rows,used_cols,depth+1);
        sign=(sign==1)?-1:1;
      }
    return total;

  }

void print_matrix(FILE * fptr,float ** mat,int rows, int cols)
{
  int i,j;
  for(i=0;i<rows;i++)
    {
      for(j=0;j<cols;j++)
        {
          fprintf(fptr,"%10.4f ",mat[i][j]);
        }
      fprintf(fptr,"\n");
    }
}

float coefficient(float **matrix,int size, int row, int col)
{
  float coef;
  int * ur, *uc;

  ur = malloc(size*sizeof(matrix));
  uc = malloc(size*sizeof(matrix));
  ur[0]=row;
  uc[0]=col;
  coef = (((row+col)%2)?-1:1)*determinant(matrix,size,ur,uc,1);
  return coef;
}
```

The make file for the inverse_parallel program is shown in Example B-4.

Example: B-4   The inverse_parallel make file

```
CC = mpcc
CFLAGS = -g
LIBDIR = -L/usr/lpp/ppe.poe/lib
LIBS = -lmpi
INCLUDE = -I/usr/lpp/ppe.poe/include

inverse_parallel: inverse_parallel.c
    $(CC) -o inverse_parallel $(INCLUDE) $(CFLAGS) $(LIBDIR) $(LIBS)
inverse_parallel.c
```

The series_parallel.c MPI program

The series_parallel.c source code is listed in Example B-5.

Example: B-5 The series_parallel.c source code

```
/************************************************************************
 *
 * Hitchhiker's Guide to the IBM PE
 * Series Evaluation - parallel version
 * Chapter 2 - The Answer is 42
 *
 * To compile:
 * mpcc -g -o series_parallel series_parallel.c -lm
 *
 ************************************************************************/

#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<mpi.h>

double angle[] = { 0.0, 0.1*M_PI, 0.2*M_PI, 0.3*M_PI, 0.4*M_PI,
              0.5*M_PI, 0.6*M_PI, 0.7*M_PI, 0.8*M_PI, 0.9*M_PI, M_PI };

int main(int argc, char **argv)
{
  double data, divisor, partial, sine;
  int a, t, angles = sizeof(angle)/sizeof(angle[0]);
  int me, tasks, term;

  MPI_Init(&argc,&argv);                /* Initialize MPI */
  MPI_Comm_size(MPI_COMM_WORLD,&tasks); /* How many parallel tasks are there?*/
  MPI_Comm_rank(MPI_COMM_WORLD,&me);    /* Who am I? */

  term = 2*me+1;                        /* Each task computes a term */
  /* Scan the factorial terms through the group members   */
  /* Each member will effectively multiply the product of */
  /* the result of all previous members by its factorial  */
  /* term, resulting in the factorial up to that point    */
  if(me==0)
    data = 1.0;
  else
    data = -(term-1)*term;
  MPI_Scan(&data,&divisor,1,MPI_DOUBLE,MPI_PROD,MPI_COMM_WORLD);

  /* Compute sine of each angle */
  for(a=0;a<angles;a++)
    {
      partial = pow(angle[a],term)/divisor;
      /* Pass all the partials back to task 0 and   */
      /* accumulate them with the MPI_SUM operation */
  /* the result of all previous members by its factorial  */
  /* term, resulting in the factorial up to that point    */
  if(me==0)
    data = 1.0;
  else
    data = -(term-1)*term;
  MPI_Scan(&data,&divisor,1,MPI_DOUBLE,MPI_PROD,MPI_COMM_WORLD);

  /* Compute sine of each angle */
  for(a=0;a<angles;a++)
    {
      partial = pow(angle[a],term)/divisor;
      /* Pass all the partials back to task 0 and   */
      /* accumulate them with the MPI_SUM operation */
      MPI_Reduce(&partial,&sine,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
      /* The first task has the total value */
      if(me==0)
        {
          printf("sin(%lf) + %lf\n",angle[a],sine);
        }
    }
  MPI_Finalize();
}
```

The make file for the series_parallel program is shown in Example B-6.

Example: B-6   The series_parallel make file

```
CC = mpcc
CFLAGS = -g -lm
LIBDIR = -L/usr/lpp/ppe.poe/lib
LIBS = -lmpi
INCLUDE = -I/usr/lpp/ppe.poe/include

series_parallel: series_parallel.c
    $(CC) -o series_parallel $(INCLUDE) $(CFLAGS) $(LIBDIR) $(LIBS)
series_parallel.c
```

C

Parallel tools

This appendix provides introductory information on two tools that are useful for understanding performance and tuning issues associated with parallel codes. These codes are typically found in the scientific and technical arena. The tools covered here are:

- PE Benchmarker
- MPI Trace

PE Benchmarker

The PE Benchmarker is a suite of applications and utilities that you can use to analyze the performance of programs run within the IBM Parallel Environment for AIX. The PE Benchmarker suite consists of:

- The Performance Collection Tool (PCT)
- A set of Unified Trace Environment (UTE) utilities
- The Profile Visualization Tool (PVT)

The PE Benchmarker suite is described in more detail in the next paragraphs.

- The Performance Collection Tool (PCT)

  This tool enables you to collect either MPI and user event data or hardware and operating system profiles for one or more application processes (or tasks). This tool is built on dynamic instrumentation technology, the Dynamic Probe Class Library (DPCL). Unlike more traditional tools for collecting message-passing and other performance information, the PCT, because it is built on DPCL, enables you to insert and remove instrumentation probes into the target application while the target application is running. More traditional tools require the application to be instrumented through compilation or linking. This often results in more instrumentation being inserted into the application than is actually needed, and so such tools are more likely to create situations in which the instrumented version of the application is no longer representative of the actual, uninstrumented, version of the application. Since the PCT enables you to make the decision of what data is collected at run time, this typically results in a more acceptable intrusion cost of the instrumentation. What’s more, the files output by the PCT are output on each machine running instrumented processes rather than on a single, centralized, machine. This means that your analysis can be efficiently scaled to collect information on a large number of processes running on a large number of nodes.

- A set of Unified Trace Environment (UTE) utilities

  When you collect MPI and user event traces using the PCT, the collected information is saved, on each machine running instrumented processes, as a standard AIX event trace file. The UTE utilities enable you to convert one or more of these AIX trace files into UTE interval files. While an AIX event trace file has a time stamp indicating the point in time when an event occurred, UTE interval files take this information to also determine how long an event lasts before encountering the next event. Because they include this duration information, UTE interval files are easier to visualize than traditional AIX event trace files. The UTE utilities are:

  - The uteconvert utility, which converts AIX event trace records into UTE interval trace files
  - The utemerge utility, which merges multiple UTE interval files into a single UTE interval file
  - The utestats utility, which generates statistics tables from UTE interval files
  - The slogmerge utility, which converts and merges UTE interval files into a single SLOG file for analysis within Argonne National Laboratory's Jumpshot tool
- The Profile Visualization Tool (PVT)

  When you collect hardware and operating system profiles using the PCT, the collected profile information is saved on each machine running instrumented processes as netCDF (network Common Data Form) files. The PVT can read netCDF files and summarize the profile information in reports.

There are various tools in the PE Benchmarker toolset that work together to enable you to analyze the performance of programs run within the IBM AIX Parallel Environment. Please note that Jumpshot is not part of the PE Benchmarker toolset, but is instead a public domain tool developed at Argonne National Laboratory. It is mentioned here because PE Benchmarker provides the slogmerge utility for converting UTE files into the SLOG format required by Jumpshot.

There is a procedure for collecting and analyzing data using the PE Benchmarker toolset. This procedure starts with the PCT. When using the PCT, you must select the type of data you are collecting — either MPI and user event trace data or hardware and operating system performance data. You use the PCT to connect to existing processes, or start processes running (which also connects to the processes). By *connect to processes*, we mean that the PCT establishes a communication connection that enables it to control the process' execution (suspend, resume, and terminate the process), and also instrument the process with data collection probes. Data files containing the collected information will be generated on each machine running at least one instrumented process. The format of the files generated depends on the type of data you are collecting:

- If you are collecting MPI and user event trace data, standard AIX trace files will be generated. You will first need to take the AIX trace files output by the PCT and convert them, using the uteconvert utility, into UTE interval files. If you want to view statistical tables of the information contained in the UTE interval files, you can use the utestats utility. You can optionally merge multiple UTE files into a single UTE file using the utemerge utility before using the utestats utility to generate the statistical tables. If you instead want to view the information contained in the UTE interval files graphically, you can convert them into SLOG files which are readable by Argonne National Laboratory's

Jumpshot tool. To convert UTE interval files into SLOG files, you use the slogmerge utility. The slogmerge utility can convert a single UTE interval file into a single SLOG file, or it can convert multiple UTE interval files into a single, merged, SLOG file.

- If you are collecting hardware performance data, netCDF files will be generated. You can use the PVT to generate graphs and reports of the information contained in the netCDF files.

For more information about the PE Benchmarker Toolset, see *Parallel Environment for AIX: Operations and Use, Volume 1*, SA22-7425, and *Parallel Environment for AIX: Operation and Use, Volume 2*, SA22-7426. Both can be downloaded from:

http://www.ibm.com/servers/eserver/pseries/library/sp_books/pe.html

MPI Trace

The MPI Trace tool enables you to see how much time your codes are spending doing message passing calls. If you recall from the IP vs. US Investigation in Chapter 4, "Investigations" on page 77, the best we could do with the **timex** command was to show the real time component. The MPI Trace tool provides much finer granularity and is very useful for finding load imbalances within the code.

Example C-1 shows a sample of output from the tool. The output consists of three sections:

- The top section shows the accumulated message passing time in each MPI routine, then a summary of total elapsed time, total communication time, and so on.
- The middle section shows the message-size distribution.
- The bottom section shows the hardware counter output (obtained from the pmapi interface).

Example: C-1 Sample MPI Trace output

```
----------------------------------------------------------------
MPI Routine                  #calls     avg. bytes      time(sec)
----------------------------------------------------------------
MPI_Comm_size                     2            0.0          0.000
MPI_Comm_rank                     2            0.0          0.000
MPI_Isend                    117688        78153.9          4.546
MPI_Recv                          1      1901592.0          0.901
MPI_Irecv                    117688        78153.9          3.132
MPI_Waitall                   58844            0.0        459.150
MPI_Bcast                       107        29903.4          7.827
```

```
MPI_Barrier                   4801             0.0          35.685
MPI_Gather                       2             4.0           0.000
MPI_Gatherv                     28       2328480.0          71.437
MPI_Allgatherv                4800       2449440.0         303.591
MPI_Allreduce                  311            12.0           9.512
MPI_Alltoallv                15600        244332.8         592.384
----------------------------------------------------------------
total communication time = 1488.165 seconds.
total elapsed time       = 9972.984 seconds.
user cpu time            = 9953.430 seconds.
system time              = 11.430 seconds.
maximum memory size      = 650752 KBytes.

----------------------------------------------------------------
Message size distributions:

MPI_Isend                #calls     avg. bytes      time(sec)
                          16824            0.0          0.530
                           6012            4.0          0.212
                           1804         1368.0          0.014
                           1804         2152.0          0.061
                           9600        63368.0          0.376
                          81644       105128.0          3.354

MPI_Recv                 #calls     avg. bytes      time(sec)
                              1      1901592.0          0.901

MPI_Irecv                #calls     avg. bytes      time(sec)
                          16824            0.0          0.076
                           6012            4.0          0.032
                           1804         1368.0          0.006
                           1804         2152.0          0.020
                           9600        63368.0          0.215
                          81644       105128.0          2.783

MPI_Bcast                #calls     avg. bytes      time(sec)
                              6            4.0          7.624
                             17           15.5          0.000
                             36           24.1          0.102
                             12           44.0          0.000
                              8           85.0          0.000
                              5          221.6          0.000
                              4          292.0          0.000
                              1          524.0          0.000
                              3         1268.0          0.000
                              2         3036.0          0.000
                              3         4881.3          0.000
                              4         9548.0          0.001
                              2        26208.0          0.002
                              2       40872.0          0.010
                              1      128144.0          0.003
                              1     2869488.0          0.082

MPI_Gather                #calls     avg. bytes      time(sec)
                              2           4.0          0.000

MPI_Gatherv               #calls     avg. bytes      time(sec)
                             28     2328480.0         71.437

MPI_Allgatherv            #calls     avg. bytes      time(sec)
                           4800     2449440.0        303.591

MPI_Allreduce             #calls     avg. bytes      time(sec)
                            310           4.0          9.510
                              1        2496.0          0.001

MPI_Alltoallv             #calls     avg. bytes      time(sec)
                           7800      189728.0        520.050
                           7800      298937.7         72.334

---------------------------------------------------------------------------
Power-4 counter report for group 60.
 pm_hpmcount2, Hpmcount group for computation intensity analysis
---------------------------------------------------------------------------
    36926785537  FPU executed FDIV instruction (PM_FPU_FDIV)
  5877311570772  FPU executed multiply-add instruction (PM_FPU_FMA)
  3711681267059  FPU0 produced a result (PM_FPU0_FIN)
  3588092240172  FPU1 produced a result (PM_FPU1_FIN)
 12880187645642  Processor cycles (PM_CYC)
   570900890071  FPU executed store instruction (PM_FPU_STF)
 15182533250822  Instructions completed (PM_INST_CMPL)
  3813369774432  LSU executed Floating Point load instruction (PM_LSU_LDF)
```

The MPI Trace tool was developed and is supported by Bob Walkup of the IBM Watson Research Center. It is provided *as-is*, meaning that there is no formal support. However, Bob does provide support for it as his time permits. The MPI Trace package consists of a set of wrapper libraries and a couple of README files. It is available from Bob directly. You can reach him via e-mail at:

```
walkup@us.ibm.com
```

The hardware counter portion of the MPI Trace tool has dependencies on the bos.pmapi.* filesets, which are part of the standard AIX distribution.

Example C-2 on page 179 is a listing of both of the README files that come in the mpi_trace.tar file.

Example: C-2 MPI Trace README files

```
# cat README

There are three main files:

libmpitrace.a : wrappers for low-overhead MPI elapsed-time measurements
libmpihpm.a   : the trace wrappers above plus power-4 hpm counter data
libmpiprof.a  : provides elapsed-time call-graph data for MPI routines

To build these files, just type "make".

For 64-bit MPI, you have to build the trace-wrappers using the same
AIX version that you will use to run the code. The libraries above
should contain both 32-bit and 64-bit code; it is only necessary to
add -q32 or -q64 when linking. If you don't have AIX 5 and power-4,
just comment-out the hpm stuff in the makefile.

To use the trace wrappers, just link with libmpitrace.a, and then run the
application. By default each MPI task will create files in the working
directory with names: mpi_profile.0, mpi_profile.1, ... If you want to
reduce the number of output files, you can set the environment variable
TRACE_SOME to "yes" or "1" (or anything else), then you will get output
from only some of the tasks. The output is written when the application
calls MPI_Finalize().

To use libmpihpm.a, link with libmpihpm.a -lpmapi. Choose a power-4
performance counter group (for example group 5):

  export HPM_GROUP=5

then run the code. The pmapi library (bos.pmapi.lib) must be installed.
A list of counter groups is in the file power4.ref. By default, you will
only get counter group 60. The counters are started in MPI_Init(), and
stopped in MPI_Finalize(). You get an output file for each task:
mpi_profile_group5.0, mpi_profile_group5.1, ...
where the group number is identified, and the MPI task id is appended onto
the file name. In general you will have to run with several different values
for HPM_GROUP. Good choices are groups 5, 53, 56, 58, and 60.

To use libmpiprof.a, the code must be compiled with "-qtbtable=full" or
"-g" as an additional compiler option. The wrappers in libmpiprof.a use a
trace-back method to find the name of the routine that called the MPI
function, and this only works if there is a full trace-back table. Just
link the application with libmpiprof.a and run the code. Each MPI task
writes an output file, mpi_profile.0, ..., in the working directory when
the application calls MPI_Finalize().

For all of the versions you can choose to bind MPI tasks to processors if
```

```
you set an environment variable BIND_TASKS to anything. When BIND_TASKS is
set, the wrapper for MPI_Init will attempt to bind the MPI tasks to
processors in a way that spreads the tasks out over the available cpus
as much as possible.

The main objective for libmpitrace.a was to provide a very low overhead
elapsed-time measurement of MPI routines for applications written in
any mixture of Fortran, C, and C++. The overhead for the current
version is about 1 microsecond per call. The read_real_time() routine
is used to measure elapsed-time, with a direct method to convert
timebase structures into seconds. This is much faster than using rtc()
or the time_base_to_time() conversion routine.

The main objective for libmpiprof.a was to provide an elapsed-time profile
of MPI routines including some call-graph information so that one can
identify communication time on a per-subroutine basis. For example, if an
application has MPI calls in routines "main", "exchange", and "transpose",
the profile would show how much communication time was spent in each of
these routines, including a detailed breakdown by MPI function. This
provides a more detailed picture of message-passing time at the expense
of a bit more overhead: ~5 microseconds per call. In some applications
there are message-passing wrappers, and one would like the profile to
indicate the name of the routine that called the wrapper, not the name
of the routine that called the MPI function. In this case, one can set an
environment variable TRACEBACK_LEVEL=2, and then run the application
(which must be compiled with either -g or -qtbtable=full, and linked with
libmpiprof.a). It may also be useful to try higher levels such as
TRACEBACK_LEVEL=3, which associates the message-passing time with the
great-grandparent in the call chain.

Note: The Fortran trace wrappers are in lower case (mpi_send). If the
Fortran compiler option -qextname is used, all external names will have
an underscore added. To get the names in the wrappers to match, you will
have to re-compile mpi_trace.c with -DEXTNAME. Similarly, if Fortran source
is compiled with the -qmixed option, then the names of MPI routines are not
mapped to lower case, and you have to make sure that the wrappers have names
that match the case used in the application. Check the makefile for the
-DEXTNAME option.

Note: The current version is not thread-safe, so it should only be used
in single-threaded applications, or when only one thread makes MPI calls.
The wrappers could be made thread-safe by adding mutex locks around updates
of static data - which would add some additional overhead.

Recent fixes/features (March 2002):

  (1) Fixed the filename for output files. Older versions could fail in
      applications that use multiple communicators and task reordering.
  (2) All Fortran wrappers now have the Fortran profiling interface,
      so codes that use MPI_BOTTOM should run correctly.

  (3) Both 32-bit and 64-bit objects are packaged into the trace libraries.
      Must build the 64-bit library for the target AIX level.

  (4) Output from power-4 counters has been added (for AIX 5 and power-4
      only) with libmpihpm.a

      (May 2002)

  (5) The option to bind tasks to processors was added.

Please send corrections/suggestions to walkup@us.ibm.com.
```

# **cat README.timing_wrappers**

```
The purpose of timing_wrappers.o is just to get a summary of the aggregate
user and system time, and the aggregate memory used in a parallel
application, as requested in many benchmarks. To get timing and memory
info, link your app with timing_wrappers.o and run it.

The source file "timing_wrappers.c" has simple wrappers for MPI_Init and
MPI_Finalize (both the Fortran and C/C++ interfaces). All they do is to start
a wall-clock timer in MPI_Init(), and stop the timer in MPI_Finalize(). Also,
there are calls to getrusage() so that the memory utilization, user-time, and
system-time is measured for each MPI task. The wrapper for MPI_Finalize()
gathers all of the info to task 0, which prints the output to standard out.

You can use these wrappers to get timing information without any
source-code modification; just link with timing_wrappers.o. This
object file only has wrappers for MPI_Init() and MPI_Finalize(), so
it will not affect message-passing performance at all. You can either
add the timing_wrappers.o file to your makefile, or just re-link your
binary:

mv your.x old.x
mpxlf timing_wrappers.o old.x -o your.x  #(use mpxlf or mpcc)

If you have any linker options, like -bmaxdata, you need to use them when
you re-link. Also, if you have additional shared-libraries such as -qsmp,
they need to be included as well. There may be duplicate symbol warnings,
but the re-link should work, and you will get a useful summary in standard
out when MPI_Finalize() is called. If the re-link above is too complicated,
just add timing_wrappers.o to your makefile, and re-build the binary.
```

D

Integrating p690 in an IBM eServer Cluster 1600

Note: This appendix was originally published as *Configuring p690 in an IBM @server Cluster 1600*, REDP0187 by Carsten Hoffmann, Rene Akeret, Dino Quintero, and Subramanian Kannan.

IBM @server Cluster 1600 unifies the existing offerings of RS/6000 SP and Cluster Enterprise Servers (CES), and is managed by IBM cluster management software, the Parallel System Support Programs (PSSP). A collection from the following systems can be part of this Cluster, including:

- Legacy RS/6000 SP
- Stand-alone servers, such as RS/6000 S70/S7A and pSeries p680
- Rack-mounted servers, such as pSeries 660 model 6H1/6M1/6H0
- New LPAR-supported servers, such as pSeries 690 and pSeries 670

IBM @server Cluster 1600 builds on the success of RS/6000 SP technology, extends the benefits to more hardware building blocks, and provides flexibility for creating a new Cluster configuration.

IBM eServer pSeries 690

The pSeries 690 and p670 family of servers incorporates the advanced technologies available on the IBM @server line, as well as technology enhancements from IBM research divisions. The results are high-performance, high-availability servers that offer enhanced features and benefits.

The pSeries 690 and 670 are based on a modular design. It features a Central Electronics Complex (CEC) where memory and processors, power subsystem, and I/O drawers are installed. Optional battery backups can provide energy for an emergency shutdown in case of a power failure. The POWER4 chip offers advanced microprocessors, with an SMP design, in a single silicon substrate.

Building on the IBM @server zSeries heritage, the pSeries 690 supports logical partitioning. Each server can be divided into as many as 16 partitions, each with its own set of system resources, such as processors, memory, and I/O.

To use the new LPAR feature, the Hardware Management Console (HMC) is required. If the p690 runs in SMP mode only, you do not need an HMC. However, to integrate the p690 into the Cluster 1600, you must use an HMC.

For more details about the pSeries 690, see the *IBM @server pSeries 690 System Handbook*, SG24-7040.

What is an LPAR

Logical partitions (also called LPARs) are the most outstanding feature of the pSeries 690 because they enable the system to run several operating system instances concurrently. In a logical partition, an operating system instance runs with dedicated resources:

- Processors
- Memory
- I/O slots

These resources are assigned to the logical partition. The total amount of assignable resources is limited by the physically installed resources in the system. In an SP-attached p690 environment, an LPAR is seen from the CWS as a thin node.

What is an HMC

The IBM Hardware Management Console for pSeries (HMC) provides a standard user interface for configuring and operating partitioned and SMP systems. The HMC supports the system with features that allow a system administrator to manage configuration and operation of partitions in a system, as well as to monitor the system for hardware problems. It consists of a 32-bit Intel-based desktop PC with a DVD-RAM drive. In an SP-attached p690 environment, you must use at least one HMC for up to four p690 servers.

The HMC's main functions include:

- Providing a console for system administrators and service representatives to manage system hardware
- Creating and maintaining a multiple partition (LPAR) environment on a managed system
- Detecting, reporting, and storing changes in hardware conditions
- Acting as a service focal point for service representatives to determine an appropriate service strategy (Ethernet connection required)
- Displaying a virtual operating system session terminal (VTERM) for each partition

For further information about the HMC and its functions, consult the *HMC Operations Guide*, SA38-0590.

IBM eServer p690 in a Cluster

As shown in Figure D-1 on page 186, you can attach the p690 to the IBM @server Cluster 1600 either in Full System Partition mode (also known as SMP mode), or LPAR mode. Therefore, the configuration is different for each mode.

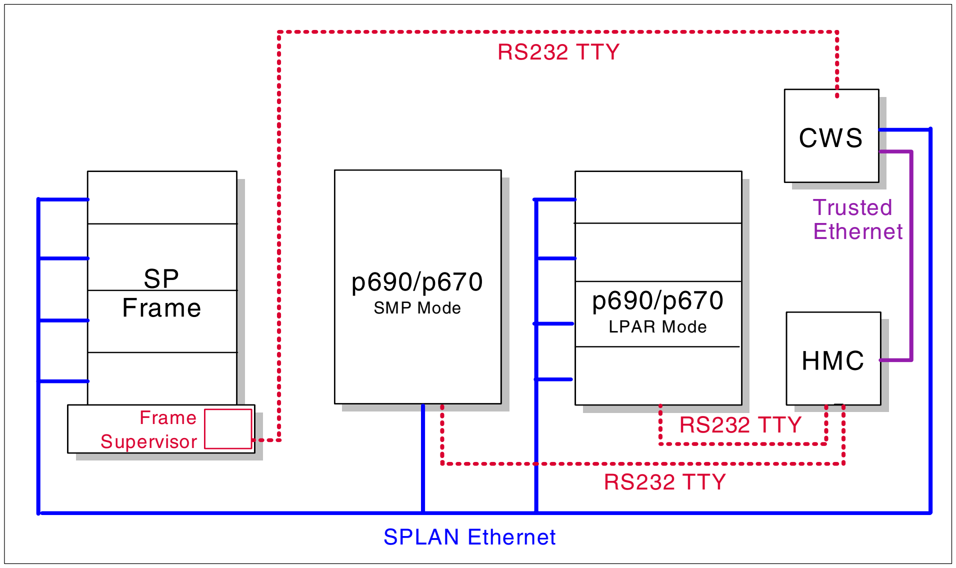


Figure D-1 p690 attachment in a Cluster

For detailed explanations about the network connections see “Connectivity between CWS, HMC, and p690” on page 187.

CWS, HMC, and p690 functions

In this section, we describe the functions of CWS, HMC and p690, and explain how they work together in an IBM @server Cluster 1600 environment.

The role of the CWS for an attached p690 server

The role of the CWS for an attached p690 configuration continues to be the same as it would be in a classic SP configuration or in an SP-attached server configuration. The difference is that the CWS is not directly connected via an RS232 serial line interface to the p690 server. The CWS is connected to the p690 via the HMC on an Ethernet LAN.

The role of the HMC in a Cluster 1600

In a Cluster 1600 the HMC has some additional tasks, as described in “What is an HMC” on page 185. The p690 server has no frame supervisor card like the SP frame or SP node supervisor cards. The HMC provides the CWS with all information needed for managing hardware control of the p690 server like an normal SP frame. The HMC provides the following functions:

- Shows the LPAR definitions to the CWS
- Provides the CWS with the hardware status of the p690 server (for example, power information)
- Manages the LPAR definitions of the p690 server

The HMC is connected via an RS232 line to the p690 server, and via an Ethernet connection to the CWS.

Connectivity between CWS, HMC, and p690

In this section, we explain the connectivity between CWS, HMC, and p690 in an IBM @server Cluster 1600 environment.

The RS232 connection

In classic SP environments, the CWS has a direct connection to each SP frame and to each SP-attached server to perform the following tasks:

- Controlling the hardware (for example, power on power off)
- Transferring the Kerberos tickets and password files
- Communicating with the firmware (for example, node conditioning)

Note: In an p690-attached configuration, there is no direct RS232 connection between the CWS and p690. The CWS is connected to the HMC via Ethernet, and the HMC is connected to the p690 server using an RS232 serial line.

The SPLAN

The SPLAN can be described as the management LAN for the SP system. The SPLAN is essential for the Cluster 1600 and for the following PSSP software management tasks:

- RSCT communications (for example, hosts responds)
- NIM operations
- Systems management communications

Note: The I/O drawer of the p690 server does not support the BNC type network adapters. If you want to integrate the p690 server into an existing IBM @server Cluster 1600 with a BNC SPLAN, you will need a HUB for the SPLAN to provide BNC and twisted pair (TP) cabling in one network.

The RMC LAN

The Resource Monitoring and Control (RMC) LAN is an Ethernet connection between the HMC and the p690 server in an LPAR or SMP mode. Using this connection, the HMC gathers data from active LPARs or SMP servers about the health of the system, and acts as a service focal point for service representatives to determine an appropriate service strategy. It is possible to have the RMC traffic over the SPLAN. For more information about the RMC LAN, read the *HMC Operations Guide*, SA38-0590.

Important: The RMC LAN is not mandatory! It is possible to use the function of the RMC LAN over the SPLAN.

The trusted LAN

Since there is no direct RS232 connectivity between the HMC and the CWS, it is *mandatory* to have a trusted network connection between the HMC and the CWS to transfer Kerberos tickets and password files. A separate trusted LAN is recommended, but not necessary. If your SPLAN is secure, you do not need to set up an additional trusted LAN connection. You can send your secure sensitive data over the existing SPLAN.

What is a trusted network

A trusted network is one where all hosts on the same network (LAN) are regarded as trusted, according to site security policies and procedures governing the hosts. Data on a trusted network can be “seen” by all trusted hosts (and users on trusted hosts), but the implied trust among and between the hosts assumes the data will not be intercepted or modified. Therefore, by way of implied mutual trust, traffic flowing across the trusted network is regarded as safe from unwanted or unintended interception or tampering. However, it does not imply that the data on the trusted network is itself private or encrypted.

Configuration examples

In this section, we show different possible network connections. In this appendix, however, all other definitions are focused on the example shown in Figure D-2 on page 189.

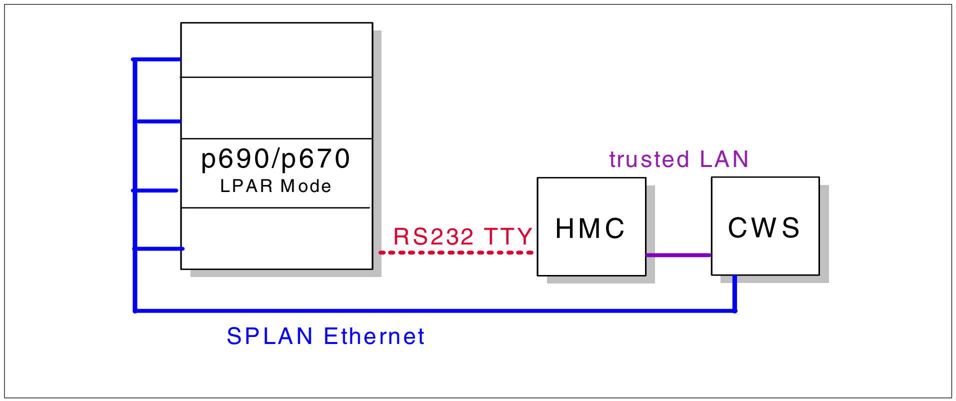


Figure D-2 The RMC LAN uses the SPLAN and the trusted network

Figure D-3 shows a Cluster with one physical network for RMC, SPLAN, and trusted LAN.

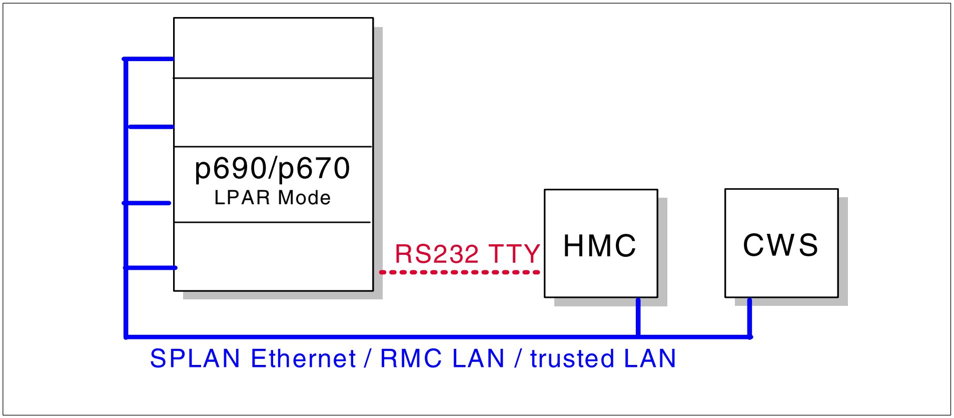


Figure D-3 Cluster with only one physical network

Figure D-4 on page 190 shows two physical networks, one for SPLAN and one for trusted network. RMC LAN uses the SPLAN.

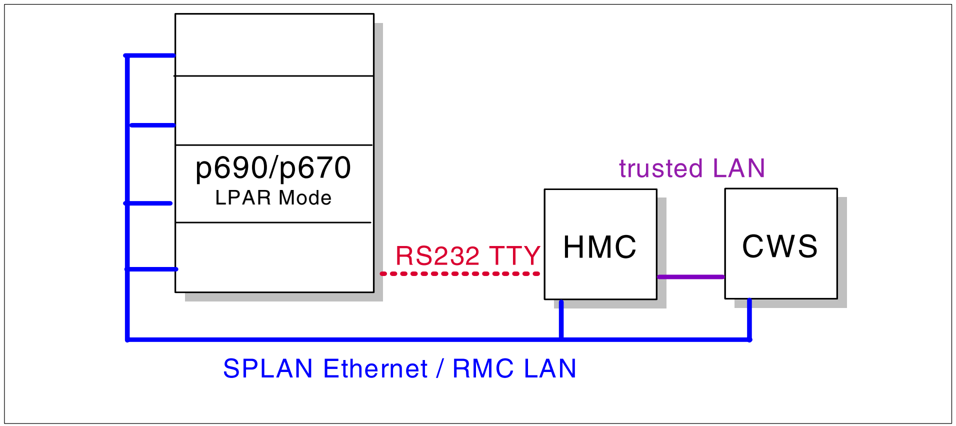


Figure D-4 Cluster with two physical networks

Figure D-5 shows three separate physical networks for SPLAN, RMC LAN, and trusted network.

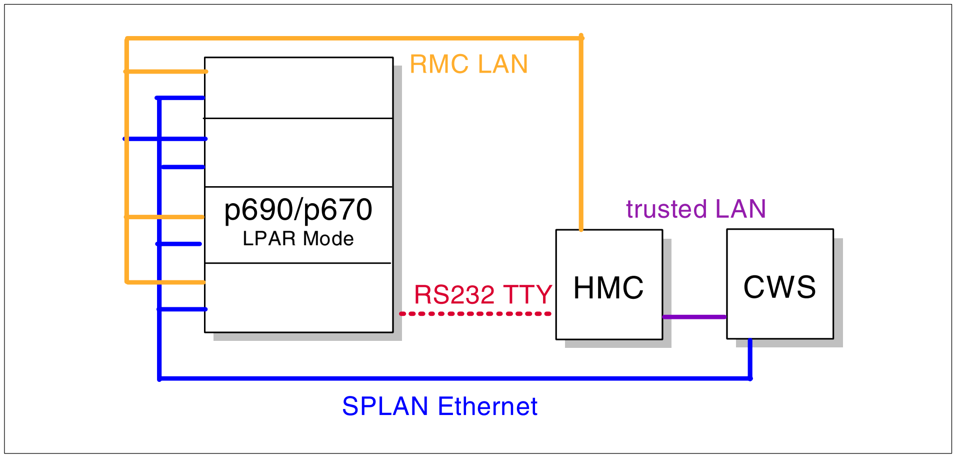


Figure D-5 Each function has its own physical network

Mapping LPAR numbers and node numbers

Depending on the operating mode of the p690 server, the CWS will see the system as either:

- An SP Frame with one thin node in the Full System Partition mode
- An SP Frame with several thin nodes in LPAR mode

Before you add the p690 server to the Cluster as an SP frame, you should first configure all the needed LPARs using the HMC. When the HMC is connected, and the CWS and the HMC are up, the hardmon daemon defines all LPARs as logical nodes when you add the Frame Information into the SDR from the CWS. Refer to “Limitations” on page 239 for more information about the rules.

If the p690 is used in LPAR mode, an LPAR is mapped as a thin node in the system configuration. In the SDR, the node is defined as a thin node and the p690 server is defined as an SP frame. Example D-1 shows the partial output from the **spmon -d** command on the CWS. In this example, the p690 is configured as frame 4 and each LPAR is configured as one thin node.

Example: D-1 spmon -d command output (partial)

```
------------------------------- Frame 4 -----------------------------
                      Host     Key     Env   Front Panel      LCD/LED
Slot Node Type  Power Responds Switch  Error LCD/LED          Flashes
---- ---- ----- ----- -------- ------- ----- ---------------- -------
  1    49  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  2    50  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  3    51  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  4    52  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  5    53  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  6    54  thin   on      yes     N/A    N/A  LCDs are blank     N/A
```

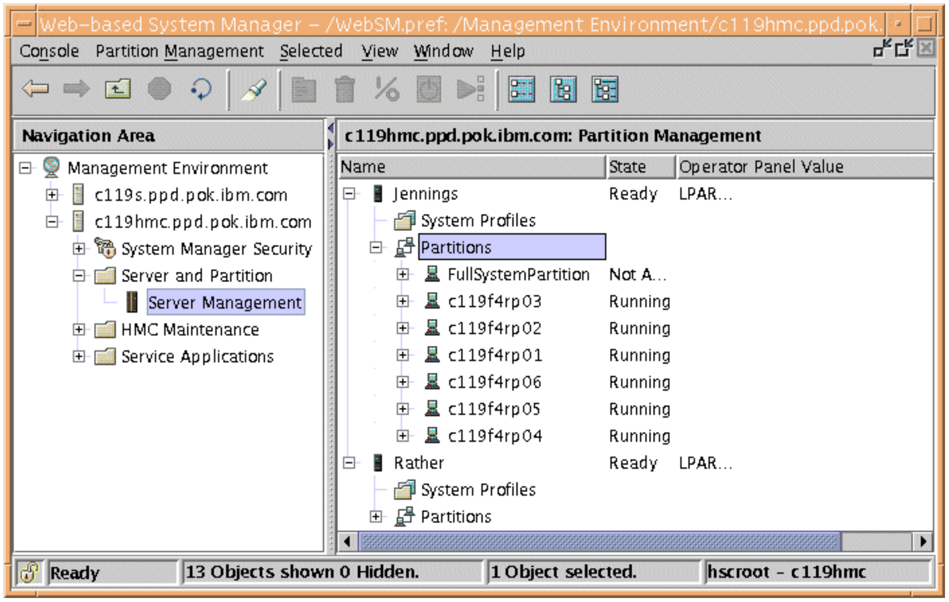
To identify which LPAR on the HMC is related to which node number on the CWS, compare the partial printout of the **splstdata -n** command, as shown in Example D-2, and the partition definitions of the HMC in Figure D-6 on page 192. As you see, the LPAR_name in the **splstdata** output is the same as the name of the partition list on the HMC screen shot.

Example: D-2 Matching LPAR name and node numbers

```
[c119s][/]> splstdata -n
                  List Node Configuration Information

node#frame#slot#slotsinitial_hostnamereliable_hostnamedce_hostname
default_routeprocessor_typeprocessors_installeddescriptionon_switch LPAR_name
--------------------------------------------------------- ------------
49 4  1  1  c119f4rp01.ppd.poc119f4rp01.ppd.po""              9.114.76.126MP
4  7040-6811 c119f4rp01
50 4  2  1  c119f4rp02.ppd.poc119f4rp02.ppd.po""              9.114.76.126MP
4  7040-6811 c119f4rp02
51 4  3  1  c119f4rp03.ppd.poc119f4rp03.ppd.po""              9.114.76.126MP
2  7040-6811 c119f4rp03
52 4  4  1  c119f4rp04.ppd.poc119f4rp04.ppd.po""              9.114.76.126MP
2  7040-6811 c119f4rp04
```

```
53 4   5   1   c119f4rp05.ppd.poc119f4rp05.ppd.po""              9.114.76.126MP
2  7040-6811 c119f4rp05
54 4   6   1   c119f4rp06.ppd.poc119f4rp06.ppd.po""              9.114.76.126MP
2  7040-6811 c119f4rp06
```


Web-based System Manager – /WebSM.pref: /Management Environment/c119hmc.ppd.pok.
Console Partition Management Selected View Window Help
Navigation Area
Management Environment
c119s.ppd.pok.ibm.com
c119hmc.ppd.pok.ibm.com
System Manager Security
Server and Partition
Server Management
HMC Maintenance
Service Applications
c119hmc.ppd.pok.ibm.com: Partition Management

Name	State	Operator Panel Value
Jennings	Ready	LPAR...
System Profiles		
Partitions		
FullSystemPartition	Not A...	
c119f4rp03	Running	
c119f4rp02	Running	
c119f4rp01	Running	
c119f4rp06	Running	
c119f4rp05	Running	
c119f4rp04	Running	
Rather	Ready	LPAR...
System Profiles		
Partitions		

Ready | 13 Objects shown 0 Hidden. | 1 Object selected. | hscroot - c119hmc

Figure D-6   Screen shot of the HMC where you find the LPAR names

For more Information about node numbering when an LPAR is reconfigured, refer to “Node numbering” on page 233.

Figure D-7 on page 193 shows the node numbering pattern that is valid for 9076 frames when using thin nodes. This node numbering pattern is also valid for p690 frames where each LPAR definition is a thin node. The LPAR numbers correspond to the slot numbers from thin nodes.

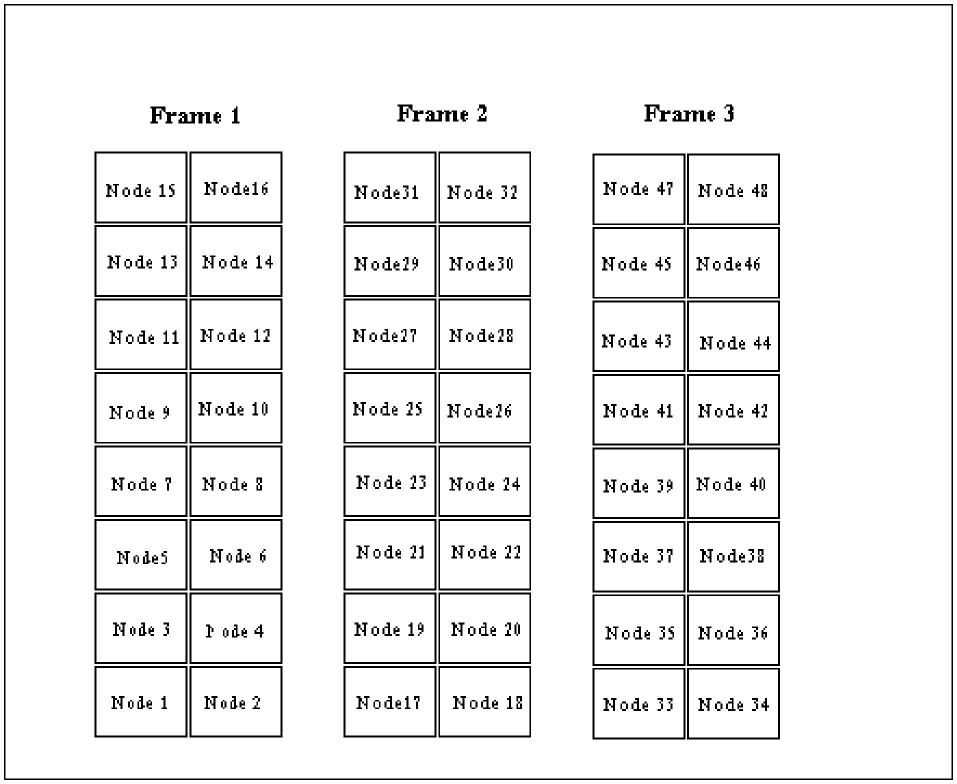


Figure D-7 Example node numbering in frames (9076 and p690)

Planning considerations

To attach a p690 to an IBM @server Cluster 1600, you need to do some additional planning. The planning considerations should include new hardware and software, as well as the existing configuration.

Control workstation

Since the IBM @server Cluster 1600 still needs a CWS, make sure the existing (or new) CWS meets the requirements. The software requirements are discussed in "AIX and PSSP software requirements" on page 206.

Supported hardware

The following hardware is currently supported for use as a control workstation:

- RS/6000 7024 Models E20 and E30

- RS/6000 7025 Model F30
- RS/6000 7025 Model F40
- RS/6000 7025 Model F50 and F80
- IBM @server pSeries 620 Model 6F1 – 7025-6F1
- RS/6000 7026 Models H10 and H50
- RS/6000 7026 Model H80
- IBM @server pSeries 660 Model 6H1 – 7026-6H1
- RS/6000 7043 Models 140, and 240
- RS/6000 7044 Model 170 – 44P-170
- IBM @server pSeries 660 Model 6C1/6E1

For a complete list with all features and limitations, see *RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment*, GA22-7281.

Control workstation failure

In an IBM @server Cluster 1600 environment with attached servers, the High Availability CWS (HACWS) is not supported. So to get around that, you might consider the following alternatives:

- Use LVM mirroring to avoid data lost by disk failure.
- Always have an up-to-date backup of the rootvg as well as /spdata.
- Put the /spdata in a separate volume group on a shared disk configuration.
- Use a different file system for PTFs and non-BOS lppsources to reduce the restore time of /spdata.

Network connections

As discussed in Chapter 8, “Control workstations”, in *RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280, the CWS needs a connection to the SPLAN. In an IBM @server p690 environment, the control workstation also needs a trusted network connection to the HMC.

The trusted network connection can be one of the following:

- A separate physical LAN
- The SPLAN network

Therefore, if your trusted network is a separate LAN, you need an additional Ethernet adapter on your control workstation. For further details refer to “Connectivity between CWS, HMC, and p690” on page 187.

Hardware Management Console

Since the Hardware Management Console (HMC) is also part of the Cluster, you need to consider the following.

Failure of the HMC

To avoid having the HMC become, potentially, a single point of failure, we recommend that you connect each p690 to two different HMCs. That way, in the event of an HMC failure, you are still capable of reaching the p690 through the second HMC connection. For further details, see “CWS with two HMCs and four p690s” on page 239. Depending on how many p690 servers you have, additional 8-Port async adapters may be required.

Network connections

As discussed in “Connectivity between CWS, HMC, and p690” on page 187, the HMC needs a trusted network connection to the control workstation. The trusted network connection can be either one of the following:

- A separate network
- An existing SPLAN connection

The RMC service can use the SPLAN. Therefore, if you want to use the RMC services, the HMC can use the trusted network connection between the control workstation and the HMC. In fact, the HMC does not need a connection directly to the SPLAN. However, if you choose to have this connection, some additional IP settings need to be made on the control workstation.

IBM eServer p690

There are several ways to attach the p690 servers to the IBM @server Cluster 1600:

- In Full System Partition mode (also called SMP mode)
- In LPAR mode
- In an SP Switch or SP Switch2-attached or switchless configuration

We discuss each method more fully in the following sections.

Full System Partition mode

If the p690 is running in Full System Partition mode, it is just like any other attached server. You will have one single AIX image using all the system resources.

LPAR mode

If LPARs are configured and used on the p690, the planning considerations are quite different from those of other attached servers, especially in regard to the following cases:

- Adapters in general

  You cannot share device adapters between active LPARs. So each adapter can only be assigned to one active LPAR at a time. Hence, you probably need more adapters in LPAR mode than running the p690 in SMP mode only. For more information about all supported adapters and their limitations and placements, refer to *RS/6000 & pSeries PCI Adapter Placement Reference*, SA38-0538.

- AIX system disks

  Each LPAR needs at least one disk for the AIX Operating System. You cannot share the SCSI adapter connected to that disk between active LPARs, so all disks connected to that adapter belong to the same LPAR. One I/O drawer provides four internal SCSI adapters. Up to four disks can be connected to a single SCSI adapter.

- SPLAN adapters

  You must connect each LPAR to the SPLAN. For further information, see “SPLAN” on page 197.

Note: LPAR resources defined to PSSP need to be uniquely tied to a single LPAR. Therefore, the rootvg, SPLAN adapter, and any other adapter defined to PSSP must be defined to only a single LPAR.

Switch-attached

The IBM @server p690 can be Switch- and SP Switch2-attached. We discuss the differences in this section.

- SP Switch

  All LPARs must be Switch-attached. Therefore, you cannot have some LPARs Switch-attached and some not. For SP Switch limitations, refer to “Limitations” on page 239.

- SP Switch2

  The SP Switch2 gives you more flexibility. All LPARs can be Switch-attached, but it is not necessary. In fact, with SP Switch2, you can have some LPARs not Switch-attached at all.

  For HPC environments, you might also consider using the SP Switch2 in a dual plane configuration for more performance. For SP Switch2 limitations, refer to “Limitations” on page 239.

Profiles

A *profile* allocates the system resources for a single LPAR. While creating a profile, you need to know exactly which resources—and how many—you want to allocate. Since an LPAR can have more than one profile, only one can be the default profile. PSSP uses only the default profile.

Note: For each LPAR, the profile marked as default is used by PSSP. All other profiles are ignored. If you want to change the profile used, make the new profile default and make appropriate configuration changes to PSSP.

SPLAN

Like any other SP node or attached server, each AIX image needs a SPLAN connection. If the p690 is running in Full System Partition mode, it needs only one Ethernet adapter for the SPLAN. When LPARs are configured and active, each LPAR needs a separate SPLAN connection.

Some earlier SP installations still use BNC cabling. Since the Ethernet adapters on the p690 support twisted pair cabling only, you may have to purchase a network HUB for connectivity.

Prepare the Hardware Management Console

The IBM Hardware Management Console for pSeries (HMC) is a dedicated system that provides a graphical user interface for configuring and operating single or multiple pSeries 690 systems. It consists of a PC system with a set of hardware management tools for configuration and partitioning. For more details about the HMC, refer to the *HMC Operations Guide*, SA38-0590.

Software levels

Before you start with any configuration, make sure the correct level of HMC software is installed. On the HMC WebSM interface, open the Help menu and click the About Web-based System Manager entry. The displayed build level must be equal or higher to the level described in *Parallel System Support Program for AIX: Read This First for New Users,* GI10-0641.

The latest HMC code can be obtained from IBM at:

http://techsupport.services.ibm.com/server/hmc

Serial connection

Each p690 needs a connection to at least one HMC.

HMC

If only one p690 is connected to the HMC, the first native serial port is used for the RS232 TTY connection. If more than one p690 is connected to one single HMC, an 8-Port or 128-Port Async PCI card is needed. The second native serial port is reserved for modem connection.

p690

On the p690, you can use the native serial Ports S1 and S2 for RS232 TTY connection. Since a single p690 can be connected to two HMCs, S1 is used for the first HMC connection and S2 for the second one.

System configuration

The System Configuration menu, located in the HMC Maintenance folder of the WebSM interface, offers an easy way to configure the HMC. The following tasks can be done through this menu:

- Customize console date and time
- Display console events
- Customize network settings
- Test the network connectivity
- Schedule operations, like a backup of all HMC settings
- Enable or disable remote command execution
- Enable or disable remote virtual terminal

Customize network settings

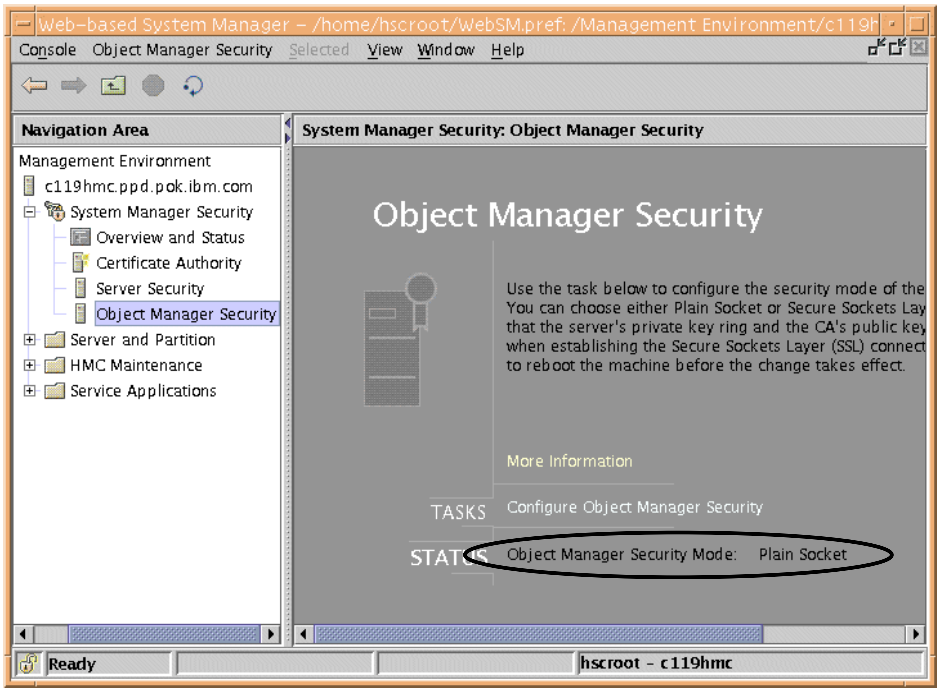
In the Network Configuration dialog, you must enter the settings of the trusted network connection. Since you can use the trusted network for the RMC communication, you do not need an additional network connection. However, you can configure a second Ethernet adapter for a dedicated SPLAN/RMC connection.

Enable or disable remote virtual terminal

In order to allow PSSP to open a virtual terminal (vterm), this option needs to be enabled. This function is used by PSSP to run commands such as **`s1term`**, **`sphrdwrad`**, **`spadaptr_loc`**, and **`nodecond`**. The remote vterm function used is also used by PSSP administrative processes to transfer sensitive data files.

Security settings

In an IBM @server Cluster 1600, the Object Manager Security Mode on the HMC needs to be set to plain socket. This is necessary for the PSSP hardware control and monitor functions. If the mode is set to secure socket layer (SSL), PSSP will not be able to perform the hardware monitor and control functions. The Object Manager Security menu is located in the System Manager Security folder of the WebSM interface. Figure D-8 shows how the settings should look.



Use the task below to configure the security mode of the You can choose either Plain Socket or Secure Sockets Lay that the server's private key ring and the CA's public key when establishing the Secure Sockets Layer (SSL) connect to reboot the machine before the change takes effect.

Object Manager Security Mode: Plain Socket

Figure D-8 Object Manager Security settings on the HMC

Domain name of p690 systems

Each p690 system is represented by a unique name, called a *domain name*. The domain name belongs to the Central Electronics Complex (CEC) and must not be confused with the LPAR names.

The HMC interface uses the domain name to display the physical system. The SDR contains the domain name to represent the p690 system. Figure D-9 on page 200 shows two attached p690 servers and their domain names. The domain name is also used in conjunction with the -d flag of the **spframe** command.

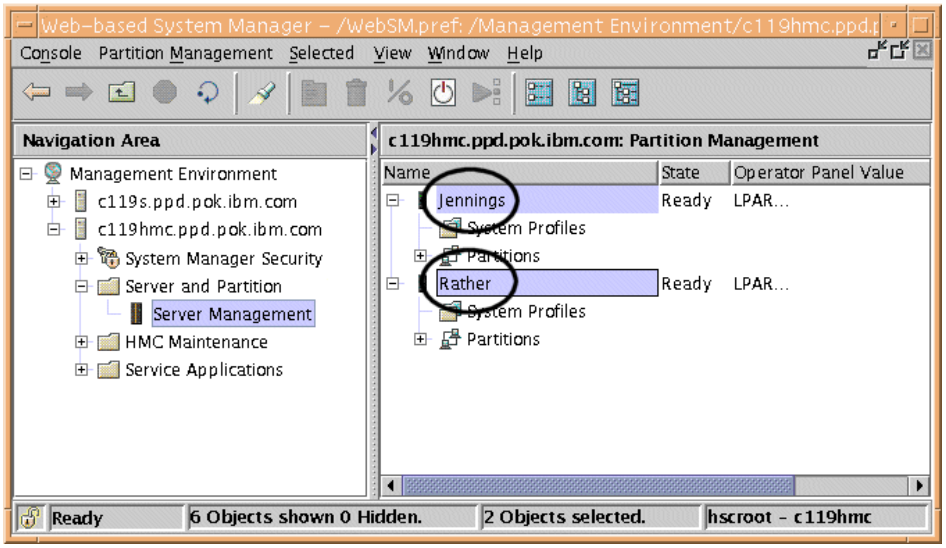


Figure D-9 HMC WebSM interface with configured p690 domain names

Note: The p690 domain name has nothing in common with the domain name used for DNS. The p690 domain name is not an IP-resolvable name. It is just a name for the IBM @server p690 CEC.

Prepare the p690

Before you can start with the integration of the p690 server into your Cluster, you have to check the firmware release of the p690 server. The next step would be to set up the SPLAN adapters and the Switch adapters as you planned in “The SPLAN” on page 187 and “Switch-attached” on page 196. There are rules for the Switch adapters, which we discuss in the following sections.

Required p690 firmware

There are three different firmwares installed on the p690 server.

Important: For the latest platform firmware version for the p690 server, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641.

To determine which firmware levels are installed on your p690 server, do the following:

If AIX is installed on your p690 system

1. Open a telnet or a vterm session to one of your LPARs or SMP server.
2. Next, log in as user root.
3. Enter the command shown in Example D-3.

Example: D-3 Determine which firmware is installed on the p690 server

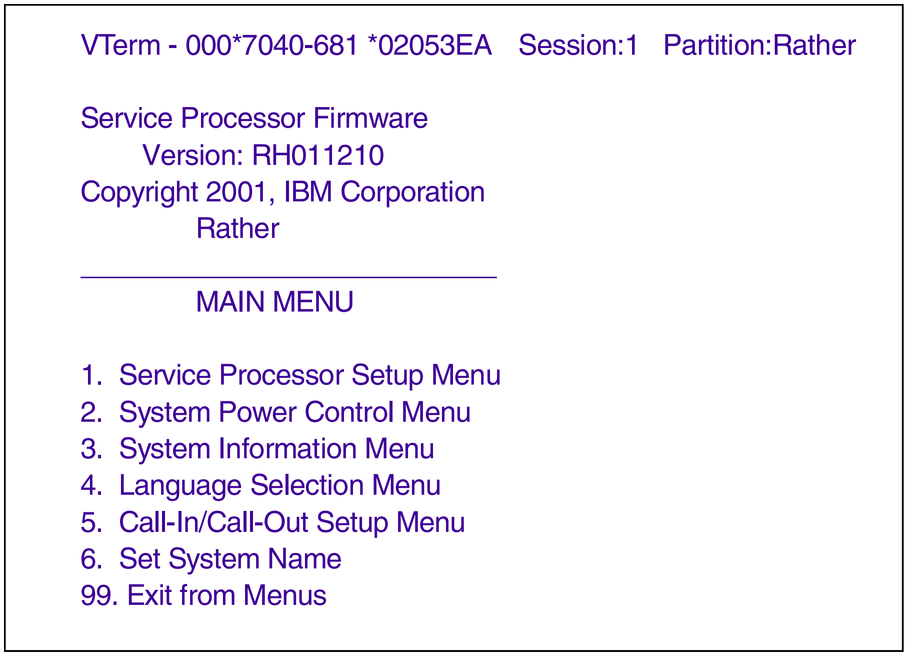
```
[c119f4rp05][/]> lscfg -vp|grep -p -e "Firmware"
      System Firmware:
        ROM Level.(alterable).......RH011204_srv1
        Version.....................RS6K
        System Info Specific.(YL)...U1.18-P1-H2/Y2
      Physical Location: U1.18-P1-H2/Y2

      Platform Firmware:
        ROM Level.(alterable).......RH011210
        Version.....................RS6K
        System Info Specific.(YL)...U1.18-P1-H2/Y1
      Physical Location: U1.18-P1-H2/Y1

      SPCN Firmware:
        ROM Level.(alterable).......0000RHE11081
        Version.....................RS6K
        System Info Specific.(YL)...U1.18-P1-H2/Y3
      Physical Location: U1.18-P1-H2/Y3
```

In the case where there is no operating system installed on your p690 system:

1. Shut down the p690.
2. From WebSM, click the right mouse button once on the CEC (for example, Rather)
3. Open a vterm session; after the vterm session is established, press Enter. You will see the service processor menu from the p690; refer to Figure D-10 on page 202 for more information.



```
VTerm - 000*7040-681 *02053EA   Session:1   Partition:Rather

Service Processor Firmware
      Version: RH011210
Copyright 2001, IBM Corporation
           Rather
________________________________
           MAIN MENU

1.  Service Processor Setup Menu
2.  System Power Control Menu
3.  System Information Menu
4.  Language Selection Menu
5.  Call-In/Call-Out Setup Menu
6.  Set System Name
99. Exit from Menus
```

Figure D-10 The opened vterm session to the p690 service processor

Attention: Firmware updates for the p690 system can only be done by IBM Customer Engineers.

Adapter placement for p690

There are different rules for the SP Switch and the SP Switch2 adapter.

Attention: There is no specific plug-in rule for the SPLAN Ethernet adapter placement. However, you should refer to *RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment,* GA22-7280, to see which Ethernet adapters are supported as SPLAN adapters.

For further information about adapter placements, refer to the *RS/6000 & pSeries PCI Adapter Placement Reference*, SA38-0538.

The I/O drawer of the p690 consists of two parts, and each one has 10 PCI slots. Figure D-11 on page 203 shows the rear view of one p690 I/O drawer.

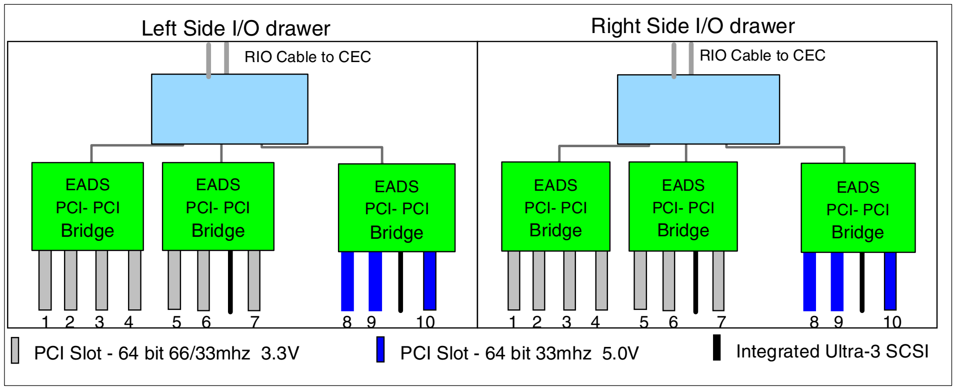


Figure D-11 Logical view (rear view) of an I/O adapter drawer of a p690

SP Switch (TB3PCI adapter) placement rules

The following are the placement rules for the SP Switch adapter:

- One per LPAR
- For the maximum number of LPARs supported per p690 server, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641
- No non-switched LPARs
- Plug-in slot 8 - EADS 3
- Two per I/O adapter drawer (one per I/O planar, slot 8 only)
- Takes up to two slots due to large heatsink
- *Not* to be mixed with the SP Switch2 PCI Attachment Adapter

Restriction: The TB3PCI adapter is a 5-volt adapter and can only be placed in slot 8 in the left side I/O drawer and slot 8 in the right side I/O drawer.

For best performance, we recommend that no other adapters be plugged in slots supported by this EADS chip, since it already has an integrated SCSI (they are connected to the SCSI 4-packs on the front side of the I/O drawer). The SPLAN adapter for this LPAR does not need to be in the same drawer. Figure D-12 on page 204 shows the SP Switch adapter configuration for two LPARs.

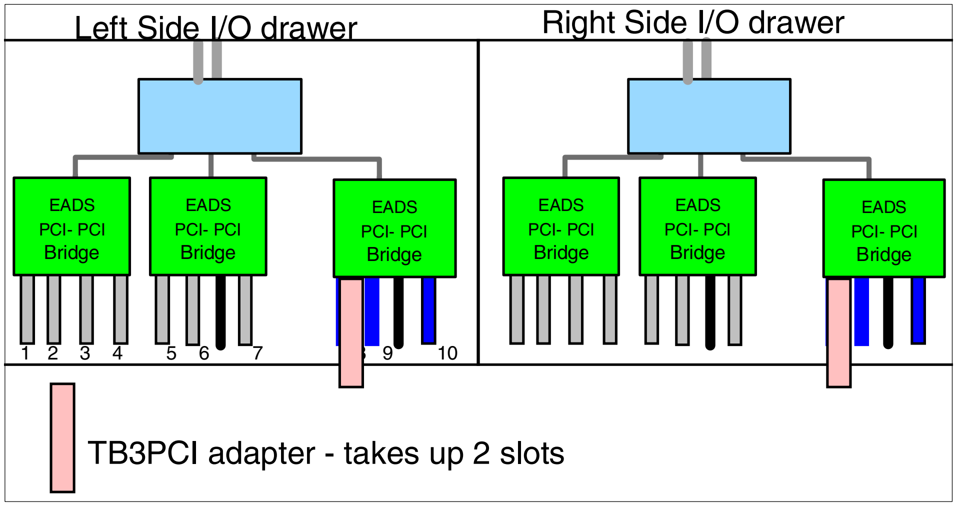


Figure D-12 I/O adapter drawer with two SP Switch adapters

SP Switch2 PCI Attachment Adapter placement rules

The following are the placement rules for the SP Switch2 adapter:

- For the maximum number of LPARs supported per p690 server, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641
- Plug-in slot 3 or 5, or both if on separate LPARs
- Up to two SP Switch2 adapters per LPAR
- One I/O adapter drawer supports four switched LPARs
- The SPLAN Ethernet adapter may be plugged in the same EADS
- Takes up two slots due to large heatsink

Restriction: The SP Switch2 adapter can only be used in slots 3 and 5 in the left side I/O drawer and in slots 3 and 5 in the right side I/O drawer.

For best performance, we recommend that no other adapters be plugged into slots supported by this EADS chip, except for the SPLAN Ethernet adapter. In Figure D-13 on page 205, we show the maximum usage of one I/O adapter drawer. If you want to use more than the shown I/O adapters (for example, Fibre Channel (FC) or LAN adapter), you need to use additional I/O drawers.

Single Switch plane

Figure D-13 shows an SP Switch2 adapter using a single switch plane configuration for four LPARs.

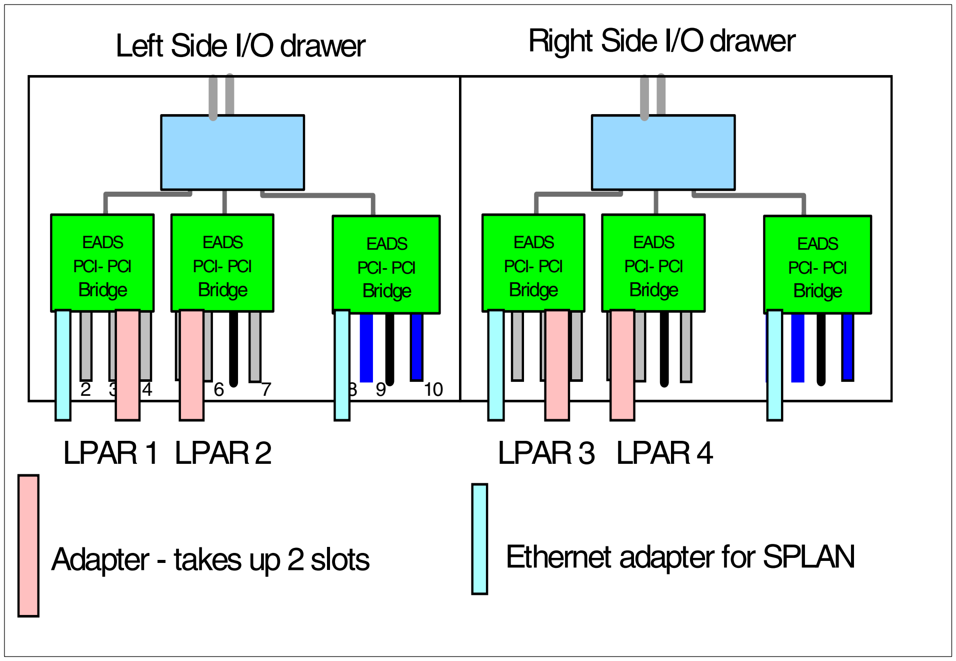


Figure D-13 Single-plane SP Switch2 with four SPLAN adapters

Dual Switch plane

It is possible to have two Switch adapters per LPAR. Figure D-14 on page 206 shows two SP Switch2 adapters configured for two LPARs.

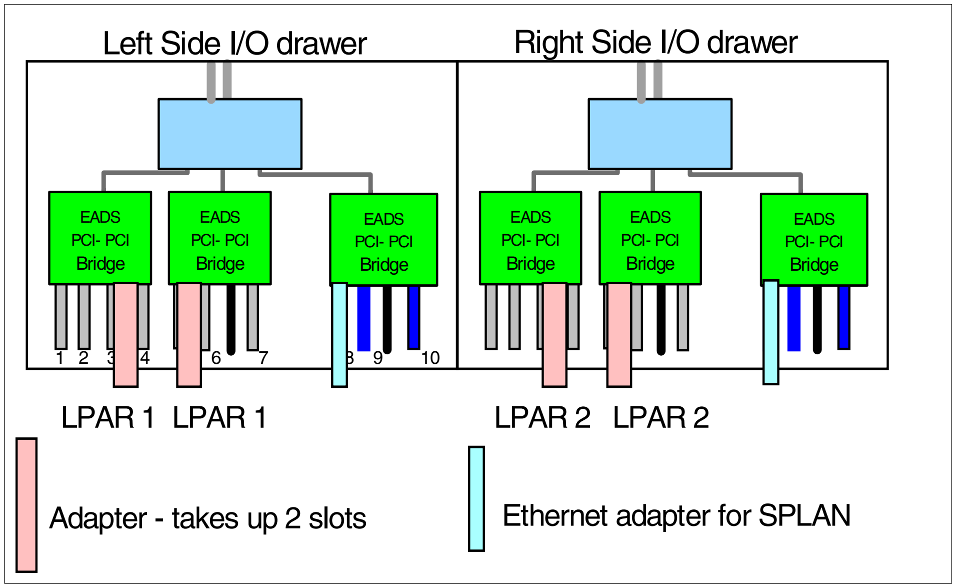


Figure D-14 Dual-plane SP Switch2 with two SPLAN adapters

Prepare the control workstation

In order to attach p690 servers to the IBM @server Cluster 1600, you must prepare the control workstation (CWS) and the Boot Install Servers (BIS). The preparation might also include some planning considerations as discussed in “Planning considerations” on page 193.

For further details on how to prepare the control workstation, see Chapter 1, “Overview of the installation and migration process”, in the *PSSP for AIX: Installation and Migration Guide*, GA22-7347.

AIX and PSSP software requirements

The p690 Clustering requires a certain level of AIX and PSSP software. On the control workstation and the Boot Install Servers, AIX 5L Version 5.1 and PSSP 3.4 is required. Table D-1 on page 207 shows the required AIX and PSSP software.

Table D-1 Important software requirements for p690 Clustering

Product	Software level and filesets
AIX 5L V5.1	► Java130.rte ► Java130.xml4j ► csm.clinet ► openCIMOM-.61.aix5.1.noarch.rpm (Located on the AIX toolbox for Linux applications CD) ► For the correct APAR level, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641
PSSP 3.4	► For the correct APAR level, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641

The list of required software on the control workstation is covered by Chapter 1 “Overview of the installation and migration process, Step 18: Install PSSP prerequisites and Step 19: Install PSSP on the control workstation” in the *PSSP for AIX: Installation and Migration Guide*, GA22-7347.

Note: For the latest supported software levels, check the *Read This First* document. You can also visit the IBM Web page:

http://www.ibm.com/servers/eserver/pseries/library/sp_books/pssp.html

Create or update the lppsource

The p690 Clustering requires AIX 5L Version 5.1, so you probably have to create a new lppsource or update the existing one. Since AIX 5L Version 5.1 also provides support for Red Hat Package Manager (RPM) files, the directory tree of the lppsource looks different than on earlier AIX versions. The **find** output in Example D-4 shows the enhanced directory structure.

Example: D-4 lppsource directory structure on AIX 5L V5.1

```
[c119s][/] find /spdata/sys1/install/aix51/lppsource -type d
/spdata/sys1/install/aix51/lppsource
/spdata/sys1/install/aix51/lppsource/installp
/spdata/sys1/install/aix51/lppsource/installp/ppc
/spdata/sys1/install/aix51/lppsource/rpm
/spdata/sys1/install/aix51/lppsource/rpm/ppc
```

The installp/ppc directory holds all regular AIX LPP filesets. The rpm/ppc subdirectory contains all rpm files that are currently provided by the AIX toolbox for Linux applications CD.

A description of how to create or update an lppsource is provided in "Step 14: Copy the AIX LP images and other required AIX LPPs and PTFs" in Chapter 1, "Overview of the installation and migration process, of the *PSSP for AIX: Installation and Migration Guide*, GA22-7347.

Note: Since the p690 may require some additional device filesets, make sure that they all are installed in the lppsource. Also be aware that some device drivers might not be provided on the base AIX 5L Version 5.1 media.

Create or update the SPOT

In order to install the p690 servers, the SPOT needs to be at the same level as the lppsource. Maintaining and updating SPOT is covered in "Task E: Update the SPOT when installing AIX BOS service updates" in Chapter 7, "Performing software maintenance", of the *PSSP for AIX: Installation and Migration Guide*, GA22-7347.

Software coexistence

In an existing SP environment, at least the control workstation has to be updated to AIX 5L Version 5.1, PSSP 3.4, and the APAR level mentioned in *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641. Table D-2 shows which AIX and PSSP levels can coexist in the same environment.

Table D-2 Coexistence of AIX and PSSP levels

CWS	Nodes	
AIX 5L Version 5.1 (32-bit kernel) PSSP 3.4	AIX 5L Version 5.1 (32-bit kernel)	PSSP 3.4
	AIX 4.3.3	PSSP 3.4
		PSSP 3.2
		PSSP 3.1.1

Accessing the Hardware Management Console

From the control workstation, you can access the HMC in two different ways:

- Using Perspectives
- Using a WebSM client

Using Perspectives

Perspectives allows you to launch WebSM through its own interface as follows:

1. Double-click the Hardware: SP-attached Servers icon.
2. Scroll to the Frames and Switches pane.
3. Select the p690 frame icon.
4. Open the Action menu.
5. Click the Open HMC interface menu entry.
6. On the HMC login window that appears, enter the user name and password.

Using WebSM

Since the HMC is managed through WebSM interfaces, all tasks can also be done on the control workstation or any other machine running a WebSM client. However, the Service Agent UI registration/customization task from the Service Agent menu can only be launched from the HMC console itself.

For the correct level of WebSM to connect to the HMC, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641.

Install WebSM on the control workstation

In order to access the HMC WebSM interface from the control workstation, you have to install WebSM on the CWS. Table D-3 lists the filesets required for WebSM.

Table D-3 WebSM filesets on the control workstation

AIX fileset	Description
sysmgt.websm.accessibility	WebSM Accessibility Support Web-based System Manager
sysmgt.websm.apps	Web-based System Manager Applications
sysmgt.websm.diag	Web-based System Manager Diagnostic Applications
sysmgt.websm.framework	Web-based System Manager Client/Server Support
sysmgt.websm.icons	Web-based System Manager Icons
sysmgt.websm.rte	Web-based System Manager Run-time Environment
sysmgt.websm.webaccess	WebSM Web Access Enablement. Also includes the WebSM client for Windows-based PCs

After installing the WebSM filesets to launch WebSM, issue the **`wsm`** command to get the WebSM interface.

Add the HMC host object

To access the HMC from your WebSM client, you have to add the HMC host to your WebSM interface. For that, select **Console** -> **Add** -> **Hosts** and enter the host name or the IP address of the HMC as shown in Figure D-15.

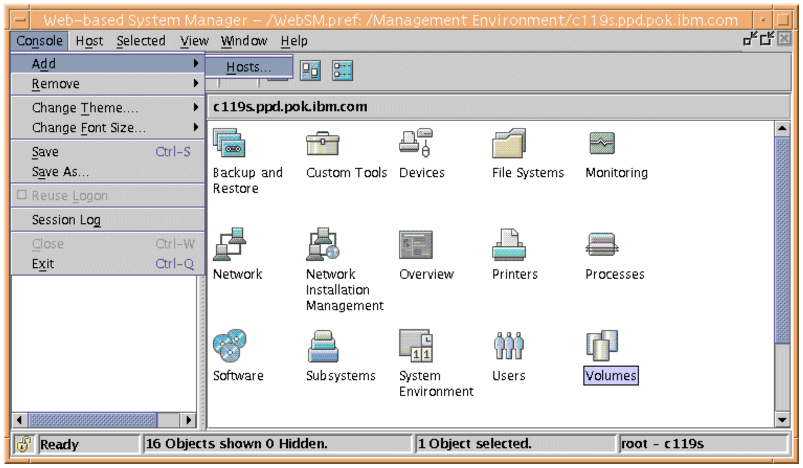


Figure D-15 Add host to WebSM interface

To access the HMC, double-click the newly created host object. In the dialog window that appears, enter the user name and the password. In general, the hscroot user is used.

Using WebSM on a Windows-based PC system

The fileset sysmgt.websm.webaccess provides a client for Windows-based PCs. If the fileset is installed on the control workstation, the PC client code can be found as /usr/websm/pc_client/setup.exe.

Install via Web browser

If the Web server is running on the control workstation and configured for WebSM, you can download the client code with a Web browser as follows:

1. Start a Web broker and enter the URL:

```
http://<CWS_hostname>/pc_client/pc_client.html
```

2. Download the client code to your PC.
3. Install the client code.

Install via ftp connection

If no Web server is running on the control workstation, the PC client code can obtained via **`ftp`**:

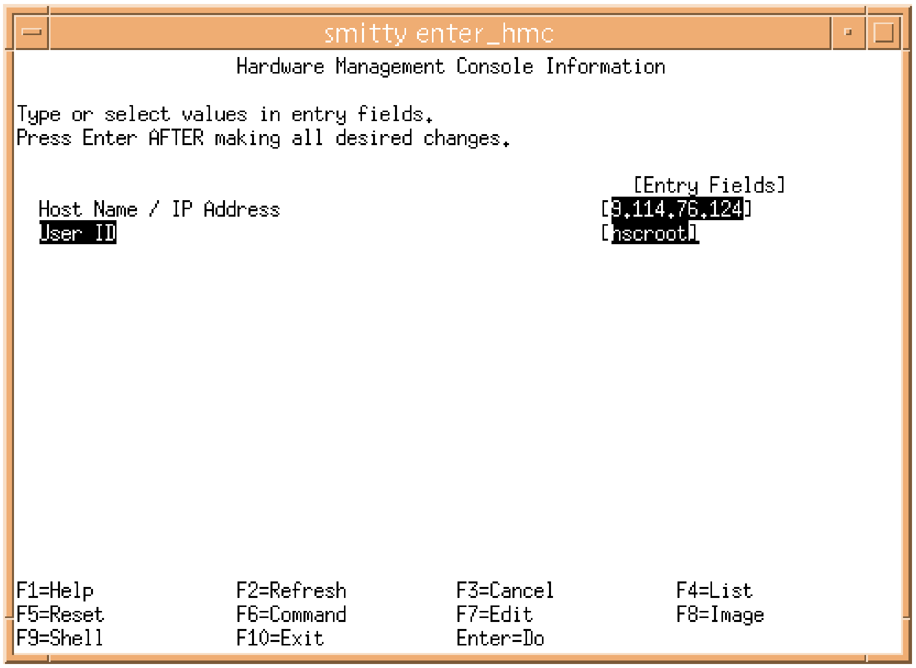
1. Open a ftp session to the control workstation.
2. Change to the directory /usr/websm/pc_client.
3. Transfer the setup.exe file to your PC.
4. Start the installation by double-clicking the program icon.

For a description of how to add the HMC host object to the WebSM client interface, see “Add the HMC host object” on page 210.

Set hardmon authentication

In a p690 cluster environment, hardmon does not directly talk with the p690 hardware. To manage the p690 hardware, hardmon communicates over the trusted network connection with the HMC (refer to “Hardware monitoring” on page 213 for further information).

Therefore, hardmon needs an user name and password when establishing a remote client session with an HMC. You can set the HMC user name and password in two different ways: by using either the SMIT **`smitty enter_hmc`** fast path, or by using the **`sphmcid`** command. Figure D-16 on page 212 shows the SMIT panel for setting the HMC user name and password.


smitty enter_hmc

```
                    Hardware Management Console Information

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
  Host Name / IP Address                            [9.114.76.124]
  User ID                                           [hscroot]

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure D-16 Setting HMC user name and password

The **sphmcid** command stores the settings in the /spdata/sys1/spmon/hmc_passwd file. For each HMC connected to the CWS, one line with the encrypted password and HMC IP address appears in this file.

Query HMC user name

You can use the **sphmcid** command to query the user name used by hardmon, as shown in Example D-5.

Example: D-5 Use sphmcid command to query the stored HMC user name

```
# sphmcid 9.114.76.124
9.114.76.124      hscroot
```

Define switch node numbers

In an SP Switch or a switchless environment, you must define a switch node number for each LPAR in the IBM @server p690 server. You can do that by manually editing the /etc/switch.info file. The file must include one line entry for each attached LPAR.

The format of the /etc/switch.info file is:

node_number switch_node_number

or

frame_number,slot_number switch_node_number

Example D-6 shows a three-frame configuration:

- Frame 1 is an attached S70. The first available node number is 1. The corresponding switch number is also 1.
- Frame 2 is an attached S80. The first free node number in frame 2 is 17. The used switch node number is set to 5.
- Frame 3 is an LPAR p690 with four LPARs. All four LPARs need a switch node number.

Example: D-6 Example of the /etc/switch.info file

```
cat /etc/switch.info
1 1
17 5
33 25
34 19
35 21
36 23
```

For further details about the /etc/switch.info file, refer to *PSSP for AIX: Command and Technical Reference*, SA22-7351.

You can skip this configuration step if you are using SP Switch2 only, or you have a switchless Clustered Enterprise Server system (CES).

PSSP changes

Integrating the IBM @server p690 introduces some changes to the PSSP software. Since the p690 represents a new generation of hardware, PSSP has been enhanced to manage this new server.

Hardware monitoring

Since the p690 does not have an SP node supervisor card, SP hardware monitoring is enhanced to manage the new hardware as well as the existing SP hardware. The p690 Clustering also introduces a new hardware protocol, named the HMC protocol. Table D-7 on page 214 shows the different hardware protocols used in an IBM @server Cluster 1600 environment.

Example: D-7 Different hardware protocols used by hardware monitoring

Protocol name	**Servers**
SP	SP nodes
SAMI	RS/6000 S70, S7A, and S80 or IBM @server pSeries 680 servers
CSP	RS/6000 H80, M80, and IBM @server pSeries 660 servers (6H0, 6H1, and 6M1)
HMC	IBM @server pSeries 670 and 690 servers

Hardmon subsystem

In earlier systems, the hardmon daemon running on the CWS talked directly to the frame or node supervisor card. In a p690 environment, the hardmon daemon sends and gets all hardware relevant information from HMC through the hmc daemon (hmcd). The hardmon daemon polls every five seconds to query the node status.

Dataflow

The hardmon uses different protocols to communicate with the hardware. Figure D-17 on page 215 shows the dataflow of the hardmon communication and all available protocols.

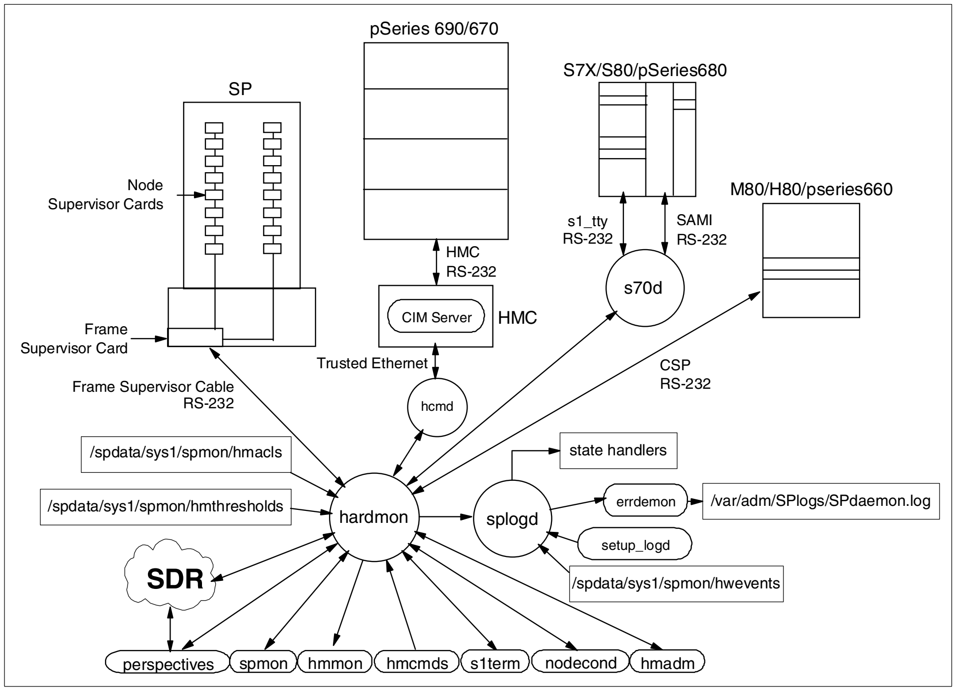


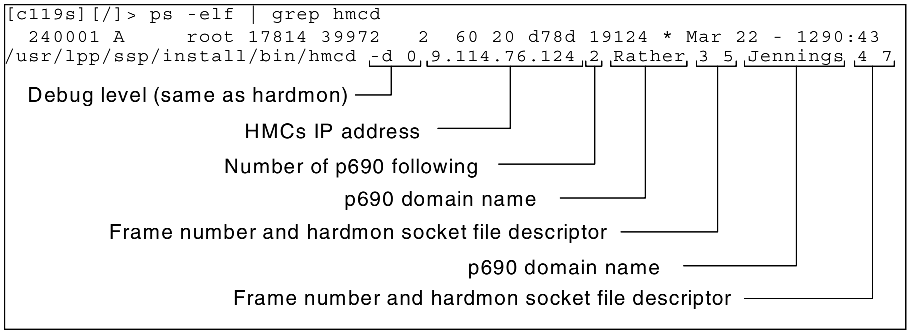
Figure D-17 Dataflow of the hardmon subsystem

HMC daemon

The hmc daemon (hmcd) is the program that indirectly interfaces to the p690 hardware through the HMC Common Information Model (CIM) Server. In fact, the hmcd acts as a frame and node supervisor and accepts all control commands from hardmon.

Once the hardmon recognizes the existence of p690 servers in its configuration, it starts the hmc daemon. Hardmon does it for each unique control_ipaddrs attribute in the SDR where the hardware_protocol is also set to HMC. For more information about the new SDR attributes control_ipaddrs and hardware_protocol, refer to “Frame Class” on page 220.

Each hmcd daemon belongs to a specific HMC connection. So, one hmcd daemon is responsible for multiple p690 systems connected to this HMC. Figure D-18 on page 216 shows the **ps** command output of an hmcd daemon running on the CWS.



```
[c119s][/]> ps -elf | grep hmcd
  240001 A      root 17814 39972    2   60 20 d78d 19124 * Mar 22 - 1290:43
/usr/lpp/ssp/install/bin/hmcd -d 0 9.114.76.124 2 Rather 3 5 Jennings 4 7
```

Debug level (same as hardmon)
HMCs IP address
Number of p690 following
p690 domain name
Frame number and hardmon socket file descriptor
p690 domain name
Frame number and hardmon socket file descriptor

Figure D-18   ps command output of a running hmcd daemon

Log files

The hmcd daemon logs its debug and error information in separate log files. The log files are located in /var/adm/SPlogs/spmon/hmcd. The naming convention of the log files are as following:

Log file	hmcd.<IP address>.log.<ddd>
IP address:	IP address of the HMC
ddd:	Julian date of the date the log file was opened by the hmcd daemon

Dataflow

Figure D-19 shows the dataflow between hardmon and hmcd.

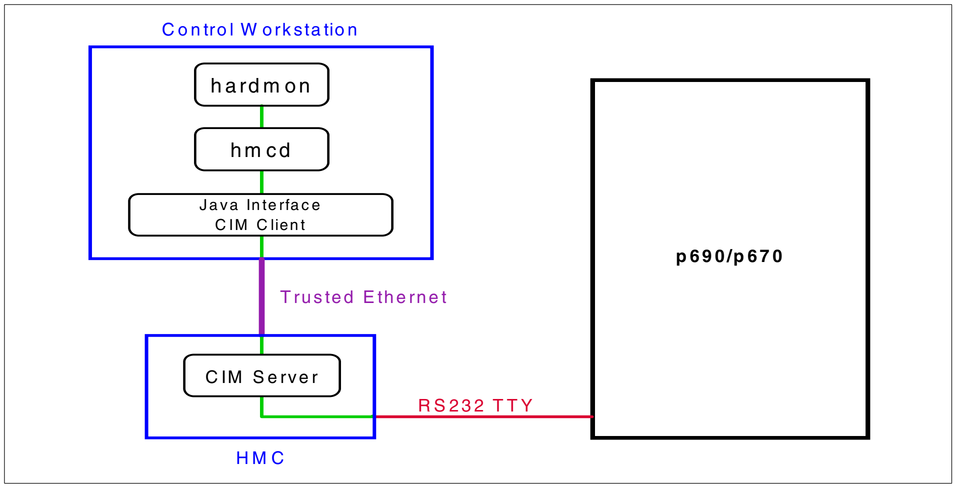


Figure D-19   hardmon and hmcd dataflow between CWS, HMC, and p690

As you can see, hardmon talks directly to hmcd. The hmcd gets the information from the CIM server running on the HMC. The connection between the control workstation and the HMC is made via a trusted Ethernet network.

SPLAN adapter

The IBM @server p690 introduces a new way to configure the SPLAN adapter. It is now recommended to configure the SPLAN adapter through its unique hardware location code.

Note: In an IBM @server p690, it is no longer required to have the first Ethernet adapter (en0) configured as the SPLAN adapter. To uniquely define the Ethernet adapter, PSSP uses the hardware physical location codes.

You can use SMIT or the **spadaptr_loc** command to get the hardware physical location codes. For further details about the **spadaptr_loc** command, refer to "New commands and new command flags" on page 223.

To get the hardware location codes through SMIT, use the following menu path: **RS/6000 SP System Management -> RS/6000 SP Configuration Database Management -> Enter Database Information -> Node Database Information -> Get Adapter Physical Location Information (pSeries 690 Only)**.

Example D-8 shows the hardware location codes in the **splstdata** output.

Example: D-8 splstdata output for hardware location codes

```
[c119s][/]> splstdata -a
                  List LAN Database Information
node#  adaptnetaddrnetmaskhostnametypet/r_rateenet_rateduplex
other_addrsadapt_cfg_statusphysical_locationSPLAN
----- -------------------------------------------------------------------
-------------------------------------------------
33     en09.114.76.33255.255.255.128c119f3rp01.ppd.poktpNA100full
""        ""     U1.9-P1-I1/E11
34     en09.114.76.34255.255.255.128c119f3rp01.ppd.poktpNA100full
""        ""     U2.9-P2-I1/E11
35     en09.114.76.35255.255.255.128c119f3rp05.ppd.poktpNA100full
""        ""     U1.5-P1-I1/E11
36     en09.114.76.36255.255.255.128c119f3rp09.ppd.poktpNA100full
""        ""     U1.5-P2-I1/E11
```

Since the order of all adapters is given by the firmware, the adapter logical name can change while reconfiguring the LPARs. To avoid the renaming of the Ethernet adapter logical names while allocating or deallocating adapters in LPARs, PSSP automatically assigns the adapter logical names using hardware physical location codes. This process is explained in the following scenario:

- In Figure D-20, (A) shows one LPAR with three configured Ethernet adapters. You can see the AIX device name (enX) and its corresponding hardware location (U1.X-PX-IX/E1).
- (B) shows the scenario after adding a new adapter. The device name gets the next free logical name, which is en3.
- (C) shows the AIX device names after reinstalling the LPAR. As you can see, the ordering of the device names is changed. Now PSSP automatically reassigns the logical names, as shown in (B), using hardware physical location codes defined in the System Data Repository (SDR).

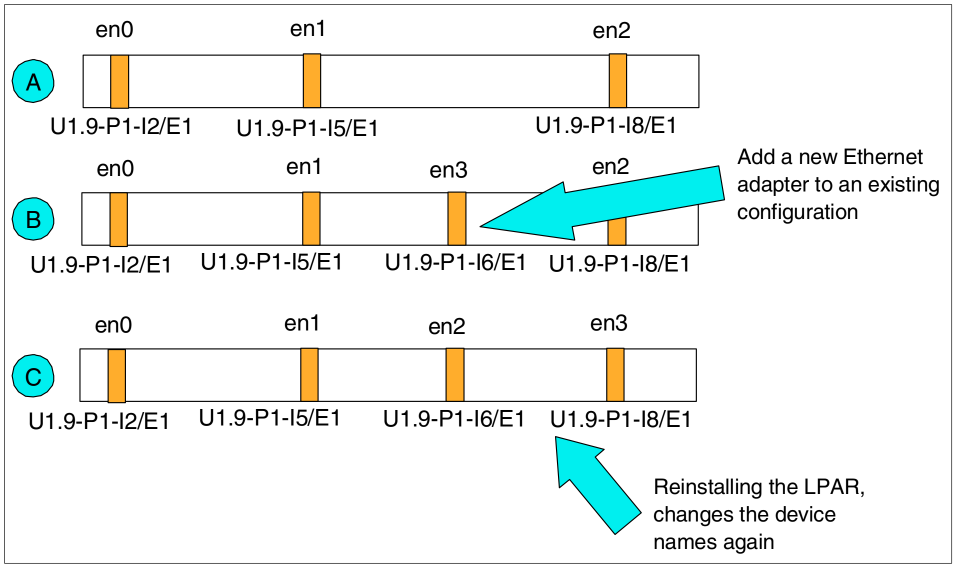


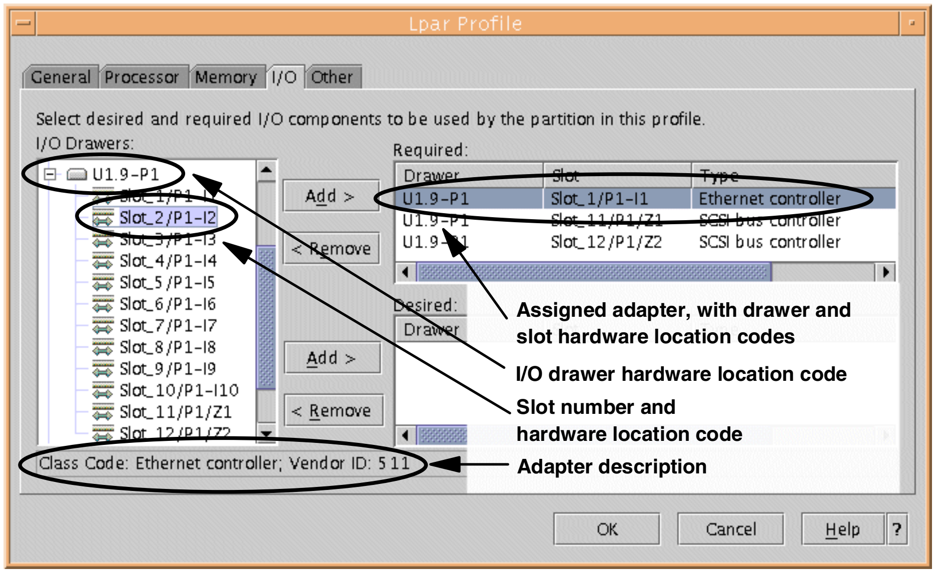
Figure D-20   Physical location codes and logical AIX device names

Note: The hardware physical location codes shown on the HMC interface are slightly different from the ones you see in the System Data Repository (SDR).

Figure D-21 shows how to get the Ethernet adapter hardware location codes from the HMC. Use the following procedure to get the hardware location codes on the HMC:

- Click the I/O drawer icon and select the slot where the Ethernet adapter is plugged in.
- The I/O drawer icon gives you the hardware location code of the drawer (for example, U1.9).
- The selected slot number shows the physical location code of the adapter (P1-I2). The adapter description is shown at the bottom of the window.
- Since this Ethernet adapter has only one port, it gives you the extension /E1. A 4-port Ethernet adapter would give the extensions E1 through E4.

Merging all the information together gives you the complete hardware adapter location code of the Ethernet adapter, which is U1.9-P1-I1/E1.


Lpar Profile
General
Processor
Memory
I/O
Other
Add >
< Remove
Add >
< Remove
OK
Cancel
Help
?
Assigned adapter, with drawer and slot hardware location codes
I/O drawer hardware location code
Slot number and hardware location code
Adapter description

Select desired and required I/O components to be used by the partition in this profile.

I/O Drawers:

```
⊟ U1.9-P1
    Slot_1/P1-I1
    Slot_2/P1-I2
    Slot_3/P1-I3
    Slot_4/P1-I4
    Slot_5/P1-I5
    Slot_6/P1-I6
    Slot_7/P1-I7
    Slot_8/P1-I8
    Slot_9/P1-I9
    Slot_10/P1-I10
    Slot_11/P1/Z1
    Slot_12/P1/Z2
```

Required:

Drawer	Slot	Type
U1.9-P1	Slot_1/P1-I1	Ethernet controller
U1.9-P1	Slot_11/P1/Z1	SCSI bus controller
U1.9-P1	Slot_12/P1/Z2	SCSI bus controller

Desired:

Drawer

Class Code: Ethernet controller; Vendor ID: 511

Figure D-21 Hardware location codes on the HMC

Additional SDR information

The p690 Clustering requires some new SDR attributes, as well as enhancements to existing ones. This section covers the SDR changes.

Frame Class

Each p690 is managed as a single SP frame. In order for hardmon to be able to contact the CIM server on the HMC, the right hardware protocol needs to be defined in the SDR. The definition of the HMC's IP address and the domain name gives a clear relationship between those two. The SDR Frame Class information is described in Table D-4.

Table D-4 List with new and enhanced attributes in the frame class

Attribute name	Description	p690 values
hardware_protocol	Hardware protocol used by hardmon.	HMC
control_ipaddrs	HMC IP addresses, used by hardmon and hmcd.	Comma-delimited list of HMC IP addresses. List order specifies in which priority the connection will be attempted.
domain_name	Identifies the p690 system. The domain name must be unique per HMC. However, there can be several domain names that are the same in the same cluster, as long as they are on different HMCs.	The p690 system name, as shown on the HMC. The name is entered from the **spframe** command.

Example D-9 shows the **splstdata** output with the domain name of the attached p690 servers.

Example: D-9 splstdata output for domain name information

```
[c119s][/]> splstdata -f
                    List Frame Database Information
frame#  tty         s1_tty      frame_type     hardware_protocol  control_ipaddrs domain_name
------ ------------ ----------- -------------- ------------------ --------------- -----------
    1  /dev/tty0    ""          switch          SP                ""              ""
    2  /dev/tty1    ""          switch          SP                ""              ""
    3  ""           ""          ""              HMC               9.114.76.124    Rather
    4  ""           ""          ""              HMC               9.114.76.124    Jennings
```

Tip: To avoid problems later on, we recommend that you use unique domain names not only per HMC, but also in the IBM @server Cluster 1600 environment.

Node Class

Since PSSP represents each LPAR as a node, the LPAR name is stored in the Node Class of the SDR. The Node Class is described in Table D-5 on page 221.

Table D-5 List with new and enhanced attributes in the Node Class

Attribute name	Description	p690 values
LPAR_name	Logical partition identifier	Retrieved by hardmon. It is the same name as shown on the HMC in the Partition Management menu.

Adapter Class

The new feature to define an SPLAN adapter through its hardware location code requires the new physical_location attribute in the SDR Adapter Class. Since the SPLAN adapter can be different than en0, it needs to define explicitly which adapter belongs to the SPLAN. The Adapter Class is described in Table D-6.

Table D-6 List with new and enhanced attributes in the Adapter Class

Attribute name	Description	p690 values
physical_location	Physical location code for the adapter.	The value as shown by the **spadaptr_loc** or **lscfg -vl** command.
SPLAN	Indicates whether or not this adapter is connected to the SP admin network (SPLAN).	A value of 1 indicates that this is an SPLAN adapter. A value of 0 means that this adapter belongs to a different network. Only one adapter per node can have this value set to 1.

Perspectives

The Perspectives functions and GUI have been enhanced to fully integrate the new attached servers.

Node status

As on the existing SP hardware, Perspectives can display different types of node states on the p690 servers, depending upon whether the p690 is in SMP or LPAR mode.

In SMP mode

If the p690 is in SMP mode, Perspectives can monitor four different modes. Each mode is displayed in a different color, as shown in Table D-7 on page 222.

Table D-7 Possible node states in SMP mode

Mode	Color	Description
No power	Red	No power to the node.
Initializing	Yellow	Power has been applied and it is initializing.
Operational	Green	Node has initialized and is operational.
Error	Red	Hardware is in an error state.

In LPAR mode

If the p690 is in LPAR mode, Perspectives can monitor six different node states as shown in Table D-8.

Table D-8 Node states in LPAR mode

Status	Color	Description
Not ready	Red	The LPAR should not be booted or reset.
Defined	Yellow	The LPAR is defined and can be activated.
Initializing	Yellow	The LPAR has been activated and is booting.
Operational	Green	The LPAR is up and running.
Open Firmware	Yellow	The LPAR is running in SMS or Open Firmware.
Error	Red	LPAR, operating system, or hardware is in error.

Power on/off

In LPAR mode, Perspectives can power on or power off LPARs or the frame. In Full System Partition mode, the p690 is managed like any other attached node. Therefore, Perspectives only powers on or off the single SMP node. This also affects the frame's power status.

The spled window

Like the Perspectives GUI, the spled window has also been enhanced to support p690 Clustering, as shown in Figure D-22 on page 223.

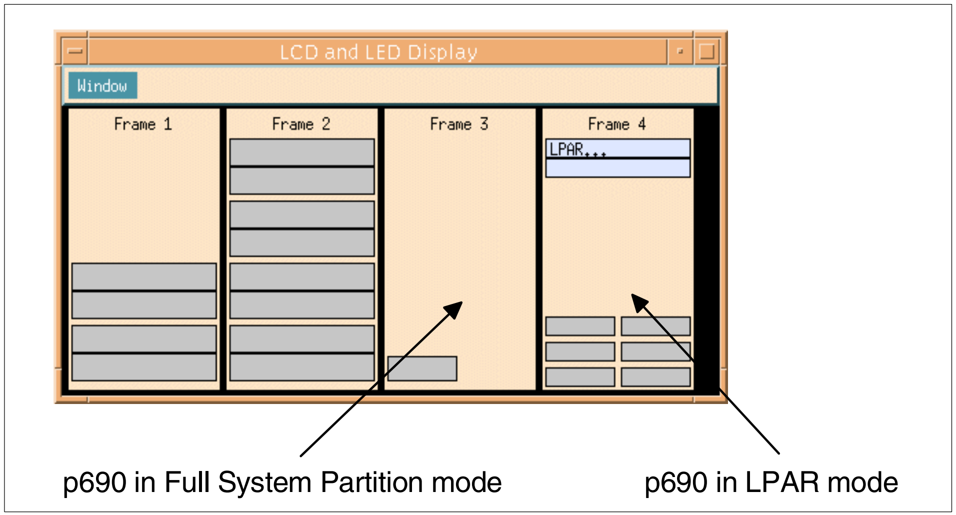


Figure D-22 The spled window with two attached p690 servers

The spled window shows two SP frames with high nodes and two p690 frames. The first p690 (Frame 3) is running in Full System Partition mode and is represented as a single thin node. The second p690 (Frame 4) is running in LPAR mode. Each configured LPAR is displayed as a thin node.

New commands and new command flags

In this section, we describe some relevant new commands and command flags. However, covering all new commands and command flags is beyond the scope of this paper. For more complete and detailed information on these subjects, refer to the following PSSP documentation:

- *PSSP for AIX: Command and Technical Reference*, SA22-7351
- *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641

Attention: We recommend that you run the **`SDRArchive`** command before you change the SDR information.

spframe

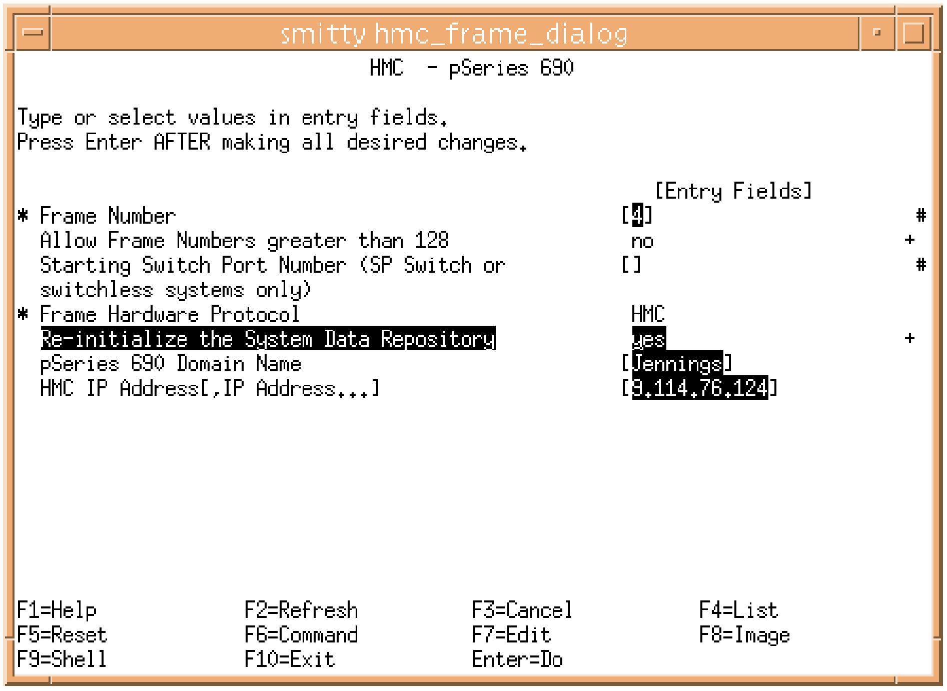
This command has new flags to add a p690 server as a frame, as shown in Example D-10 on page 224 and in Figure D-23 on page 224:

```
usage: spframe -p HMC -d {domain_name} -i {list_of_HMC_IPaddresses} [-n
{starting_switch_port}] [-r {yes|no}] [-o] frame_number
```

Example: D-10 spframe command with 2 HMCs are connected to the p690 server

```
[c119s][/]>spframe -p HMC -d Jennings -i 9.114.76.124,9.114.76.125 -r yes -o 4
```

The SMIT fast path is **smitty hmc_frame_dialog**.



```
smitty hmc_frame_dialog
                         HMC  - pSeries 690

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* Frame Number                                        [4]               #
  Allow Frame Numbers greater than 128                 no               +
  Starting Switch Port Number (SP Switch or           []                #
  switchless systems only)
* Frame Hardware Protocol                              HMC
  Re-initialize the System Data Repository             yes              +
  pSeries 690 Domain Name                             [Jennings]
  HMC IP Address[,IP Address...]                      [9.114.76.124]






F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure D-23 SMIT hmc_frame_dialog screen shot

splstdata

The flag -a now shows the hardware location code from the network card (see Example D-11).

Example: D-11 Partial output of the splstdata -a command

```
[c119s][/]> splstdata -a
                  List LAN Database Information
node# adapt  netaddr         netmask         hostname          type
t/r_rate enet_rate  duplex other_addrs     adapt_cfg_status  physical_location
SPLAN
----- ------ --------------- --------------- ----------------- ------------
-------- ---------- ------ --------------- ----------------- -----------------
-----
33 css0   9.114.76.135   255.255.255.128 c119sn07.ppd.pok. SP_Switch2_P NA
""         ""     ""               css_ready         U1.9-P1-I3/Q1     0
```

```
  34 css0  9.114.76.136   255.255.255.128 c119sn08.ppd.pok. SP_Switch2_P NA
""         ""     ""              css_ready        U1.9-P2-I3/Q1     0
```

sphmcid

This command obtains the HMC user ID and password for the hardmon daemon to establish a secure connection to the HMC, or shows the provided HMC IP address and the HMC user.

After you enter a valid HMC IP address and user ID, the command prompts you for the password, as shown in Example D-12.

```
sphmcid: Usage: sphmcid [-h]|[host_name|ip_address [user_id]]
```

Example: D-12 sphmcid command

```
[c119s][/]> sphmcid 9.114.76.124 hscroot
```

The SMIT fast path is **`smit enter_hmc`**.

spadaptr_loc

This command returns the physical location codes for all adapters installed on an LPAR/node. This command powers down the LPAR to retrieve the needed information. The information gathered is only shown on your screen and will not be saved in the SDR. The usage of this command is shown in Example D-13.

```
Usage: spadaptr_loc [-h]
spadaptr_loc {start_frame start_slot {node_count | rest} |
        -l <node_list> | -N node_group}
```

Example: D-13 spadapt_loc command

```
[c119s][/]> spadaptr_loc 4 1 4
```

The SMIT fast path is **`smit adaptr_loc_dialog`**.

spadaptrs

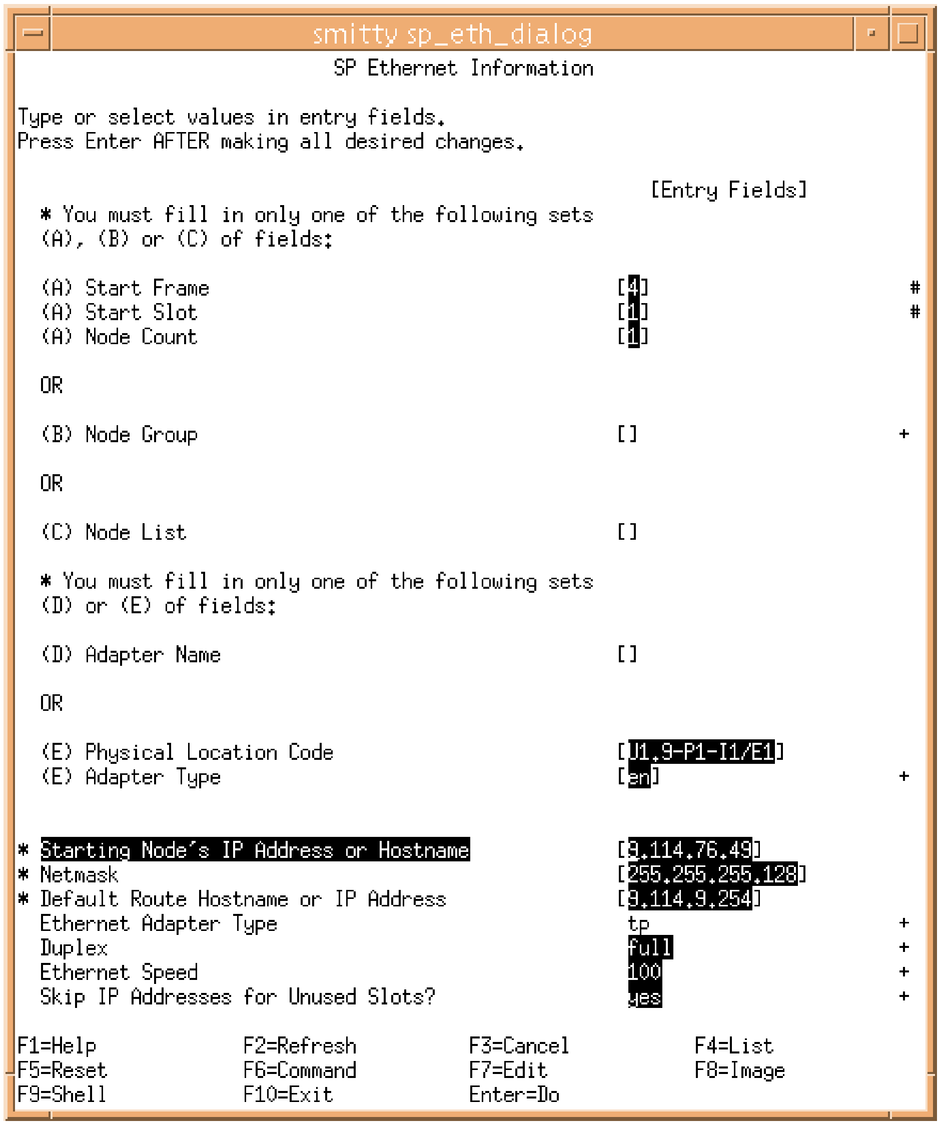
This command was changed for the p690 server to allow the physical location code (-P flag) of the adapter, and the definition of the SPLAN's default route to the CWS (-e flag). If the -P flag is used to give the physical location code of the adapter, the command accepts the adapter type rather than the adapter name (that is, en instead of en0). The -e flag defines this adapter as the SPLAN adapter. If this flag is given, you must do one of the following:

- Provide the adapter physical location code (-P) and adapter type of en
- Provide the adapter name of en0

The usage of this command is shown in Figure D-24 on page 227.

```
Usage:  spadaptrs [-s {yes | no}]
        [-t {bnc | dix | NA | tp | fiber }]
        [-r {4 | 16 | autosense}]
        [ -d {full | half | auto} ]
        [ -f {10 | 100 | auto}]
        [-a {yes | no}] [-n {yes | no}]
        [-o {ip address[,ip address ...]}]
        [-P <physical_location>]
        [-e <default_route>]
        {start_frame start_slot {node_count | rest} |
        -l <node_list> | -N node_group}
        adapter_name starting_ip_address netmask
```

The SMIT fast path is **`smit sp_eth_dialog`**.



```
smitty sp_eth_dialog
                              SP Ethernet Information

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
  * You must fill in only one of the following sets
  (A), (B) or (C) of fields:

  (A) Start Frame                                    [4]                       #
  (A) Start Slot                                     [1]                       #
  (A) Node Count                                     [1]

  OR

  (B) Node Group                                     []                       +

  OR

  (C) Node List                                      []

  * You must fill in only one of the following sets
  (D) or (E) of fields:

  (D) Adapter Name                                   []

  OR

  (E) Physical Location Code                         [U1.9-P1-I1/E1]
  (E) Adapter Type                                   [en]                     +


* Starting Node's IP Address or Hostname             [9.114.76.49]
* Netmask                                            [255.255.255.128]
* Default Route Hostname or IP Address               [9.114.9.254]
  Ethernet Adapter Type                               tp                      +
  Duplex                                              full                    +
  Ethernet Speed                                      100                     +
  Skip IP Addresses for Unused Slots?                 yes                     +

F1=Help             F2=Refresh          F3=Cancel           F4=List
F5=Reset            F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Figure D-24   SMIT sp_eth_dialog screen shot

Tip: We recommend using the SMIT panel for this command.

sphostnam

This command now has a new flag, -P, to provide the physical location code information, as shown in Example D-14 on page 228.

```
[c119s][/]> sphostnam
Usage:  sphostnam [-a adapter_name | -P physical_location_code] [-f {long |
short}]
        {start_frame start_slot {node_count | rest} |
        -l <node_list> | -N node_group}
```

Example: D-14   sphostnam command

```
[c119s][/]> sphostnam -P U1.9-P1-I1/E1 -f long 4 1 1
```

The SMIT fast path is **`smit hostname_dialog`**.

spdeladap

The new -P flag allows you to enter the physical location code for the adapter, as shown in Example D-15.

```
[c119s][/]> spdeladap
Usage:
  spdeladap -h
  spdeladap {start_frame start_slot node_count | -l <node_list> | -N
node_group} adapter_name
  spdeladap -P physical_location_code {start_frame start_slot node_count | -l
<node_list> | -N node_group}
```

Example: D-15   The spdeladap command

```
[c119s][/]> spdeladap -P U1.9-P1-I1/E1 4 2 1
```

The SMIT fast path is **`smit delete_adapter_dialog`**.

spmon

The **`spmon`** command has no new command flags, but there is a different output for the switch responds, as shown in the partial output of the **`spmon -d`** command displayed in Example D-16. You can see that the column Switch Responds is now a table on its own. Earlier, this column was located between the Host responds and the Key Switch column. The change is required to show single/dual plane switch configuration.

Example: D-16   Output of the spmon -d command

```
------------------------------- Frame 3 -----------------------------
                      Host     Key     Env   Front Panel      LCD/LED
Slot Node Type  Power Responds Switch  Error LCD/LED          Flashes
---- ---- ----- ----- -------- ------- ----- ---------------- -------
  1    33  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  2    34  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  3    35  thin   on      yes     N/A    N/A  LCDs are blank     N/A
  4    36  thin   on      yes     N/A    N/A  LCDs are blank     N/A
```

```
          Switch Responds (per plane)
Slot Node 0      1
---- ---- ------ ------
  1    33   yes    yes
  2    34   yes    yes
  3    35   yes    yes
  4    36   yes    yes
```

Configuring p690

In this section, we provide an overview on how to add, delete, and reconfigure the p690 server in a Cluster as a frame and nodes. Each p690 is defined as one frame, and each LPAR is defined as a node.

The comparison tables given in the following sections show you the differences between classic nodes in 9076 frames and the SP-attached p690 server. It is important to understand that all work performed on the HMC is similar to the work the IBM CE performs in a SP environment.

If you want to add a new LPARed node to your Cluster, you have to first assign resources from your p690 server to the LPAR on the HMC, and then do all the steps on your CWS to install a new node. If you want to delete a node, first delete the LPAR definitions using the HMC, then delete the node definitions from the CWS. For more detailed information, see “Reconfiguring the RS/6000 SP System” in the *PSSP for AIX: Installation and Migration Guide*, GA22-7347.

Adding p690 to a Cluster

We explain this task by comparing the steps involved in adding a classic SP frame. Table D-9 lists the steps required to add a classic 9076 SP frame and a p690 as a frame.

Table D-9 Adding a frame

Tasks	9076 Frame	p690	Notes
Connect the frame to the CWS via the RS232 line.	X		
Connect p690 via the RS232 to the HMC.		X	See “Serial connection” on page 197
Check the software release installed on the HMC.		X	See “Software levels” on page 197

Tasks	9076 Frame	p690	Notes
Check the firmware level on p690 server.		X	See “Required p690 firmware” on page 200
Connect HMC to CWS via the trusted LAN/SPLAN.		X	See “Network connections” on page 195 for the CWS side See this “Network connections” on page 195 for the HMC side
Connect all the nodes/LPARs to the SPLAN	X	X	See “Network connections” on page 195 for the CWS side See “Network connections” on page 195 for the HMC side
Run **`smit enter_hmc`** to enter the HMC user ID and password.		X	See Figure D-16 on page 212
Run **`smit sp_frame_dialog`**.	X		
Run **`smit hmc_frame_dialog`**.		X	See “spframe” on page 223 and Figure D-23 on page 224
Check and update the firmware of the supervisor card.	X		
Check the software release installed on the HMC.		X	See “Software levels” on page 197
Check the firmware level on the p690 server.		X	See “Required p690 firmware” on page 200

Adding p690 LPARs to a Cluster

In an IBM @server Cluster 1600, an LPAR is configured as a thin node. Defining an LPAR as a node is somewhat similar to adding a new node in an RS/6000 SP environment. In Table D-10 on page 231, we compare the steps you have to do if you want to add an LPAR as a node versus adding a node in a classic SP frame.

Table D-10 Adding nodes

Tasks	9076 node	p690 node	Notes
Add adapter cards to the p690 (Ethernet/Switch/FC and so on).		X	See "Adapter placement for p690" on page 202.
LPAR definition on HMC.		X	Refer to the *HMC Operations Guide*, SA38-0590.
Physically place a node in the frame.	X		
Power connections.	X		
SPLAN connection.	X	X	See "Connectivity between CWS, HMC, and p690" on page 187.
Switch connection.	X	X	See "Adapter placement for p690" on page 202.
Connect to all other adapters you will use.	X	X	
Run the **`spadaptr_loc`** command to obtain the hardware location code.		X	See "spadaptr_loc" on page 225 and Figure D-20 on page 218 to create a /etc/bootptab.info file for the MAC address of the SPLAN adapter.
Enter the SPLAN information to the SDR.	X	X	See "spadaptrs" on page 225 and Figure D-24 on page 227.
Run **`sphrdwradr`** to obtain the MAC address of the SPLAN adapter and put into the SDR.	X	X	This command reboots the nodes if there is no entry in the /etc/bootptab.info file.
Enter all node data to the SDR in the CWS.	X	X	
Power on/Activate the node/LPAR.	X	X	Refer to the *HMC Operations Guide*, SA38-0590.

Deleting LPARs in a Cluster

Deleting a node in a classic SP frame involves deleting the node configuration information in the SDR and physically removing the node from the SP frame. In Table D-11 on page 232, we compare these steps with an LPARed node.

Table D-11 Deleting a node

Tasks	9076 node	p690 node	Notes
Shut down and power off the node/LPAR.	X	X	
Remove the node from the SP frame.	X		
Delete the LPAR definition from the HMC.		X	Refer to the *HMC Operations Guide*, SA38-0590.
Delete all SDR information from the CWS.	X	X	Use the **`spdelnode`** command.

Adding an Ethernet adapter to an LPAR node

We now compare the steps of adding an Ethernet adapter to a node or to an LPAR. In Table D-12, we compare the tasks for adding an Ethernet adapter between an LPAR node and an SP node.

Table D-12 Adding an Ethernet adapter

Tasks	9076 node	p690 node	Notes
Shut down and power off the node.	X		
Add the adapter to the I/O drawer.	X	X	See “Adapter placement for p690” on page 202.
Assign the I/O slot to the related LPAR.		X	Refer to the *HMC Operations Guide*, SA38-0590.
Enter the SDR information for the adapter.	X	X	See “spadaptrs” on page 225.
Set the customize flag for the node/LPAR.	X	X	Use the **`spbootins`** command.
Shut down the LPAR/node.		X	
Power on/Activate the node/LPAR.	X	X	

Reconfiguring LPARs

In a p690 server, you can reconfigure the LPARs during normal system operations by adding or deleting the resources to an LPAR. Since each LPAR is defined as a node in the SDR, any changes in the LPAR configuration require a change in the SDR. In this section, we discuss a few sample scenarios.

Node numbering

In this scenario, we describe what happens to the node numbers if you delete and add nodes in a p690. The nodes are numbered from 1 to 16, one per LPAR for each p690 system. If there are four LPARs, then the frame will have four thin nodes defined in the SDR.

Rule: The lowest unused node number of a p690 frame will be used if you add an LPAR.

In Figure D-25 on page 234, you see the same frame in three different configurations.

- Configuration 1 shows the SP-attached p690 server with frame number 1. Here we have six LPARs, related to the thin nodes numbered 1, 2, 3, 4, 5, and 6.
- In Configuration 2, we deleted nodes 4 and 5 on the HMC and in the SDR. Node numbers 4 and 5 are not used. At this point the nodes defined in the SDR are 1, 2, 3, and 6.
- In Configuration 3, we added a new LPAR definition on the HMC. The hmcd daemon automatically finds the new LPAR as an node and generates a new entry in the SDR. The lowest unused node number is used for this LPAR node, and node number 4 is assigned to the new thin node.

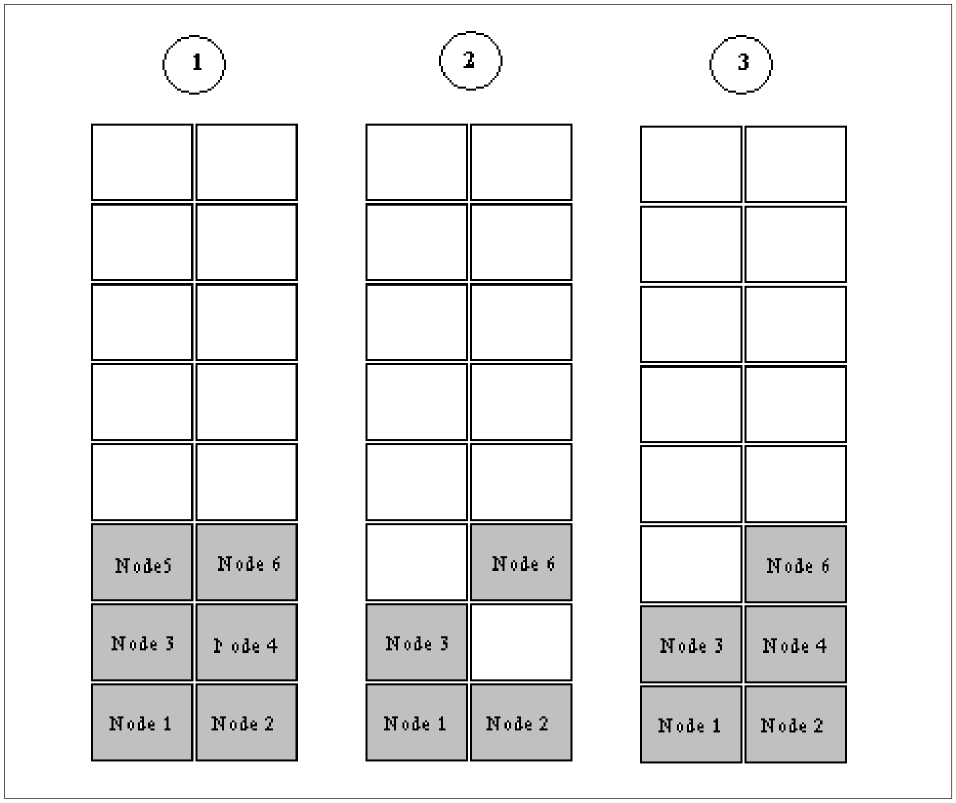


Figure D-25 Node numbers when reconfiguring LPARs

Move or replace the SPLAN adapter

In an LPAR node, you may choose to move the SPLAN adapter to a new location or replace the SPLAN adapter. We discuss these scenarios in the following sections.

Replace the SPLAN adapter

If the SPLAN Ethernet adapter fails and you have to replace it, it is like replacing it on any other SP nodes or attached servers.

1. Use the Hot-Plug feature to replace the adapter on the node.
2. Get the new hardware address (MAC) with the **`lscfg -vl ent0 | grep "Network Address"`** command.
3. Store the MAC address in the /etc/bootptab.info file.
4. Run the **`sphrdwrad`** command to store the MAC address into the SDR.

5. Remove the existing NIM object using the **`delnimclient`** command. The next time **`setup_server`** is run, the NIM object will be created for this node.

Move the SPLAN adapter to a different PCI I/O slot

If you move the SPLAN adapter to a different I/O slot, you have to reinstall the node. After replacing and moving the adapter, the SDR has no valid information about that SPLAN adapter. On the node, the new Ethernet device has no IP configuration. Booting the node in customize mode does not work, since the node has no IP connection to the control workstation.

Using multiple LPAR definitions or profiles

It is possible to use multiple profiles for one LPAR definition. You have to remember that PSSP uses only the default profile. If you reboot or shut down an LPAR node via the command line or the graphical user interface using Perspectives, all operations occur only on the default profile.

To change the default profile, do the following:

1. Shut down the node (for example, with the **`shutdown`** command from the command line on the node).
2. Change the default profile on the HMC.
3. Change the SDR if you changed any PSSP-defined resources and customize the node.
4. Activate the LPAR via the **`spmon`** command, Perspectives, or from the HMC.

The following LPAR resources in a Cluster defined to PSSP need to be uniquely assigned to a *single* LPAR or profile definition:

- SPLAN adapter (Ethernet adapter)
- rootvg devices (hard disks)
- All other devices controlled by the CWS (for example, Token Ring Adapter, other Ethernet Adapter, and so on)
- SP Switch/SP Switch2 adapters

If you want to use multiple LPAR definitions for the same p690 resources that are assigned to more than one LPAR definition (for example, FC, SSA, and so on), make sure that your SDR information is up to date.

Restriction: IBM does not support having PSSP-controlled devices (css, en, tr, or rootvg devices (hdisks), and so on) deferred to more than one LPAR definition.

Switching between LPAR and Full System Partition mode

IBM @server pSeries p690 supports two operation modes: Full System Partition mode and LPAR mode. If you plan to switch back and forth between these modes, Chapter 6, "Reconfiguring the RS/6000 SP system", in the *PSSP for AIX: Installation and Migration Guide*, GA22-7347 discusses one method for accomplishing this. This method involves deleting and recreating the Node and Frame objects in the SDR.

If you switch between LPAR and Full System Partition mode frequently, you may find this process a bit difficult. For example, let us say you use your IBM @server p690 during the day in LPAR mode with four LPARs defined (let us call them LPARs A, B, C, and D), and you use your p690 at night in Full System Partition mode. Using the procedure highlighted above, you then need to redefine your frame and nodes in the SDR twice each day (once each time you switch modes)!

You can avoid this by instead simulating Full System Partition mode while still in LPAR mode. We describe two methods for doing this:

- Create a separate LPAR definition
- Create a separate LPAR profile

Create a separate LPAR definition

You can create a new LPAR definition that simulates the server running in Full System Partition mode. Let us explore this using the preceding example.

During the day, the server runs with LPARs A, B, C, and D active. Now, create a new LPAR that will only be active at night, and call it LPAR E. In this case, you do not have to switch modes when going from day to night and vice versa, but can run your machine as follows:

- During the day, LPARs A, B, C, and D are active. LPAR E is not active.
- During the night, LPAR E is active. LPARs A, B, C, and D are not active.

Since all five LPARs will appear in the SDR at all times (PSSP sees all LPARs defined on the HMC), LPAR E can have all of the resources in the server allocated to it, other than those resources defined in PSSP to LPARs A, B, C, and D. (All of the LPARs defined in PSSP need to have unique SPLAN adapters, rootvg volume groups, and so on, and these resources cannot be shared. All other resources not defined to PSSP, such as memory and processors, can be assigned to LPAR E.) Therefore, LPAR E can be allocated all of the memory and processors in the server, but not all of the hard drives and communications adapters.

The procedure for creating a new LPAR definition to simulate running the server in Full System Partition mode is as follows:

1. Create an additional LPAR and allocate resources to it:
   a. Allocate a new disk for the AIX image (the disk cannot be allocated to any other LPAR).
   b. Allocate an Ethernet adapter for the SPLAN connection (the adapter cannot used by any other LPAR).
   c. Allocate other resources (memory, processors, and so on), which may or may not be allocated by other LPARs.
2. Add SDR information for the new node:
   a. Use a new IP address for the SPLAN adapter.
   b. Use new IP settings for any other adapters defined through PSSP.
3. Power off all the other LPARs that belong to that frame.
4. Install the new node.

The limitation to this method is that you are forced to have more node definitions in the SDR than are currently active.

Note: In this type of configuration, the maximum number of active nodes at one time is limited to 15, since one node number is used by this LPAR.

Create a separate LPAR profile

To avoid losing the use of one of the LPARs, you can instead create an additional profile of an existing LPAR. In the new profile, allocate the same resources used in the other profile, along with the rest of the resources in the server not defined in PSSP to LPARs A, B, C, and D. This procedure is described in the following:

1. Select an LPAR and create a new profile. We suggested selecting the LPAR that corresponds to the first node number in that frame.
2. Allocate resources to that LPAR:
   a. Allocate the same boot disk as on the default profile.
   b. Allocate the same Ethernet and switch adapter as on the default profile.
   c. Via the HMC, allocate all other resources not already defined to other nodes in the SDR.

3. Save the new profile.
4. Set this profile as the default profile.

Note: You can allocate all resources from the current LPAR. In addition, you can also allocate all other resources, except for PSSP-managed resources from other LPARs.

Helpful feature codes

This section lists feature codes you may find useful. For a complete list of supported adapters, check:

`http://www.ibm.com/servers/eserver/pseries`

p690 system

Table D-13 lists the feature codes that are probably the most used when adding a p690 to an IBM @server Cluster 1600.

Table D-13 List of p690-related feature codes

FC	Name and description
2122	SCSI cable from I/O drawer to media drawer. Needed for additional Ultra 3 SCSI adapter.
4962	Ethernet adapter for SPLAN connections. AIX LPP: devices.pci.1410ff01.*
6203	Ultra 3 SCSI adapter for additional devices, plugged into the internal media drawer. AIX LPP: devices.pci.00102100.*
8396	SP Switch adapter (TB3PCI). AIX LPP: ssp.css
8397	SP Switch 2 adapter. AIX LPP: ssp.css

Hardware Management Console

Since you need an HMC for p690 Clustering, Table D-14 shows a list of components for the HMC.

Table D-14 List of useful HMC-related feature codes

FC	Name and description
2943	8-Port Async PCI adapter for connection to the p690.

FC	Name and description
3629	P76/P77 color monitor.
3630	P260/P275 color monitor.
6600	US-English keyboard also includes a three-button mouse.
7316	Hardware Management Console (HMC). This feature code also includes the Red Hat Linux operating system.
8120	RS-232 attachment cable to p690 (6 m).
8121	RS-232 attachment cable to p690 (15 m).

Limitations

For information about current limitations, refer to *Parallel System Support Program for AIX: Read This First for New Users*, GI10-0641.

Example scenarios

In this section, we show a few sample scenarios using p690, HMC, and the CWS.

CWS with two HMCs and four p690s

Figure D-26 on page 240 shows four p690 servers in LPAR mode, with all nodes connected to the SPLAN. To achieve high availability, we use two HMCs. Each HMC is connected to all p690s via RS232 connections. The connection between the CWS and the HMC is via a trusted Ethernet connection.

Tip for this scenario: For better performance when installing the nodes, make sure that when defining the p690 frames (see the `spframe` command), you change the order of the HMC IP addresses.

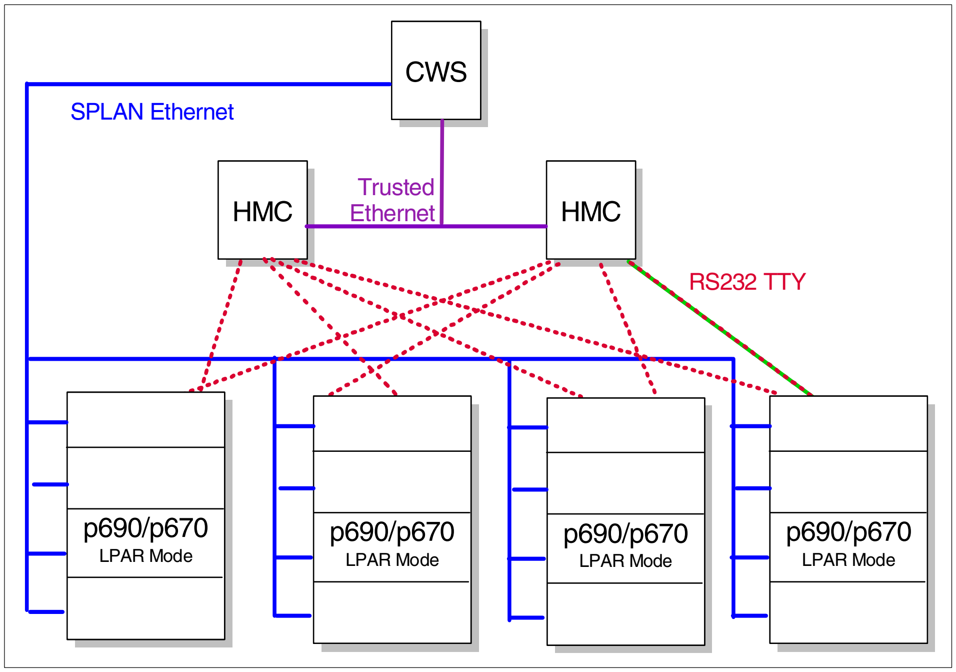


Figure D-26 Four p690s with two redundant HMCs and one CWS

One CWS, one HMC, and one p690

Figure D-27 on page 241 shows the smallest possible Cluster 1600 based on one p690. A useful upgrade for this scenario could be a trusted Ethernet between the HMC and the CWS.

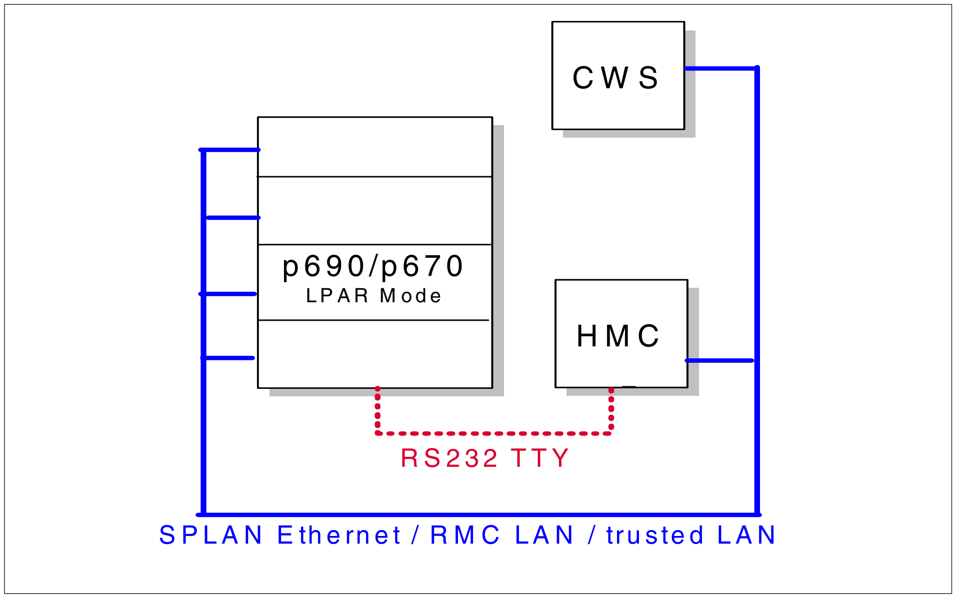


Figure D-27 Cluster 1600 with one HMC, one p690, and CWS

CWS, two HMCs, two 9076s, with one p690 and SP Switch2

Figure D-28 on page 242 shows a Cluster 1600 after integrating a p690 server in LPAR mode. Not all LPARs have a connection to the SP Switch2. This scenario shows a redundant HMC with a separate trusted LAN between the HMC and the CWS.

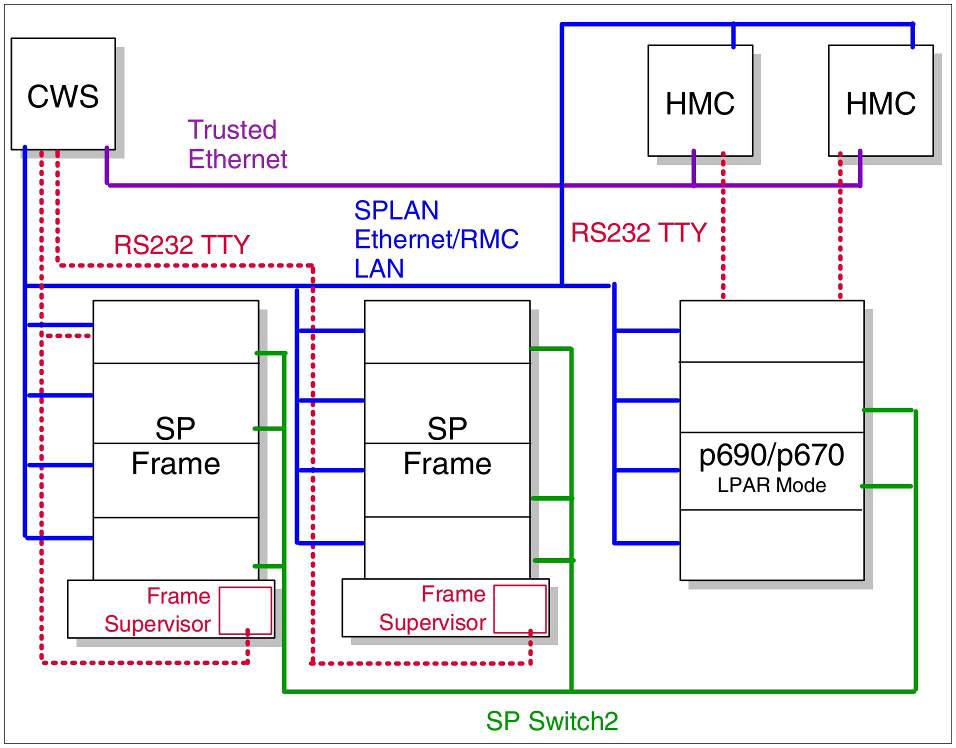


Figure D-28 SP Frames with one p690 and two LPARs on an SP Switch2

CWS, HMC, 9076 frame, and p690 with SP Switch

Figure D-29 on page 243 shows a p690 in LPAR mode where all LPARs have an SP Switch connection. The RMC LAN traffic goes over the SPLAN.

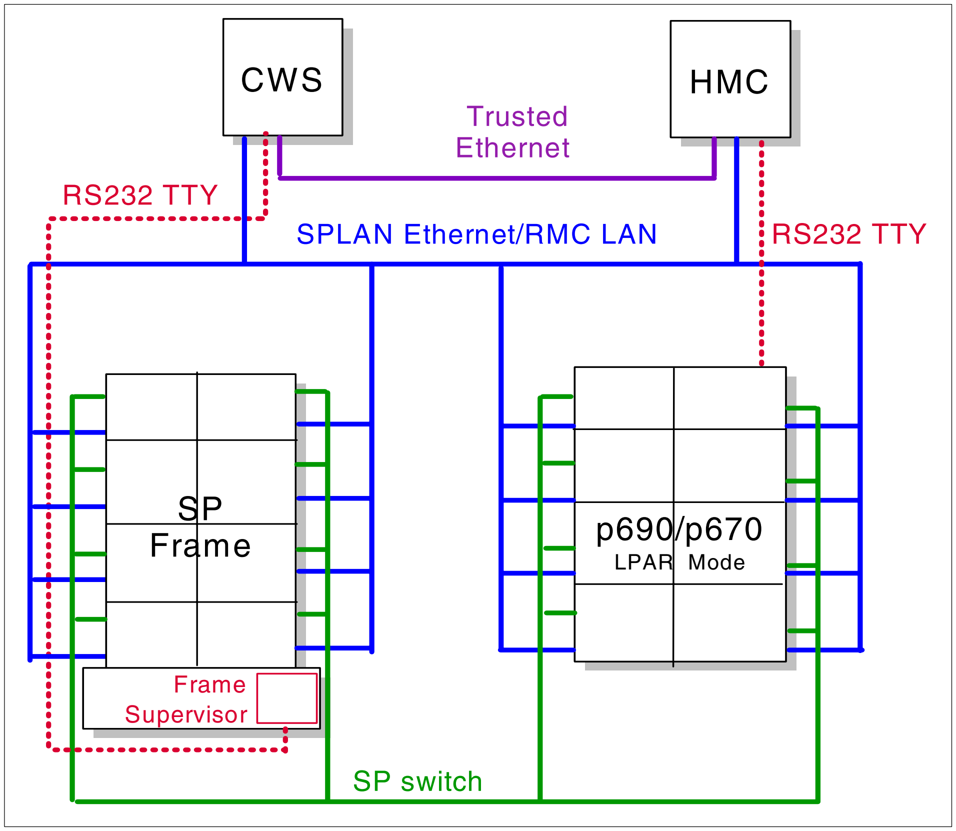


Figure D-29 SP Frame and p690 with SP-Switched LPARs

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 248.

- *Additional AIX Security Tools on IBM @server pSeries, IBM RS/6000, and SP/Cluster*, SG24-5971
- *AIX 5L Differences Guide,* SG24-5765
- *AIX 5L Performance Tools Handbook*, SG24-6039
- *AIX Logical Volume Manager, From A to Z: Introduction and Concepts,* SG24-5432
- *Configuring the IBM VSS for Performance and Availability*, SG24-5279
- *Exploiting RS/6000 SP Security: Keeping It Safe*, SG24-5521
- *IBM @server pSeries 690 System Handbook,* SG24-7040
- *The POWER4 Processor Introduction and Tuning Guide*, SG24-7041
- *RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide,* SG24-5155
- *RS/6000 SP/Cluster: New Enhancements in PSSP 3.4,* SG24-6604
- *RS/6000 SP System Performance Tuning Update,* SG24-5340
- *Scientific Applications in RS/6000 SP Environments,* SG24-5611

Other resources

These publications are also relevant as further information sources:

- *AIX Performance Monitoring and Tuning Guide,* SC23-2365
- *AIX Version 4, Optimization and Tuning Guide for Fortran, C, and C++,* SC09-1705
- *Configuring p690 in an IBM eServer Cluster 1600*, REDP0187
- *ESSL for AIX Version 3 Release 3: Installation Memo,* GI10-0604

- *ESSL Products General Information,* GC23-0529
- *ESSL Version 3 Release 3 for AIX Guide and Reference,* SA22-7272
- *HMC Operations Guide,* SA38-0590
- *IBM @server pSeries 690 Installation Guide,* SA38-0587
- *IBM @server pSeries 690 User’s Guide,* SA38-0588
- *IBM Tivoli Storage Manager for AIX Administrator's Guide Version 5.1,* GC32-0768
- *IBM Tivoli Storage Manager for AIX Administrator's Reference Version 5.1,* GC32-0769
- *LoadLeveler for AIX: Installation Memo,* GI11-2819
- *Loadleveler for AIX: Diagnosis and Messages Guide,* GA22-7882
- *LoadLeveler for AIX: Using and Administering,* SA22-7881
- *Parallel Environment for AIX: Hitchhiker's Guide,* SA22-7424
- *Parallel Environment for AIX: Installation Guide,* GA22-7418
- *Parallel Environment for AIX: MPI Programming Guide,* SA22-7422
- *Parallel Environment for AIX: Operations and Use, Volume 1,* SA22-7425
- *Parallel Environment for AIX: Operation and Use, Volume 2,* SA22-7426
- *Parallel ESSL for AIX Version 2 Release 3 Guide and Reference,* SA22-7273
- *Parallel ESSL Version 2 Release 3 for AIX: Installation Memo,* GI10-0607
- *Parallel System Support Program for AIX: Read This First for New Users,* GI10-0641
- *PSSP for AIX: Command and Technical Reference,* SA22-7351
- *PSSP for AIX: Installation and Migration Guide,* GA22-7347
- *RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment,* GA22-7280
- *RS/6000 SP: Planning Volume 2, Control Workstation and Software Environment,* GA22-7281
- *RS/6000 & pSeries PCI Adapter Placement Reference,* SA38-0538
- *Some Practical Suggestions for Performing Gaussian Benchmarks on a pSeries 690 System,* REDP0424
- *Some Practical Suggestions for Performing NCBI BLAST Benchmarks on a pSeries 690 System,* REDP0437
- *VisualAge C++, Version 5.0 Getting Started* (only comes with the product)
- *XL Fortran for AIX User’s Guide Version 7.1,* SC09-2866

The following references are shipped with the AIX 5L product:

- *AIX 5L Version 5.1 Commands Reference*
- *AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*
- *AIX 5L Version 5.1 System Management Guide: Communications and Networks*
- *AIX 5L Version 5.1 System Management Guide: Operating System and Device*

Referenced Web sites

These Web sites are also relevant as further information sources:

- *RS/6000 SP: SP Switch and SP Switch2 Performance* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch_perf.html
- *The IBM @server pSeries 690 Reliability, Availability, Serviceability (RAS)* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690_ras.html
- *IBM @server POWER4 System Microarchitecture* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html
- *RS/6000 SP: SP Switch2 Technology and Architecture* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/sp_switch2.pdf
- *Partitioning for the IBM @server pSeries 690 System* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/lpar.html
- *IBM @server pSeries 690 Configuring for Performance* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690_config.html
- Power4 focuses on Memory Bandwidth

  http://www.ibm.com/chips/news/1999/microprocessor99.pdf
- *The RS/6000 64-bit Solution* whitepaper

  http://www.ibm.com/servers/eserver/pseries/hardware/whitepapers/64bit6.html

- Information on XL Fortran

  `http://www.ibm.com/software/ad/fortran/xlfortran/`
- Information on VisualAge C/C++

  `http://www.ibm.com/software/ad/vacpp/`
- PE for AIX 5L Version 3 Release 2 Sample Programs

  `http://www.ibm.com/servers/eserver/pseries/library/sp_books/aix5pe_examples.tar`
- Melville, *Moby Dick*

  `http://www.informika.ru/text/books/gutenb/gutind/TEMP/i-_m8.html#mobydick.html`

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Abbreviations and acronyms

ABI	Application Binary Interface
ALPAR	Affinity Logical Partition
APAR	Authorized Program Analysis Report
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BI	Business Intelligence
BLAS	Basic Linear Algebra Subprograms
BNC	Bayonet Niell-Concelman
BOS	Basic Operating System
CD-ROM	Compact Disk - Read Only Memory
CEC	Central Electronics Complex
CIU	Core Interface Unit
COFF	Common Object File Format
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CWS	Control WorkStation
DASD	Direct Access Storage Device
DCA	DC Power Supply
DCE	Distributed Computing Environment
DIMM	Dual Inline Memory Module
DMA	Direct Memory Access
DPCL	Dynamic Probe Class Library
DRAM	Direct Memory Access
DVD	Digital Video Disc
DVD-RAM	Digital Video Disk - Random Access Memory
ECC	Error Checking and Correcting
EIA	Electronics Industries Association
ESCON	Enterprise Systems Connection
ESSL	Engineering and Scientific Subroutine Library
FC	Fibre Channel Feature Code
FDDI	Fiber Distributed Data Interface
FIFO	First in First Out
GB	GigaByte
GPFS	General Parallel File System
HACMP	High Availability Cluster Multi-Processing
HACWS	High Availability Control WorkStation
HiPPI	High Performance Parallel Interface
HMC	Hardware Management Console
HPC	High Performance Computing
HPS	High Performance Switch
IBM	International Business Machines Corporation
IP	Internet Protocol
ITSO	International Technical Support Organization
JFS	Journaled File System
KB	KiloByte
LAPI	Low level Application Programming Interface
LED	Light Emitting Diode
LMB	Logical Memory Block
LPAR	Logical Partition

LRU	Least Recently Used
LVM	Logical Volume Manager
MAC	Medium Access Control
MASS	Mathematical Acceleration Subsystem
MB	MegaByte
MCM	Multiple Chip Module
MMF	Multiple Mode Fiber
MPI	Message Passing Interface
MTU	Maximum Transmission Unit
NCAR	National Center for Atmospheric Research
NCU	Non-Cacheable Unit
netCDF	Network Common Data Form
NIM	Network Installation Management
NLS	National Language Support
NUMA	Non-Uniform Memory Access
PBLAS	Parallel Basic Linear Algebra Subprograms
PCI	Peripheral Component Interconnect
PCM	Parallel Climate Model
PCT	Performance Collection Tool
PE	Parallel Environment
PESSL	Parallel Engineering and Scientific Subroutine Library
PHB	PCI Host Bus
PMU	Performance Monitoring Unit
POE	Parallel Operation Environment
POWER	Performance Optimization With Enhanced RISC
PSSP	Parallel System Support Program
PVT	Profile Visualization Tool
RAID	Redundant Array of Independent Disks
RAS	Reliability, Availability, Serviceability
RIO	Remote I/O Bus
RISC	Reduced Instruction Set Computer
RMC	Resource Monitoring and Control
RPM	Revolutions Per Minute Red Hat Package Manager
RSCT	RISC System Cluster Technology
ScaLAPACK	Scalable Linear Algebra Package
SCSI	Small Computer System Interface
SDR	System Data Repository
SMI	Synchronous Memory Interface
SMIT	System Management Interface Tool
SMP	Symmetric Multiprocessing
SP	Scalable POWER Parallel
SPCN	System Power Control Network
SPOT	Shared Product Object Tree
SSA	Serial Storage Architecture
SSL	Secure Socket Layer
STI	Self-Timed Interface
TB	TeraByte
TCP	Transmission Control Protocol
TLB	Translation Lookaside Buffer
TOD	Time Of Date
TSM	Tivoli Storage Manager
US	User Space
USB	Universal Serial Bus
UTE	Unified Trace Environment
UTP	Unshielded Twisted Pair
VMM	Virtual Memory Manager

VPD	Vital Product Data
VSD	Virtual Shared Disk
WebSM	Web-based System Management
XCOFF	Extended Common Object File Format

Index

IBM

Redbooks

Performance and Tuning Considerations for the p690 in a Cluster 1600

Performance and Tuning Considerations for the p690 in a Cluster 1600

Look under the hood of the pSeries 690 and the SP Switch2 interconnect

Marvel at the new features that are relevant to performance

Watch as the authors play "what if" games

This redbook is based on the firm belief that all good tuning begins and ends with a fundamental understanding of how your system operates within your own environment. There are three main parts to the book. The first two are conceptual. The coverage in the first part includes pSeries 690 internals, such as POWER4, MCMs, memory subsystem, I/O subsystem, and network connectivity (including SP Switch2, SP Switch2 PCI Attachment Adapter, and EtherChannel). Coverage in the second part includes details about performance relevant features, such as affinity logical partitions (ALPARs), scheduling (processor) and memory affinity, technical large page support, and IP vs. US protocol over the new interconnect fabric. The third part details a series of investigations used to see how the new features work and interact. The coverage there includes technical large page, scheduling (processor) and memory affinity, LPAR and ALPAR, Tivoli Storage Manager (TSM), AIX mkramdisk, and IP vs. US. In addition, we have included *Configuring p690 in an IBM @server Cluster 1600*, REDP0187 as an additional appendix, which will tell you how to configure a pSeries 690 in a Cluster 1600.

SG24-6841-00 ISBN 0738426393