Gemini Software Development Using Simulation

Kevin Peterson, *XT Software Director, Cray Inc.*

ABSTRACT: *This paper describes Cray's pre-hardware development environment, the activities, and the testing for the Gemini software stack. Gemini is the next generation high speed network for the Cray XT-series The simulation environment is based on AMD's SimNow™ coupled with a Gemini device model that can be aggregated to form multi-node systems. Both operating system and programming environment software components have been developed within this environment. The simulated batch environment, regression test suite, and development progress are also described.*

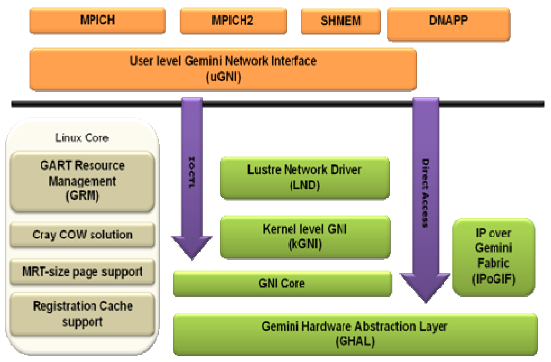
KEYWORDS: XT5, Gemini, Lustre, SimNow™

1. Introduction

Cray's next generation high-speed network (HSN) is based on the Gemini chip. Gemini will replace SeaStar in XT Series supercomputers in the 2010 timeframe. These new XT Series supercomputers based on Gemini are referred to as Baker systems.

From a software perspective, Gemini requires a change from the SeaStar Portals interface to two new application programming interfaces (APIs): the user-level Gemini Network Interface (uGNI) and the Distributed Memory Application interface (DMAPP). The uGNI API was designed for efficient message passing and DMAPP was designed for Partitioned Global Address Space (PGAS) languages, Unified Parallel C, and Co-Array Fortran. Underneath the two programming models, Cray developed a Linux Gemini network interface (GNI) driver as well as Linux IP and Lustre drivers tailored to Gemini. The Gemini software is shown in Figure 1.

Figure 1-Gemini Software Stack


MPICH
MPICH2
SHMEM
DMAPP
User level Gemini Network Interface (uGNI)
Linux Core
GART Resource Management (GRM)
Cray COW solution
MRT-size page support
Registration Cache support
IOCTL
Direct Access
Lustre Network Driver (LND)
Kernel level GNI (kGNI)
GNI Core
IP over Gemini Fabric (IPoGIF)
Gemini Hardware Abstraction Layer (GHAL)

Due to the long lead time of the Gemini chip, the software team had to build an execution environment to accelerate this software development. To that end, Cray developed a system-level simulation environment. This simulation environment was critical to the early development of key Gemini operating system features to be delivered in the Cray Linux Environment (CLE) Danube release. The simulation environment also aided corresponding changes to the programming model libraries and PGAS compiler to be included in the Cray Programming Environment (CPE) Diamond release.

This document describes the functional simulator, the Baker "batch" system, and Cray's test suite used for Gemini software development. The primary components in the simulation environment are AMD's SimNow™ and the Cray Gemini device model. The primary components in Cray's test environment include the Regression Test System (RTS) and the Gemini test suite. The software that is tested includes CLE, CPE, Lustre, and various applications and test software.

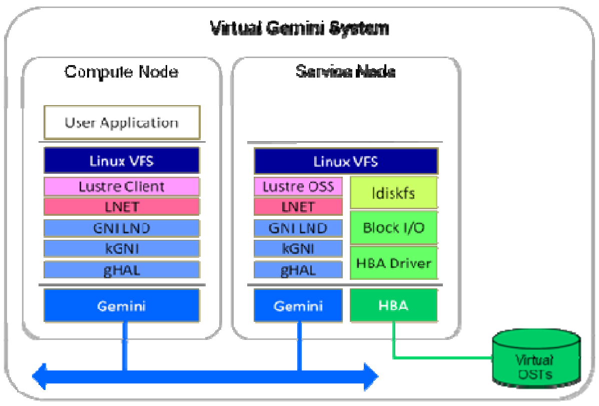
2. Simulation Goals

There were several reasons for building a Baker system-level simulation environment. First and foremost, this simulator served as an early software development vehicle prior to real Gemini hardware. The software team was able to build software in parallel with the Gemini chip and riser card development, thus minimizing the impact on the overall Baker program schedule. Most of the Gemini software stack was developed and debugged on the Baker simulator. Specifically, the User-level

Gemini Network Interface (uGNI), IP over Gemini (IPoG) driver, and Gemini Hardware Abstraction Layer (gHAL) were all developed and stabilized using the simulation environment. These primary software building blocks provided a foundation for subsequent component development within the simulation framework.

The simulation environment was later extended to develop and debug the Gemini Lustre network driver (LND) on top of uGNI. This "GNI" LND was initially ported from the Cray XD1 system and then debugged in the Gemini simulation environment, providing additional exposure to the kGNI and gHAL interfaces. In this simulation environment, both client and server Lustre Gemini stacks (shown in blue in Figure 2) were debugged simultaneously.

Figure 2-Gemini Lustre Simulation Environment



Debugging early, through simulation, ensured that Gemini software was available for operating system boot on initial prototype hardware, and that many system integration issues had been resolved prior to the hardware being available. Adding Lustre to the "ready" software stack allowed additional behavioural coverage of kGNI and gHAL, which complemented the earlier MPI and TCP/IP testing of uGNI and IPoG interfaces. This advanced development work guaranteed that once a real Gemini system was available, the development team could subject the new hardware to both application and I/O traffic patterns.

3. Baker System Simulation

The Baker System simulation environment consisted of multiple nodes. Each node comprised a simulated processor, memory, and Gemini network device. Some nodes were configured with I/O as shown in Figure 3. The processor, memory, and I/O simulator components were provided in AMD's SimNow™ tool. Cray developed a SimNow™ compliant Gemini device model to integrate on each node. Multiple compute and service nodes were aggregated to form a simulated Baker system.

3.1 SimNow™ Environment

AMD's SimNow™ instruction set simulator provides Cray engineers an early platform development environment using AMD64 processors. The following is a general description of SimNow™ from the AMD website:

The SimNow™ simulator is a fast and configurable x86 and AMD64 dynamically-translating instruction-level platform simulator. With SimNow™ users can connect complex software models to form a PC platform emulation environment. SimNow™ emulates AMD Athlon™ 64 and AMD Opteron™ uniprocessor and multiprocessor based systems that run several commercial operating systems and applications. Specifically, AMD and its partners use SimNow™ for:

- *BIOS and device driver development.*
- *Prototyping software visible architectural changes.*
- *Non-intrusive and deterministic measurement and testing of software at the instruction-level.*
- *Modeling of future platform tradeoffs for correctness and performance analysis.*

The simulator contains all the classic pieces of a PC system (CPU, memory, Northbridge, Southbridge, display, IDE drives, floppy, keyboard, and mouse support). Images (hard disk, DVD/CD-ROM, and floppy) can be created in custom sizes with the DiskTool program that is provided with the simulator. A simulation can be saved at any point in the simulation to a media file, from which the simulation can be re-run at a later time.[1]

Cray uses the AMD SimNow™ Binary Release for Linux-64 (revision 4.4.4) and hosts the tool on commodity AMD64 servers. Cray also takes advantage of the virtualized I/O provided in the default configurations to substitute for boot devices and other standard devices, such as a serial console.

3.2 Cray Simulation Hosts

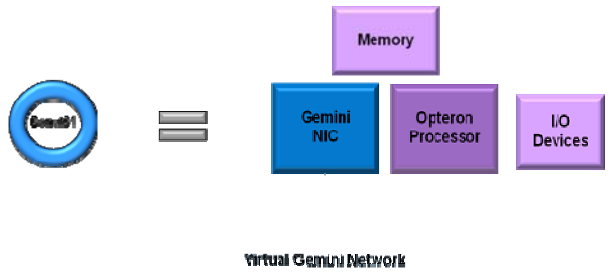
Cray has six NewIsys 1U servers with dual-core Opteron processors with 16GB memory on each. The server node names are *satin01-satin06*, as shown in Figure 4. Likewise, Cray has five Dell 4U servers with quad-core Opteron processors each with 32GB memory. These node names are *velvet01-velvet05*. Cray runs a standard Linux distribution (OpenSuSE) on all of these nodes. All these nodes are interconnected via 10-Gigabit Ethernet.

This loosely coupled cluster of Satin and Velvet nodes comprises a total of 32 cores: 12 cores on Satin nodes and 20 cores on Velvet nodes. For the best performance during simulation, a single instance of SimNow™ is launched per core. Although it is possible to run multiple SimNow™ instances on a core, overall simulation performance degrades almost linearly as the real processor cores are oversubscribed with virtual cores. Hence, the largest practical system simulation with these resources is 32 nodes. Typically, most simulations were run with fewer nodes.

3.3 Gemini Device Model

Cray developed a SimNow™ compliant Gemini device model that integrates with an instance of the simulator. This "device" combined with SimNow™ processor, memory, and I/O components form a Gemini node, as shown in Figure 3.

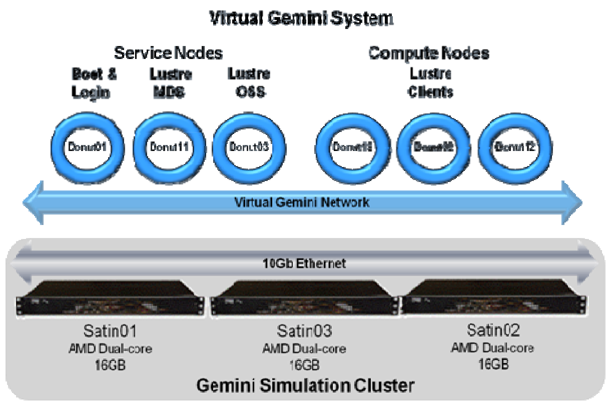
Figure 3-A Gemini Node or *Donut*



For Gemini simulation, one real Opteron core running an instance of SimNow™ is used to host a single Gemini node. Each Gemini node will boot a Cray Linux Environment (CLE) image, either for a compute node (CNL) or a service node. These images are built from the common Cray XT source repository for Baker platforms. Each image has the Gemini software components built-in, which are loaded at image boot.

In the Gemini simulation environment these simulated Gemini nodes are referred to as *donut* nodes, and are assigned node names, **donut01-donutxx**. These *donuts* are assigned as either compute nodes or service nodes, as shown in Figure 4. Cray has run simulations with as many as sixteen donut nodes communicating over the simulated Gemini network. The Gemini network is implemented by the Gemini device model via registration to a SimNow™ service, which uses the Ethernet between host servers as a message transport and proxy for a high speed network.

Figure 4-Lustre Simulation Environment



4. Lustre Simulation Environment

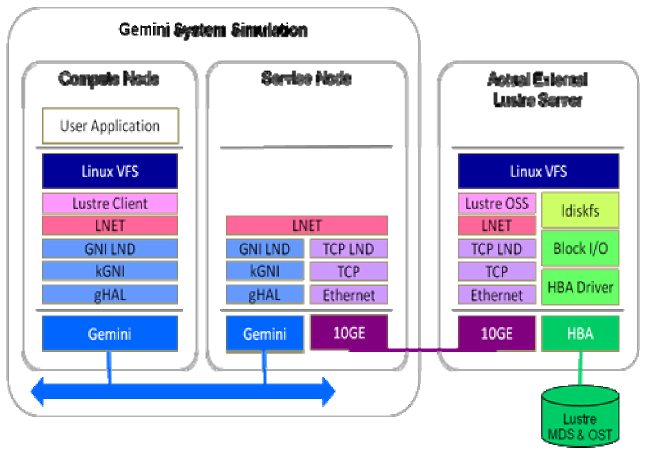
In order to simulate Lustre operation in a Baker system environment, Cray developed a multi-node simulation comprising both service and compute nodes as shown in Figure 4. Service nodes ran CLE service node Linux and compute nodes ran CNL. One service node was configured as a boot node and was used to configure two other service nodes as a Lustre MDS node and single OSS nodes. Additional nodes were configured as compute nodes that ran the Lustre client and mounted the Lustre file system on the service nodes. Applications were launched on compute nodes that performed I/O to the Lustre file system over the simulated Gemini network.

The Cray test group manages a virtual "batch system" consisting of group of donut nodes in a similar configuration to that shown in Figure 4. This batch system is a fully functional Cray XT operating system environment with both service and compute nodes. The service nodes include Lustre, boot/SDB, and even login nodes. Developers can simply logon to this simulated Baker system and run programs as if they were logged on to an XT system running CLE and CPE. This is particularly useful for the test group and programming environment developers, as this batch system has proven to be relatively stable and requires no special knowledge of the simulation environment.

One of the drawbacks of this full environment is that it is relatively slow in execution. This is acceptable for batch operation, but a bit cumbersome for interactive debug. As a result, some developers simply reserved several donut nodes and debugged in a standalone environment, which provided more control and better performance. This was also a safer method for operating system development, as it is common to re-boot or even crash nodes during normal debugging and this induced instability would not have been tolerable in a shared batch environment.

In order to further improve the simulation experience for Lustre development and debug, another Lustre file system configuration was created using external Lustre (eLustre) servers as shown in Figure 5. For Gemini Lustre debug, this configuration was very effective and involved replacing multiple Lustre server nodes with a single node configured as a Lustre router. In this setup the Lustre router can actually route out of the simulation environment to real external Lustre servers.

Figure 5-Gemini Lustre Routing to eLustre



This significantly improves the overall simulation execution speed while at the same time allowing for debug of the Lustre router software. This mode also places a slightly different load on the simulated Gemini software components.

5. Gemini Software Testing

Gemini testing was done at the component level by developers and at the system level by Cray's operating system test group. Both developers and testers have used the simulation environment for their development, debug, and testing of the Gemini software. Developers used a wide variety of tools and techniques to debug their software. The test group used a common set of tools across projects. The following is a summary of the test infrastructure that leveraged the simulated Baker systems. The same infrastructure will be used with real Gemini based systems as they become available.

5.1 Regression Test System (RTS)

Cray's Regression Test System (RTS) provides a mechanism for running tests and analyzing the results. RTS is used for running the Gemini test suite on the Baker simulated systems. The suite consists of over seventy tests that focus on the uGNI interface, and additional tests that target IPoG and DMAPP interfaces. Beside targeted API tests, the test suite also runs both MPI and DMAPP applications. For testing Lustre, generic I/O exercisers, targeted I/O functional tests, and a few MPI applications were run that perform I/O to a Lustre file system. Additionally, a Lustre scratch file system was built and mounted in the simulated batch user environment, which allowed users to launch applications stored on that Lustre file system.

Table 1-Bugs Found in Simulation

Bug	Description	SW/HW
741977	Pallas hang with uGNI MPICH	Gemini Driver
743697	GNI_MemRegisterSegment behaves differently using 1 segment than with > 1 segment	Gemini Driver
743755	Kernel crash using GNI_MemRegisterSegments	Gemini Driver
743977	Short message send does not update mbox credits correctly for certain configurations	Gemini Driver
743983	Smsg mbox credits are updated incorrectly in certain configurations	Gemini Driver
744014	Smsg buffer credits are lost(?) in certain sequences of transactions	Gemini Driver
744500	Memory registration num_pages improperly calculated (?) with libhugetlbfs	Gemini Driver
745897	FMA window allocations can exhaust memory	Gemini Driver
747647	NULL qc_handle parameter to GNI_CqCreate dies ungracefully	uGNI
747689	Test mem_reg_validate fails on hardware with data miscompare	Test issue
747707	Upper limit on the number of CQs is 959, 960'th gets "No space left on device" error	uGNI
747772	dmapp_get of byte-arrays fails with DMAPP_RC_ALIGNMENT _ERROR when target declared on the stack	DMAPP

During this testing, a variety of software bugs were discovered in the Gemini software stack. The majority of these bugs are listed in Table 1. In addition, a large number of additional unreported bugs were identified and fixed by developers in debug and unit testing within the simulation environment prior to software integration testing. Performing this software bug identification and resolution in simulation keeps this activity off the project critical path and will allow the focus of Gemini bring-up to be on the hardware.

6. Lustre Debug under Simulation

The most significant Lustre related development for Baker was the GNI LND. The GNI LND was a port of the Cray XD1 LND to Gemini. Although the preliminary port of the driver was done on the XD1, with kGNI running on the native HAL, the final debug and integration of Cray XT series Lustre features was done under the Gemini simulation environment. Both the internal and external Lustre service modes for fabric attached and remote Lustre servers were exploited to debug and test the GNI LND.

6.1 Gemini Lustre Debug

During the development, debug, and testing of Lustre on the Gemini stack, a number of bugs (listed in Table 2) were found in the Gemini software components. Again, finding these bugs early in simulation prior to Gemini hardware helped to avoid spending time debugging software issues while on the critical path with prototype hardware.

Table 2-Lustre Bugs Found in Simulation

Bug	Description	SW/HW
745553	Lustre IO hangs	kGNI
747905	oops in {:kgni:kgni_error_processing+127}	kGNI
747176	gni_ep_bind:kgni_reqlist_alloc failed (-12)	kGNI
747175	NULL pointer 0x40 @ kgnilnd: kgnilnd_active_conn_handshake+1065	GNI LND
747559	kgnilnd_check_fma_rx()) ASSERTION (conn->gnc_rxmsg == NULL)	GNI LND
747698	RIP: <ffffffff880ae0fd> {:kgnilnd:_kgnilnd_debug_msg+109}	GNI LND

In addition to explicit Lustre I/O testing, the simulated batch environment also mounted Lustre as scratch I/O for users and as a home for test applications and log files.

Two examples of Gemini development are provided at the end of this paper. These annotated examples are of a Lustre debug session (Appendix A) and a comprehensive I/O test suite run (Appendix B). The debug session example provides additional insight as to the level of developer interaction possible in the Gemini simulation environment by showing process output from two nodes. The test run example illustrates the scope of system test supported within the simulation environment and provides some sample application output.

7. Gemini Simulation Summary

The Baker Simulated Batch environment and dedicated simulated Gemini nodes proved to be critical to Gemini software development. This environment was used by software developers and testers for all aspects of Gemini development, from the Gemini Hardware Abstraction Layer (gHAL) all the way up the stack to Cray's PGAS compiler.

This strategy was validated when the first Gemini chips arrived. Once the chips were powered up and initialized, the operating systems booted on the first day. Shortly thereafter multiple nodes were connected via real Gemini chips and applications were launched and run. As more and more Gemini paths have been validated, the Cray software stack has run on that hardware and served as a stable platform for Gemini hardware debug.

Functional simulation played a significant role in the development of Gemini software as well as on prior systems such as the Cray X1/X1E. Cray plans to employ this same strategy for future projects such as the Aries chip, the second generation of the Gemini high speed network.

8. Acknowledgments

The author would like to thank the Cray system simulation team who built-up the Gemini simulation infrastructure and made it a turnkey environment for developers. Specifically, thanks to Bryan Hardy who single-handedly built and maintained the system level SimNow "donut" infrastructure.

Additionally, thanks to the test team who built-up a very comprehensive test infrastructure for Gemini, allowing the software development team to test their software well in advance of the hardware. And a special thanks to Jason Godfrey for pulling together the Gemini test logs and screen shots for this paper and other related presentations, in addition to his day job of building the Gemini test infrastructure and writing tests.

Of course, the entire Gemini software team deserves a lot of credit for their patience as the simulator evolved and more importantly for delivering the software well in advance of hardware milestones, Gemini ready!

9. About the Author

Kevin Peterson has been with Cray for five years and is currently the Software Director for the Cray XT Series supercomputers. He joined Cray as the Software Technical Project Leader for the Cascade Program. Prior to Cray, he was a software professional for over two decades at several other computer companies, including Hewlett Packard, Compaq, Digital Equipment, and Texas Instruments. He can be reached at Cray, Inc. 1340 Mendota Heights Road, Mendota Heights, MN 55120 or by E-mail at kpeterso@cray.com.

10. References

The following websites, documents, and presentations were referenced for this report.

1. *AMD x86-64 SimNow™ website: http://developer.amd.com/cpu/simnow/Pages/default.aspx*
2. *Gemini Lustre I/O Configurations,*
   John Carrier
   Cray Inc.
3. *Cray Gemini Test environment OSTEST Page: http://insidecray.mw.cray.com/~tests/*
4. *RTS Page:*
   http://insidecray.mw.cray.com/~tests/rts/
5. *Gemini test log files:*
   a. Gemini_Lustre_Sim_nid00055_lctl.txt
   b. Gemini_Lustre_Sim_nid00056_lctl.txt
   c. Gemini_Lustre_Sim_LST_Write_1m_1.txt
   d. Gemini_Lustre_Sim_IO_Test_Run.txt
   e. Gemini_Lustre_Sim_IO_Test_Run_Gromacs.txt

Appendix A: GNI LND Debug Session with LNET Self-Test

The following output represents snippets of log files from a Lustre client and server running the full Gemini software stack (gHAL, kGNI, GNI LND, and LNET). The LNET Self-Test, LST is a generic utility for testing LNET services on different platforms. In this example, LST will be used to initiate an LNET function to test the Gemini Lustre Network Driver (GNI LND) and underlying kernel Gemini Network Interface (kGNI) and Gemini Hardware Abstraction Layer (gHAL) on both the client and server.

In this example, LST is used to generate a 1MB message write from one node (nid00055) to another node (nid00056) using an LNET RPC. For reference, the nodes have the following NID assignments.

- nid00055 has the LNet NID 10.128.1.108@gni (a donut node)
- nid00056 has the LNet NID 10.128.1.109@gni (another donut node)

Besides these excerpts, there are also three supplemental files that are relevant to this example.

1. Gemini_Lustre_Sim_nid00055_lctl.txt is a Lustre internal trace file for node 1
2. Gemini_Lustre_Sim_nid00056_lctl.txt is a Lustre internal trace file for node 2
3. Gemini_Lustre_Sim_LST_Write_1m_1.txt is the LST output file

The two Lustre internal trace logs had the following settings:

- sysctl -w lnet.debug=-1
- sysctl -w lnet.subsystem_debug=-1
- sysctl -w lnet.debug_mb=40

The two trace files contain client and server logs for the LNET reader and writer. These files are rather lengthy and difficult to visually synchronize, so key client/server transactions were extracted and provided below. Set-up portions of the LST output file are also included.

Adding nodes:

```
10.128.1.109@gni are added to session
Group [ servers ]
        12345-10.128.1.109@gni: Active
Total 1 nodes [ servers ]
10.128.1.108@gni are added to session
Group [ clients ]
        12345-10.128.1.108@gni: Active
Total 1 nodes [ clients ]
```

Test creation:

```
lst add_test --batch write_test --concurrency
1 --loop 1 --from clients --to servers brw
write check=simple size=1m
```

This is the test sequence showing Lustre transactions over Gemini from alternate Lustre trace log files. The full code flow is in the logs sorted by time (4th field). Here are the main RPCs that result in the 1MB write.

nid00055 calls LNetPut, which generates FMA to 10.128.1.109@gni:

```
00000400:00000200:0:1235132804.358898:0:14968:
0:(lib-move.c:2202:LNetPut()) LNetPut ->
12345-10.128.1.108@gni
00000800:00000200:0:1235132804.359529:0:14962:
0:(gnilnd_cb.c:1494:kgnilnd_sendmsg()) $$
ffff81002d86cc00 sending FMA ffff81000159bd10
02 id 200 [ffff81001cb8a110 for 168]
msg@ffff81000159bd10 m/v/ck/pl
0be91b94/3/1b034d40/168
x30:GNILND_MSG_IMMEDIATE from
10.128.1.108@gni(1235132248969285)
00000800:00000200:0:1235132804.359602:0:14962:
0:(gnilnd_cb.c:1374:kgnilnd_check_fma_send_cq(
)) SMSG Completed 200
```

gets the LNetPut, turns it into a LNetGet and sets up the PUT_REQ:

```
00000800:00000200:0:1235258772.077889:0:20392:
0:(gnilnd_cb.c:1750:kgnilnd_check_fma_rx()) $$
RX on ffff810035cedc00 from 10.128.1.108@gni
msg@ffffc200c6403640 m/v/ck/pl
0be91b94/3/1b034d40/168
x30:GNILND_MSG_IMMEDIATE from
10.128.1.108@gni(1235132248969285)
00000400:00000200:0:1235258772.078301:0:20399:
0:(lib-move.c:2379:LNetGet()) LNetGet ->
12345-10.128.1.108@gni
00000800:00000200:0:1235258772.078562:0:20392:
0:(gnilnd_cb.c:1494:kgnilnd_sendmsg()) $$
ffff810035cedc00 sending FMA ffff810035d75910
07 id 246 [0000000000000000 for 0]
msg@ffff810035d75910 m/v/ck/pl
0be91b94/3/a5cf361f/0 x31:GNILND_MSG_GET_REQ
from 10.128.1.109@gni(1235258178730530)
00000800:00000200:0:1235258772.088135:0:20392:
0:(gnilnd_cb.c:1374:kgnilnd_check_fma_send_cq(
)) SMSG Completed 246
```

nid00055 sees the PUT_REQ and pushes out the RDMA:

```
00000800:00000200:0:1235132804.361943:0:14962:
0:(gnilnd_cb.c:1750:kgnilnd_check_fma_rx()) $$
RX on ffff81002d86cc00 from 10.128.1.109@gni
msg@ffffc200c64036d0 m/v/ck/pl
0be91b94/3/a5cf361f/0 x31:GNILND_MSG_GET_REQ
from 10.128.1.109@gni(1235258178730530)
00000800:00000200:0:1235132804.361981:0:14962:
0:(gnilnd_cb.c:816:kgnilnd_recv()) $$ conn
ffff81002d86cc00, rxmsg ffffc200c64036d0,
lntmsg ffff81001c1c2a00 niov=256
kiov=ffff81001cf76078 iov=0000000000000000
offset=0 mlen=1048576 rlen=1048576
msg@ffffc200c64036d0 m/v/ck/pl
0be91b94/3/a5cf361f/0 x31:GNILND_MSG_GET_REQ
from 10.128.1.109@gni(1235258178730530)
00000800:00000200:0:1235132804.361991:0:14962:
0:(gnilnd_cb.c:218:kgnilnd_setup_phys_buffer()
) niov 256 offset 0 nob 1048576
00000800:00000200:0:1235132804.362030:0:14962:
0:(gnilnd_cb.c:583:kgnilnd_rdma()) Post RDMA
(type = 0x09) tx = 0xffff810001534400,
dlvr_mode 0x0
```

nid0055 sees the RMDA complete and sends the GET_DONE:

```
00000800:00000200:0:1235132804.443792:0:14962:
0:(gnilnd_cb.c:1310:kgnilnd_check_rdma_cq())
RDMA completion event for tx
0xffff810001534400 type 0x09
00000800:00000200:0:1235132804.443815:0:14962:
0:(gnilnd_cb.c:1494:kgnilnd_sendmsg()) $$
ffff81002d86cc00 sending FMA ffff810001534510
09 id 199 [0000000000000000 for 0]
msg@ffff810001534510 m/v/ck/pl
0be91b94/3/634fa92d/0 x31:GNILND_MSG_GET_DONE
from 10.128.1.108@gni(1235132248969285)
```

nid00056 gets the GET_DONE:

```
00000800:00000200:0:1235258772.276743:0:20392:
0:(gnilnd_cb.c:1750:kgnilnd_check_fma_rx()) $$
RX on ffff810035cedc00 from 10.128.1.108@gni
msg@ffffc200c6403778 m/v/ck/pl
0be91b94/3/634fa92d/0 x31:GNILND_MSG_GET_DONE
from 10.128.1.108@gni(1235132248969285)
```

Appendix B: Gemini Lustre Test Session

The following sections of annotated text are snippets of Gemini_Lustre_Sim_IO_Test_Run.txt, the Gemini Lustre test log file. This log file contains a Lustre file system mount and a number of file system exercises. The beginning of the file displays the start-up of the Cray RTS test manager.

```
<<<invocation>>>
rts_driver -A suite=IO -O OS=CNL -O ARCH=XT3 -
n 4
<<<invocation_end>>>
<<<driver_info_start>>>
rts_driver: Info: TMPDIR and CWD are
/lus/scratch/godfrey/tmp
rts_driver: Info: CURRENT_DRIVER_PID = 23148
<<<driver_info_end>>>
<<<rts_keyword_start>>>
Information used for qualification and PATH
adjustment
.
.
.
<<<rts_keyword_end>>>
```

The next section is the system execution environment setup of the test directory paths and the test launch command, `aptrun`.

```
<<<system_info_start>>>
INFORMATION ONLY: Stderr from…
/ostest/dev.gemini/baselinux/ostest/ROOT.lates
t/bin…
/rts_config -O OS=CNL -O ARCH=XT3:
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=
Start time: 09:06:30

----------------------------------------------
---------------
ENVIRONMENTAL INFORMATION:
----------------------------------------------
---------------
CURRENT WORKING DIRECTORY =
/lus/scratch/godfrey/tmp
RTS =
/ostest/dev.gemini/baselinux/ostest/ROOT.lates
t
. . .
LTPROOT =
/ostest/dev.gemini/baselinux/ltp/ROOT.latest
```

```
APPS =
/ostest/dev.gemini/xtcnl/apps/ROOT.latest
. . .
----------------------------------------------
--------------
APTRUN ENVIRONMENTAL INFORMATION:
----------------------------------------------
--------------
R_APP_RUN_NPES =
. . .
R_ARCH_TYPE = XT-alps
. . .
----------------------------------------------
--------------
UBRUN ENVIRONMENTAL INFORMATION:
----------------------------------------------
--------------
UB_CONFEQFAIL =
. . .
TESTDIR = /lus/scratch/godfrey/tmp
```

The next section is information from the test build (make).

```
-rw-r--r-- 1 vers uftest 2026 2008-12-18
10:00…
/ostest/dev.gemini/baselinux/ostest/ROOT.lates
t/make.info
----- make.info Begin -----
Wed Dec 17 19:19:48 CST 2008
Linux baffin 2.6.5-7.244-smp #1 SMP Mon Dec 12
18:32:25
UTC 2005 x86_64 x86_64 x86_64 GNU/Linux
. . .
######### target.mh info #####################
. . .
# pmake directives generated by target_info on
baffin
#     DATE:     Wed Dec 17 19:18:09 CST 2008
#     CMDLINE:
/ptmp/craytest/xt3.compile/ostest/xt3
linux.src/tools/target_info -t xt3-linux
TARGET_OS = XT3
. . .
#
# End of generated pmake directives
#
----- make.info End -----
```

The following is a series of file system mount commands, beginning with local file systems, followed by NFS file systems, and lastly the target Lustre file system mount.

```
----------------------------------------------
--------------
-rw-r--r-- 1 root root 0 2008-04-30 09:50
/etc/motd
----- System /etc/motd Begin -----
----- System /etc/motd End -----
----------------------------------------------
--------------

----- File Systems Mounted Begin -----
/dev/hda2 on / type ext3 (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
udev on /dev type tmpfs (rw)
devpts on /dev/pts type devpts
(rw,mode=0620,gid=5)
hugetlbfs on /libhugetlbfs type hugetlbfs
(rw,mode=0777)
velvet05:/ostest on /ostest type nfs
(rw,vers=3,addr=172.30.74.83)
velvet01:/home on /home type nfs
(rw,vers=3,addr=172.30.74.58)
iss:/.fs/a01 on /cray/iss/a0 type nfs
(rw,vers=3,addr=172.30.31.44)
. . .
iss:/.fs/libs_src/ulib on /cray/iss/ulib type
nfs
(rw,vers=3,addr=172.30.31.44)
iss:/.fs/u09/u5 on /cray/iss/u5 type nfs
(rw,vers=3,addr=172.30.31.44)
nfsd on /proc/fs/nfsd type nfsd (rw)
nid00002@gni:/scratch on /lus/scratch type
lustre (rw,flock)
----- File Systems Mounted End -----
```

Each test is bracketed by `<<<test_start>>>` and `<<<test_end>>>`. Any output from the test engine is preceded by `<<<test_output>>>` and status is preceded by `<<<execution_status>>>`. In each of the tests below, the `tag=CL_LTPFSXnnn` is the unique symbolic name generated for the specific test and `stime` is the decimal value of the internal clock when the test was started. Each `cmdline=` is the actual command executed by the test engine. Additional test environment variables and status is also displayed. In the test output, the pass/fail status is preceded by the tests symbolic name.

```
<<<test_start>>>
tag=CL_LTPFSX027 stime=1232118631
cmdline="ubrun -t -x -t -D -T CL_LTPFSX027 -e
LTPROOT_CL
```

The first test is a Linux file system exerciser, `fsx`. Note the test success message `PASS` printed prior to the execution status.

```
<<<test_start>>>
tag=CL_LTPFSX027 stime=1232118631
cmdline="ubrun -t -x -t -D -T CL_LTPFSX027 -e
LTPROOT_CL…
aptrun -n 1 LTPROOT_CL=testcases/bin/fsx …
-linux -d -l 500000 -r 4096 -t 2048 -w 2048 -W
-N 10000…
junkfile0.000000"
contacts="darason"
analysis=cuts
initiation_status="ok"
<<<test_output>>>
ubrun: Env
LTPROOT_CL=/ostest/dev.gemini/xtcnl/ltp/ROOT.l
atest
ubrun: Execute Cmd: '  aptrun -n 1
/ostest/dev.gemini/xtcnl/ltp/ROOT.latest/testc
ases/bin/fsx
```

```
-linux -d -l 500000 -r 4096 -t 2048 -w 2048 -W
-N 10000
junkfile%f'
CL_LTPFSX027    1  PASS  :  No failures found
with the command 'aptrun'
+ The return value was 0 as expected.
<<<execution_status>>>
duration=241 termination_type=exited
termination_id=0 corefile=no
cutime=164 cstime=38
<<<test_end>>>
```

This test, `fcntl14`, is another I/O test. Notice the test failure informational messages flagged with `FAIL` prior to the execution status.

```
<<<test_start>>>
tag=CL_LTPfcntl14 stime=1232119204
cmdline="ubrun -t -D -o -T CL_LTPfcntl14 -e
LTPROOT_CL
aptrun -n 1 LTPROOT_CL=testcases/bin/fcntl14"
contacts="darason"
analysis=cuts
initiation_status="ok"
<<<test_output>>>
ubrun: Env
LTPROOT_CL=/ostest/dev.gemini/xtcnl/ltp/ROOT.l
atest
ubrun: Execute Cmd: '  aptrun -n 1
/ostest/dev.gemini/xtcnl/ltp/ROOT.latest/testc
ases/bin/fcntl14'
CL_LTPfcntl14    1  FAIL  :  There was a
failure when executing 'aptrun'.:
    - The return value was 1, expected 0.
See output below:
aptrun : Final launch cmd is ' aprun    -n 1
/ostest/dev.gemini/xtcnl/ltp/ROOT.latest/testc
ases/bin/fcntl14 '
fcntl14     0  INFO  :  Enter block 1: without
mandatory locking
fcntl14     1  FAIL  :  First parent lock
failed
fcntl14     2  FAIL  :  Test case 1, errno =
38
.
.
.
fcntl14     0  INFO  :  Exit block 4
Application 416 exit codes: 1
Application 416 resources: utime 0, stime 0
<<<execution_status>>>
duration=11 termination_type=exited
termination_id=1 corefile=no
cutime=26 cstime=17
<<<test_end>>>
```

The following Lustre simulation example is another run of GROMACS that was demonstrated at Technical Milestone 2. Aside from the GROMACS computational segment, what is of interest here is that GROMACS is actually run from a Lustre scratch file system mounted in the simulation environment. In this case, Lustre is tested by inference via the application launch with ALPS command `aprun`.

```
godfrey@nid00001:/lus/scratch/godfrey/RUN/d.dppc>
aprun -n 4 ../mdrun
[PE_0]: inet_ipaddr_from_dev: ioctl SIOCGIFADDR
call failed 19
NNODES=4, MYRANK=0, HOSTNAME=nid00004
NNODES=4, MYRANK=2, HOSTNAME=nid00006
NNODES=4, MYRANK=3, HOSTNAME=nid00007
NNODES=4, MYRANK=1, HOSTNAME=nid00005
 . . .
             :-)  G  R  O  M  A  C  S  (-:

GRoups of Organic Molecules in Action
for Science

               :-)  VERSION 3.2.1  (-:
 . . .
M E G A - F L O P S   A C C O U N T I N G

   Based on real time for parallel computer.
RF=Reaction-field Free=Free Energy SC=Softcore
T=Tabulated  S=Solvent W=Water  WW=Water-Water

Computing:      M-Number      M-Flop's  % Flop's
        LJ     280.116462   8683.610322     9.9
 Coulomb     241.217644   6512.876388     7.5
Coulomb(W)      32.240146   2611.451826     3.0
Coulomb(WW)     50.441971  11803.421214    13.5
LJ + Coulomb   138.795587   5274.232306     6.0
LJ + Coul(W)    59.749233   5496.929436     6.3
LJ + Coul(WW)  112.533799  27570.780755    31.6
Innerloop-Iatom 63.019825    630.198250     0.7
NS-Pairs       649.734587  13644.426327    15.6
Reset In Box     1.340416     12.063744     0.0
Shift-X         24.614912    147.689472     0.2
CG-CoM           0.664576     19.272704     0.0
Sum Forces      36.922368     36.922368     0.0
Angles           5.895168    960.912384     1.1
Propers          1.758208    402.629632     0.5
Impropers        0.310272     64.536576     0.1
RB-Dihedrals     2.482176    613.097472     0.7
Virial          12.318364    221.730552     0.3
Update          12.307456    381.531136     0.4
Stop-CM         12.185600    121.856000     0.1
Calc-Ekin       12.429312    335.591424     0.4
Lincs            5.168128    310.087680     0.4
Lincs-Mat       72.142848    288.571392     0.3
Shake-V         12.307456    184.611840     0.2
Shake-Vir       12.307456    221.534208     0.3
Settle           2.425856    783.551488     0.9
Total                       87334.11690   100.0

               NODE (s)   Real (s)      (%)
       Time:    128.000    128.000    100.0
                       2:08
Performance:
(Mnbf/s) (MFlops) (ps/NODE hour) (NODE hour/ns)
18.772    682.298      5.625      177.778

gcq#132: "It's Not Your Fault" (Pulp Fiction)

Application 251 resources: utime 0, stime 0
godfrey@nid00001:/lus/scratch/godfrey/RUN/d.dppc>
```