New features

The following is a list of the major new features found in Version 1.0 Release 3.0 (V1R3) of Blue Gene/L.

Control system

Browser-based administration console

This new console consolidates most administration tasks into a single user interface. Since it is browser based, it is easily accessible and usable. Some of the tasks that can be performed from the new console include:

- Block management
  - Blocks (partitions) can be created, deleted, booted and released. Their status can be tracked.
- Job management
  - Jobs can be terminated and status can be tracked.
- Query and viewing of RAS events
  - Improved capability to search through RAS events, and pinpoint areas with higher error rates.
- Diagnostics
  - Diagnostics can be configured to run only on a specific subset of hardware, and a time estimate is provided for how long the operation will take.
- Access to log files
  - A new Blue Gene/L-specific log viewing facility is provided that makes it much easier to view, query and compare events from the various log files on the system.
- Analysis of system utilization
  - This feature allows you view the utilization of several areas of interest over a defined period of time. Examples include: number of submitted jobs, software RAS events, and overall system utilization.
- Other miscellaneous new functionality

This new interface also provides a mechanism from which MMCS commands can be invoked.

The web server (Tomcat) providing this service operates on the service node. Logging into the new administration console requires a username and password on the Service Node, and that username must also be part of the bgl or bgladmin group.

New MMCS functionality

Several new commands have been added to the Midplane Management Control System (MMCS) application.

- list_bps
  This command shows the status of all base partitions configured.
- locaterank
  This command allows the user to map an MPI rank to a physical node.
- max_psets_per_bp
  This command allows the administrator to configure a new property that tells MMCS not to attempt to use all IO nodes on the machine. This can be useful in several instances, including uncabled nodes or a temporary shortage of switch capacity.

Bridge API changes

Several new APIs are provided to allow for more control over job control and submission, and for the different memory sizes now allowed. For more detailed information, see *Blue Gene/L: Application Development*, SG24-7179.

- RM_BPComputeNodeMemory
  Returns a pointer to an enum value indicating the compute node memory size for the base partition.
- RM_JobStartTime
  Returns a pointer to a string containing the job start time with format of yyyy-mm-dd-hh.mm.ss.nnnnnn. If the job never went to running state it will be an empty string. Data is only valid for completed jobs. The rm_get_data spec RM_JobInHist can be used to determine if a job has completed or not. If the job is an active job then the value returned is meaningless.
- RM_JobEndTime
  Returns a pointer to a string containing the job end time with format of yyyy-mm-dd-hh.mm.ss.nnnnnn. Data is only valid for completed jobs. The rm_get_data spec RM_JobInHist can be used to determine if a job has completed or not. If the job is an active job then the value returned is meaningless.
- RM_JobRunTime
  Returns a pointer to the job run time in seconds. Data is only valid for completed jobs. The rm_get_data spec RM_JobInHist can be used to determine if a job has completed or not. If the job is an active job then the value returned is meaningless.
- RM_JobComputeNodesUsed
  Returns a pointer to the number of compute nodes used by the job. Data is only valid for completed jobs. The rm_get_data spec RM_JobInHist can be used to determine if a job has completed or not. If the job is an active job then the value returned is meaningless.
- RM_JobExitStatus
  Returns a pointer to the job exit status. Data is only valid for completed

jobs. The rm_get_data spec RM_JobInHist can be used to determine if a job has completed or not. If the job is an active job then the value returned is meaningless.

  - RM_JobInFile
    This API was deprecated.

Mixed-memory support

This new support allows a machine containing some racks with 512MB memory per compute node and other racks with 1GB memory per compute node to be part of the same BG/L machine. MMCS will no longer enforce uniformity across the entire machine; rather, MMCS will enforce uniformity at the midplane level.

Compute Node Kernel

For V1R3, the Compute Node Kernel (CNK) has added the following new features:

- Support for allocating memory regions with specific L1 cache attributes
- Support for allocating storage from SRAM
- Support for application assisted L1 data cache parity error recovery
- Support for the execve system call
- File and socket I/O performance enhancements

Support for allocating memory regions with specific L1 data cache attributes

The Blue Gene/L memory subsystem supports the following L1 data cache attributes:

- Cache-inhibited or cached. Memory with the cache-inhibited attribute causes all load and store operations to access the data from lower levels of the memory subsystem. Memory with the cached attribute may use the data cache for load and store operations. The default attribute for application memory is cached.
- Store without allocate (SWOA) or store with allocate (SWA). Memory with the store without allocate attribute bypasses the L1 data cache on a cache miss for a store operation and the data is stored directly to lower levels of the memory subsystem. Memory with the store with allocate attribute allocates a line in the L1 data cache when there is a cache miss for a store operation on the memory. The default attribute for application memory is store with allocate.
- Write-through or write-back. Memory with the write-through attribute is written through to the lower levels of the memory subsystem for store operations. If the memory also exists in the L1 data cache, it is written to the data cache and the cache line is marked as clean. Memory with the write-back attribute is written to the L1 data cache and the cache line is marked as dirty. The default attribute for application memory is write-back.

Starting with V1R2, CNK supports setting L1 data cache attributes for all of application memory. The user can choose store without allocate mode by setting the BGL_APP_L1_SWOA environment variable to a non-zero value and write-through mode by setting the BGL_APP_L1_WRITE_THROUGH environment variable to a non-zero value.

In V1R3, CNK supports allocating only a region of memory with a particular L1 data cache attribute with the rts_get_dram_window() system call. To make memory available for the system call, the application must be linked with a larger starting address. More information is in Chapter 3 of *Blue Gene/L: Application Development*.

Application developers must understand the way their code interacts with memory and cache very well when using rts_get_dram_window(), as misuse could actually degrade performance. One example of where this feature can improve performance is during matrix multiplication operations where one matrix is used repeatedly and another is calculated, written out to disk (or to another node), and then discarded. The latter matrix might best occupy a region of memory that has the SWOA attribute, since using valuable L1 data cache for this transitory data could push other data (such as the repeatedly-used matrix data) out of the L1 data cache.

Support for allocating storage from SRAM

SRAM is a low- latency memory that is accessible by both PPC440 processors on a compute node. There is no caching performed on SRAM, and coherence is maintained between both processors. It can be used as a very fast communications buffer to pass data between the processors. There is 8KB of SRAM available to applications and 4KB of it is used by the MPI libraries. CNK now supports two new system calls:

- rts_allocate_sram() to allocate storage from SRAM.
- rts_free_sram() to free storage from SRAM.

More information is in Chapter 3 of *Blue Gene/L: Application Development*.

Application assisted L1 data cache parity error recovery

For Blue Gene/L hardware most data in the memory subsystem is ECC protected which allows for hardware recovery of single bit errors along with detecting all double bit errors. However, the L1 instruction cache, L1 data cache, and Translation Lookaside Buffers (TLB) are only parity protected. A parity error is detected by the hardware and can sometimes be handled by software. While only a relatively small amount of the memory subsystem is exposed to parity errors, the large number of processors in bigger Blue Gene/L installations greatly magnifies the probability and impact of a single processor taking a parity error.

CNK has always handled L1 instruction cache parity errors by invalidating the instruction cache and allowing the instructions to be reloaded from lower levels of the memory subsystem.

When running in write-through mode, L1 data cache is always clean and CNK handles all parity errors by invalidating the L1 data cache and allowing the data to be reloaded from lower levels of the memory subsystem. But there is additional overhead in using write-through mode which impacts application performance.

In V1R3, CNK has support for handling L1 data cache parity errors when running in write-back mode. If the parity error occurs on a clean cache line, CNK flushes any dirty cache lines from the L1 data cache, invalidates the data cache, and allows the data to be reloaded from lower levels of the memory subsystem.

If the parity error occurs on a dirty cache line, CNK flushes any other dirty cache lines in the L1 data cache, invalidates the data cache, and transfers control to an application supplied handler. The handler is passed the effective address of the memory with the parity error and the address of the instruction that was running when the parity error occurred. If both the memory address and instruction address are in a part of the application (and not in a library linked to the application), the application can recalculate the data and resume. If the data cannot be recalculated, the application must exit and terminate the job.

The user must explicitly choose to use write-back parity error recovery by setting the environment variable BGL_APP_L1_WRITEBACK_RECOVERY to a non-zero value. If the environment variable is not set, a L1 data cache parity error in write-back mode causes CNK to terminate the job. There is additional overhead added by the processor's load pipeline to enable proper recovery of parity errors which impacts application performance. The overhead is less than running in write-through mode.

Users are encouraged to benchmark their applications with the various memory configurations to find the right balance between performance and the ability to recover from L1 data cache parity errors. More information is in Chapter 3 of *Blue Gene/L: Application Development*.

Support for the execve system call

In V1R3, CNK supports the execve system call which allows the user to run a different program on the compute node. The execve system call has the following limitations:

- The control system does not know about the new program that is running after calling the execve system call. So the job table and job history table contain the name of the initial program that was submitted to the partition. Any interfaces

that use the program name field from the two job tables return the name of the submitted program.

- If a debugger is currently attached to the job, the execve system call returns an error and sets errno to ENOTSUP.
- The execve system call can only be used once per job. After the first call to execve, any subsequent calls return an error and set errno to ENOTSUP.
- There is no support for changing the effective user id and effective group id based on the SetUserID and SetGroupID mode bits. CNK does not support this on job submission either.
- There is no support for the close-on-exec flag for file descriptors so all open file descriptors remain open after the execve system call.
- There is a limit of 32 KB for the size of the arguments and environment variables passed to the new program.
- The new program must use the same origin address as the submitted program. If the submitted program obtains storage using the rts_get_dram_window() system call, the storage is still available and mapped in the same way after the execve system call.
- The submitted program must not use any MPI functions. The MPI library does not allow MPI_Init() to be called more than once per job.
- Extreme caution must be used if the application uses checkpoint/restart APIs. A checkpoint uses a barrier to sync all of the nodes in a partition and quiesce the networks. With separate programs running it will be highly variable as to when each program gets to the barrier and the checkpoint can start.

The execve system call can be used by an application to implement Multiple Program Multiple Data (MPMD) support. The application can provide a “launcher” program that is submitted and which in turn uses either the execl(), execlp(), execle(), execv(), execvp(), or execve()function to run other programs based on the MPI rank of the node.

File and socket I/O performance enhancements

The performance of the read, readv, recv, and recvfrom system calls is improved by eliminating a memory copy of the received data. In the previous releases, CNK received the data into a temporary kernel buffer and then copied the data into the user’s buffer. CNK now receives the data directly into the user’s buffer. The improvements are most noticeable when a small number of compute nodes in the pset are receiving data.

In addition, the default read/write buffer size was increased from 87600 bytes to 262144 bytes (256KB). The larger buffer size improves performance for I/O operations. The buffer size can be configured by setting the CIOD environment variable CIOD_RDWR_BUFFER_SIZE.

I/O Node

For V1R3, the I/O Node has added the following new features:

- Running an exit program on the I/O node when a job starts
- Support for additional CIOD environment variables
- Changed the MCP version from 4.0 to 4.1

Running an exit program on the I/O node when a program starts

CIOD can now call an exit program supplied by the administrator each time a new job starts. For example, the exit program can change tuning parameters for the file system client on a per job basis or mount a specific file system based on the user. More information is in Chapter 6 of *Blue Gene/L: System Administration*, SG24-7178.

Support for additional CIOD environment variables

CIOD added support for the following environment variables:

- CIOD_DEBUGGER_DISCONNECT_TIMEOUT
- CIOD_NEW_JOB_EXIT_PGM
- CIOD_NEW_JOB_EXIT_PGM_TIMEOUT
- CIOD_REPLY_MSG_SIZE
- CIOD_SOCKET_BUFFER_SIZE

More information is in Chapter 6 of *Blue Gene/L: System Administration*, SG24-7178.

Upgraded the MCP version from 4.0 to 4.1

The version of MCP used on the I/O node was upgraded from version 4.0 to version 4.1. The new version of MCP uses the same Linux 2.6.5 kernel but includes various security-related and other minor fixes.

Communications

Note: One-sided communication support (ARMCI and global arrays) will be made available at a later date. They are not supported on GA of V1R3. Please contact your technical advocate to find out when these functions are officially made available.

One-sided communication through ARMCI

From the design documentation created by the Pacific Northwest National Laboratory (PNNL), ARMCI's creator:

> "The purpose of the Aggregate Remote Memory Copy (ARMCI) library is to provide a general-purpose, efficient, and widely portable remote memory access (RMA) operations (one-sided communication) optimized for contiguous and

> noncontiguous (strided, scatter/gather, I/O vector) data transfers. In addition, ARMCI includes a set of atomic and mutual exclusion operations."

The main goal of ARMCI is to allow for optimized communications while maintaining compatibility with MPI. On Blue Gene/L, it serves as the underpinnings for the Global Array support described below.

Further information, such as programming APIs for ARMCI, can be found at:
http://www.emsl.pnl.gov/docs/parsoft/armci/index.html.
In addition, the actual code that provides ARMCI support on Blue Gene/L will not be made available from IBM itself, but rather from the PNNL website. Users will need to download the source code from PNNL, build it (using the makefiles provided), and install it on their Service Node such that it is available to applications and it has access to the Blue Gene/L control code.

Global Arrays

The implementation for global arrays was built on top of the ARMCI support described earlier. Again, a quote from the PNNL website:

> "The Global Arrays (GA) toolkit provides an efficient and portable 'shared-memory' programming interface for distributed-memory computers. Each process in a MIMD parallel program can asynchronously access logical blocks of physically distributed dense multi-dimensional arrays, without need for explicit cooperation by other processes. Unlike other shared-memory environments, the GA model exposes to the programmer the non-uniform memory access (NUMA) characteristics of the high performance computers and acknowledges that access to a remote portion of the shared data is slower than to the local portion. The locality information for the shared data is available, and a direct access to the local portions of shared data is provided.
>
> Global Arrays have been designed to complement rather than substitute for the message-passing programming model. The programmer is free to use both the shared-memory and message-passing paradigms in the same program, and to take advantage of existing message-passing software libraries. Global Arrays are compatible with the Message Passing Interface (MPI)."

Further information on global arrays can be found at http://www.emsl.pnl.gov/docs/global/. As with the ARMCI support, the actual source code must be downloaded from PNNL and built as described above.

New Cartesian communicator functions to map nodes to psets

There are three new APIs that make it easier to map nodes to specific hardware and/or pset configurations. We will go over each individually.

- PMI_Cart_comm_create()
  This function creates a 4-dimensional Cartesian communicator that mimics the exact hardware on which it is run. The X, Y & Z dimensions match those of the partition hardware, while the T dimension will have cardinality 1 in coprocessor mode, 2 in virtual node mode. The communicator wrap-around links match the true mesh/torus nature of the partition. In addition, the coordinates of a node in the communicator exactly match its coordinates in the partition.

  It is important to understand that this is a collective operation and must be run on all nodes. The function may be unable to complete successfully for a number of different reasons (mostly likely when it is run on fewer nodes than the entire partition). It is important to ensure that the return code is MPI_SUCCESS before continuing to use the returned communicator.

- PMI_Pset_same_comm_create()
  This is a collective operation that creates a set of communicators (each node seeing only one), wherein all nodes in a given communicator are part of the same pset (in other words, they all share the same I/O node).

  The most common use for this would be to coordinate access to the outside world to maximize the number of I/O nodes. For example, node 0 in each of the communicators can be arbitrarily used as the "master node" for the communicator, collecting information from the other nodes for writing to disk.

- PMI_Pset_diff_comm_create()
  This is a collective operation that creates a set of communicators (each node seeing only one), wherein no two nodes in a given communicator are part of the same pset (they all have different I/O nodes). The most common use for this would be to coordinate access to the outside world to maximize the number of I/O nodes. For example, an application that has an extremely high bandwidth per node requirement could run both PMI_Pset_same_comm_create() and PMI_Pset_diff_comm_create().

  Nodes without rank 0 in PMI_Pset_same_comm_create() could just sleep, leaving those with rank 0 independent and parallel access to the functional Ethernet. Those nodes would all belong to the same communicator from PMI_Pset_diff_comm_create(), allowing them to use that communicator instead of MPI_COMM_WORLD for group communication/coordination.

MPI coprocessor sends (environment variable BGLMPI_COPRO_SENDS)

Prior to V1R3, when running in coprocessor mode, responsibility for only message receives was handled by the communications coprocessor; sends were still managed by the core running the user application. By turning the environment variable BGLMPI_COPRO_SENDS on (to a value of "1"), the communications subsystem is instructed to hand off responsibility for both the receiving and sending of messages to the communications coprocessor. Depending on the computational and communications mix of the application in question, enabling this function may result in improved overall performance.

Note that this new functionality only applies when the application is being executed in coprocessor mode. If the application is executing in virtual node mode, the setting of environment variable BGLMPI_COPRO_SENDS is ignored.

It is very difficult to provide concrete guidelines that specify when turning this environment variable on will result in improved performance. Testing is underway to determine whether such guidelines can be defined – as of right now, your best option is to try running your application in both modes and see which performs better.

Interrupt-driven communications (environment variable BGLMPI_INTERRUPT)

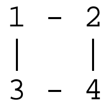
To enable one-sided communication (described earlier in the sections on ARMCI and global arrays), interrupt-driven communications support was added. Turning the BGLMPI_INTERRUPT environment variable on will allow the MPI communications subsystem to decide to use interrupts if it determines that may be a faster mechanism, even when executing MPI-only applications. A possible scenario where this may occur is if there are a "chain" of nodes where the first sends a non-blocking message to the second (using isend), does some computational work, and then does a wait. The second receives the message from the first node using irecv, does an isend to a third node, performs some computational work, and then waits. This then continues for many nodes. Without interrupts we would likely see a "stair step" pattern form. With interrupts, it is possible that much more of this activity could be done concurrently.

Note that this setting does not disable one-sided communications mechanisms such as ARMCI and global arrays. These functions require interrupt-driven communication, and interrupts will be used even if this environment variable is configured to disable the support.

Bandwidth improvements for planar and diagonal torus communications

Enhancements have been made to allow more efficient use of available bandwidth when sending messages between nodes which are connected in a planar or diagonal manner.

Let's say the given application is sending a relatively large message from node 1 to node 4 in the following two-dimensional diagram:



Previously, all packets were sent only along one path (all either via node 2, or via node 3). Now, the packets making up the message can be sent along both routes, potentially doubling bandwidth. In a three-dimensional (diagonal) example, which would commonly occur in a torus network, packets can be sent along three different routes, potentially tripling bandwidth.

Note that this is primarily important for point-to-point messages; large-scale collectives (such as all-to-all) will likely benefit little from this enhancement, as there will be higher contention for channels. With this enhancements, however, developers may want to consider arranging sends of larger point-to-point messages such that the nodes involved are physically connected in a planar or diagonal fashion.

It important to understand that this discussion is strictly tied to bandwidth. When considering any such optimizations, one must obviously take latency into consideration. Any gains made in improved bandwidth can easily be wiped out, or even made worse, if the nodes involved are a great distance apart.

Application development

Numerous changes and enhancements have occurred in the area of application development. Many of these are driven by the fact that, prior to V1R3, the toolchain was based on SLES8.

XL compiler updates

There are significant new XL compiler features available as of March 17, 2006, via a PRPQ.

The compilers have moved to a new level. We now have:

- XL C/C++ Advanced Edition V8.0 for Blue Gene
- XL Fortran Advanced Edition V10.1 for Blue Gene

Some of the improvements include:

- Perform subset of loop transformations at –O3 optimization level.
- Improved performance of quad precision floating point.
- Improved support for auto-simdization.

For more information see the following URLs:

- http://www-306.ibm.com/software/awdtools/xlcpp/features/bg/xlcpp-bg.html

- o http://www-306.ibm.com/software/awdtools/fortran/xlfortran/features/bg/xlf-bg.html

In addition, a new white paper has been published called "Exploiting the Dual Floating Point Units in Blue Gene/L." It is available at: http://www-1.ibm.com/support/docview.wss?uid=swg27007511

Now that the BG XL compilers are their own PRPQ, they are no longer part of the Linux XL compilers. Only the Linux XL compilers were available through the "Scholars Program," so if you obtained your V7/V9.1 BG XL compilers from this program you will need to purchase the V8/V10.1 licenses. For those customers who already purchased V7/V9.1 BG XL compilers, it will not be necessary to purchase a new license. You are no longer required to have the Linux compiler, but both the Blue Gene/L and Linux compilers can be installed at the same time. You can tell the difference between the two by the "bg" in the path name and in the rpm names (e.g., `xlf.bg.lib-10.1.0-0`, and `opt/ibmcmp/xlf/bg/10.1/bin/blrts_xlf`).

Important: When using the IBM XL Fortran Advanced Edition V10.1 for Blue Gene, customers need to use ESSL 4.2.3 and not ESSL 4.2.2. If an attempt is made to install a wrong mix of ESSL and XLF, the rpm install will fail with a dependency error message.

Updated toolchain

The toolchain has been updated. As part of this change, several version updates have occurred:

- o gcc: from 3.2 to 3.4.3. The new version contains significant improvements in compiler optimizations.
- o binutils: from 2.13 to 2.16.1
- o glibc: from 2.2.5 to 2.3.6

One of the implications of this upgrade to the toolchain is that **all user libraries an application links to must be recompiled if the application is recompiled**.

Note: if you see error: "1486: undefined reference to `__ctype_b' ," then you have not rebuilt all libraries your application links with. Rebuild those libraries and then retry linking your application.

Another implication of this upgrade is that the runtime now detects heap corruption, such as those caused by double frees or passing invalid pointers to the free procedure. Your application will be ended if failures such as these are detected. You can turn this strict behavior off by adding `--env export MALLOC_CHECK_=0` to your mpirun command. In addition, syntax checking is significantly stricter: applications that used to compile may fail because of bad syntax that is no longer allowed.

Since the toolchain was upgraded you will also need to upgrade your Blue Gene XL compilers. Here are the new required Blue Gene XL compilers: the XL Fortran V10.1.0.1 Advanced Edition for Blue Gene and XL C/C++ V8.0.0.1 Advanced Edition for Blue Gene compiler PTFs. Another consideration is that, since the toolchain had not changed from V1R1M0 until now, you had not been required to retrieve the GNU source upon installation of every driver (i.e., you could use the same GNU source to build the toolchain in every driver). This will no longer work - the V1R3M0 installation ReadMe file will contain instructions on retrieving the new GNU source prior to building the toolchain.

Performance tooling enhancements

The External Performance Instrumentation Facility (known as "Perfmon") has been significantly enhanced in V1R3. The major new functions include:

- Provides for twenty-two different groups of performance counters that can be monitored. Previously, only one group of counters could be collected.
- Provides a derived floating point operation (flop) counter. The derived flop counter provides an all-inclusive counter for floating point activity and is key to understanding what floating point percentage of peak an application is achieving. The default group of counters has been changed to use this new derived flop counter.
- Supports the collection of different performance counters for a given application, without having to recompile the application between runs.
- Provides a summary sample type, which only stores a summary of the performance counter data as collected from the monitored jobs.
- Supports more than one instance of the monitor to be run at the same time, each using a different set of collection parameters.
- Supports two data stores, one being a specified directory within the external file system and, optionally, an SQL data base.
- Provides a GUI interface that allows for the performance data to be viewed in real time, or at a later time, using the data stored in the external file system. The GUI interface also provides for some statistical analysis of the collected data.
- Provides functions that allow for the performance data to be extracted into comma separated value (CSV) files, and imported into and exported from the SQL data base. The extract facility allows for simple projection and selection of the data and can work in conjunction with the GUI display interface.

Other enhancements

Coreprocessor debug tool

A new debugging tool is available that enables parallel debug of problems at all levels: hardware, kernel and application. The coreprocessor tool uses the low-level hardware JTAG interface to read and organize various hardware information, including instruction address registers (IAR), general purpose registers (GPR), special purpose registers (SPR), and device control registers (DCR). It was originally developed to process compute node

core files, but has evolved into a parallel debugger. The debugger levies no dependencies on the application code running on the node (no special calls that need to be made, libraries to link in, etc.). It can sort nodes based on their stack traceback and kernel status, which can help isolate a failing/problem node quickly. It also supports stack dumping on a per node basis.

The debugger scales to 64K nodes (coprocessor mode) or 128 K nodes (virtual node mode), and has been used extensively during bringup of large multi-node systems.

Miscellaneous enhancements

The following miscellaneous enhancements have also been made:

- Lightweight diagnostics.
- RAS improvements for Double Hummer problem detection.
- Numerous miscellaneous bug fixes incorporated.