# Table of Contents

Cray XE and XT Systems Application Programming and Optimization ........................................... 9  
Course Description ................................................................................................................ 10  
Topic and Daily Schedule .................................................................................................. 11  
Reference Documents ........................................................................................................ 12  
Cray System Overview ........................................................................................................ 13  
Overview Topics ................................................................................................................ 14  
Cray System ....................................................................................................................... 15  
System Overview ............................................................................................................... 16  
System Overview (External Services) ................................................................................ 17  
Cabinet ............................................................................................................................... 18  
Liquid Cooled Cabinets ..................................................................................................... 19  
Blades ................................................................................................................................ 20  
Cray XT and Cray XE Systems .......................................................................................... 21  
XT5 Compute Blade ........................................................................................................... 22  
XT5 Node Block Diagram .................................................................................................. 23  
X6 Compute Blade with Gemini ASICs ............................................................................. 24  
XE6 Node Block Diagram .................................................................................................. 25  
XK6 Compute Blade ......................................................................................................... 26  
NVIDIA Tesla™ X2090 GPU ............................................................................................. 27  
XIO Service Blades ............................................................................................................ 28  
XIO Blade .......................................................................................................................... 29  
SeaStar ASIC ..................................................................................................................... 30  
SeaStar Diagram ................................................................................................................. 31  
Node Connection Bandwidths ......................................................................................... 32  
Portals API Design ............................................................................................................ 33  
Portals API Design ............................................................................................................ 34  
Gemini ............................................................................................................................... 35  
Performance and Resiliency Features ............................................................................... 36  
Terminology ....................................................................................................................... 37  
Gemini Block Diagram ...................................................................................................... 38  
Gemini vs SeaStar – Topology ....................................................................................... 39  
Node Types ......................................................................................................................... 40  
Service Nodes .................................................................................................................... 41  
Service Nodes .................................................................................................................... 42  
Service Node Software Components .............................................................................. 43  
System Networks ............................................................................................................... 44  
Network Topology Classes .............................................................................................. 45  
A Single-chassis Configuration ......................................................................................... 46  
Class 0 Cable Drawing, 3 Cabinets .................................................................................. 47  
Class 1 Cable Drawing, 4 Cabinets .................................................................................. 48  
Class 2 Cable Drawing, 16 Cabinets ............................................................................... 49  
Class 3 Topology ............................................................................................................... 50  
HSN Cabling Photo ........................................................................................................... 51  
Cray XT5m Topology ......................................................................................................... 52
**Cray XE and XK Application Programming and Optimization Student Guide**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifying Components</td>
<td>53</td>
</tr>
<tr>
<td>Node ID (NID)</td>
<td>54</td>
</tr>
<tr>
<td>Cabinet 0 NID Numbering Example</td>
<td>55</td>
</tr>
<tr>
<td>DVS, DSL (Dynamic Shared Libraries), and Cluster Compatibility Mode (CCM)</td>
<td>57</td>
</tr>
<tr>
<td>DVS</td>
<td>58</td>
</tr>
<tr>
<td>DVS</td>
<td>59</td>
</tr>
<tr>
<td>Dynamic Shared Objects and Libraries (DSL)</td>
<td>60</td>
</tr>
<tr>
<td>DSL Diagram</td>
<td>61</td>
</tr>
<tr>
<td>DSL configuration with DVS</td>
<td>62</td>
</tr>
<tr>
<td>Cluster Compatibility Mode (CCM)</td>
<td>63</td>
</tr>
<tr>
<td>CCM Batch Integration</td>
<td>64</td>
</tr>
<tr>
<td>CCM Process Flow</td>
<td>65</td>
</tr>
<tr>
<td>Lustre File System</td>
<td>67</td>
</tr>
<tr>
<td>I/O Support</td>
<td>68</td>
</tr>
<tr>
<td>I/O Architecture</td>
<td>69</td>
</tr>
<tr>
<td>Node Interaction</td>
<td>70</td>
</tr>
<tr>
<td>Lustre Terminology</td>
<td>71</td>
</tr>
<tr>
<td>Lustre Diagram</td>
<td>72</td>
</tr>
<tr>
<td>Using the df Command</td>
<td>73</td>
</tr>
<tr>
<td>Lustre Commands</td>
<td>74</td>
</tr>
<tr>
<td>Client View of File System Space</td>
<td>75</td>
</tr>
<tr>
<td>File Striping and Inheritance</td>
<td>76</td>
</tr>
<tr>
<td>lfs Command</td>
<td>77</td>
</tr>
<tr>
<td>lfs Command Example</td>
<td>78</td>
</tr>
<tr>
<td>External Services</td>
<td>79</td>
</tr>
<tr>
<td>Why External Services for Cray Systems</td>
<td>80</td>
</tr>
<tr>
<td>External Services</td>
<td>81</td>
</tr>
<tr>
<td>Direct Attached Lustre</td>
<td>82</td>
</tr>
<tr>
<td>Lustre Diagram</td>
<td>83</td>
</tr>
<tr>
<td>External Lustre Diagram</td>
<td>84</td>
</tr>
<tr>
<td>Basic Elements of an esFS Configuration</td>
<td>85</td>
</tr>
<tr>
<td>esLogin Typical HW Configuration</td>
<td>86</td>
</tr>
<tr>
<td>esLogin</td>
<td>87</td>
</tr>
<tr>
<td>es Data Mover (esDM)</td>
<td>88</td>
</tr>
<tr>
<td>CRAY SONEXION™</td>
<td>89</td>
</tr>
<tr>
<td>Cray Sonexion</td>
<td>90</td>
</tr>
<tr>
<td>Cray System</td>
<td>91</td>
</tr>
<tr>
<td>Cray Sonexion Hardware Overview</td>
<td>92</td>
</tr>
<tr>
<td>Cray Sonexion Hardware Overview</td>
<td>93</td>
</tr>
<tr>
<td>Sonexion Base Rack Example</td>
<td>94</td>
</tr>
<tr>
<td>Sonexion Expansion Rack Example</td>
<td>95</td>
</tr>
<tr>
<td>Sonexion MDU</td>
<td>96</td>
</tr>
<tr>
<td>Scalable Storage Unit (SSU) Hardware</td>
<td>97</td>
</tr>
<tr>
<td>Simple Routing</td>
<td>98</td>
</tr>
<tr>
<td>Round-Robin Fine Grained Routing</td>
<td>99</td>
</tr>
<tr>
<td>Programming Environment</td>
<td>101</td>
</tr>
</tbody>
</table>
Loopmark: Compiler Feedback ................................................................. 194
PGI Compiler .............................................................................................. 195
PGI compiler ............................................................................................... 196
PGI Fortran Compiler Flags ........................................................................ 197
Other PGI Compiler Information ............................................................... 198
Other PGI Compiler Information ............................................................... 199
PathScale Compiler .................................................................................... 200
Memory Access Patterns (L1) ................................................................. 201
Memory Access Patterns (L2) ................................................................. 202
Cache Optimization ................................................................................... 203
Two types of cache data reuse ................................................................. 204
Cache Blocking ......................................................................................... 205
Cache Use in Stencil Computations ......................................................... 206
Blocking to Increase Reuse ....................................................................... 207
Blocking to Increase Reuse ....................................................................... 208
Initial Test ................................................................................................. 209
After Cache Blocking ................................................................................ 210
Cray XE MPI Environment ........................................................................ 211
MPICH2 and Cray MPT ............................................................................. 212
Gemini Software Stack ............................................................................. 213
How MPICH2 Uses GNI – Key Concepts ................................................ 214
How MPICH2 Uses GNI – SMSG Mailboxes .......................................... 215
Implications of GNI SMSG Mailboxes for Apps .................................. 216
MPICH2 GNI Netmod Message Protocols ............................................ 217
Max Message Size for E0 Varies with Job Size ..................................... 218
MPICH_GNI_MAX_VSHORT_MSG_SIZE .............................................. 219
MPICH_GNI_MAX_EAGER_MSG_SIZE ............................................... 220
MPICH_GNI_RDMA_THRESHOLD .......................................................... 221
MPICH_GNI_NDREG_LAZYMEM ......................................................... 222
MPICH_GNI_DMAPP_INTEROP .............................................................. 223
MPICH_GNI_NUM_BUFS ........................................................................ 224
MPICH_GNI_DYNAMIC_CONN ............................................................. 225
MPICH_GNI_MBOX_PLACEMENT ......................................................... 226
MPI_Allgather ......................................................................................... 227
MPI_Allgatherv ....................................................................................... 228
MPI_Alltoall ........................................................................................... 229
MPI_Allreduce/MPI_Reduce ...................................................................... 230
MPI_Bcast ............................................................................................... 231
MPICH_SMP_SINGLE_COPY_SIZE .................................................... 232
MPICH_SMP_SINGLE_COPY_OFF ......................................................... 233
Rank Placement ...................................................................................... 234
Rank Placement ...................................................................................... 235
Reordering example: GYRO ..................................................................... 236
Reordering example: TGYRO .................................................................... 237
Rank order choices .................................................................................. 238
Record of Revision

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 2010</td>
<td>Revision 0: Original printing</td>
</tr>
<tr>
<td>November 2010</td>
<td>Revision A: Minor updates</td>
</tr>
<tr>
<td>January 2011</td>
<td>Revision B: Minor updates</td>
</tr>
<tr>
<td>August 2011</td>
<td>Revision C: Added Interlagos information and removed duplicate information from the Single node Optimization chapter.</td>
</tr>
<tr>
<td>December 2011</td>
<td>Revision BW: Added Sonexion and customized materials for onsite delivery of class.</td>
</tr>
</tbody>
</table>
Cray XE and XK Systems
Application Programming and Optimization

Customer Documentation and Training
Course Description

• Designed for people who work in applications support or development for Cray XE and Cray XK computer systems
• Familiarizes students with the hardware and software architecture, programming models, and optimization of user codes that run on Cray XE and Cray XK systems
• Provides sufficient instruction to inexperienced users to enable them to develop and optimize applications

Note: In this course documentation, the term Cray XT4 system sometimes appears as XT4, the term Cray XT5 system sometimes appears as XT5, and the term Cray XT6 system sometimes appears as XT6. The term XT refers to the Cray XT4, Cray XT5, and Cray XT6 systems and sometimes appears simply as XT. Also, the term Cray XE6 system sometimes appears as XE6 and sometimes appears simply as XE. The term Cray SeaStar ASIC sometimes appears as the SeaStar and may refer to either the Cray SeaStar ASIC, the Cray SeaStar2 ASIC, or both. The term Cray Gemini ASIC sometimes appears as the Gemini.
### Topic and Daily Schedule

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>• System Overview</td>
<td>• DVS, DSL, CCM</td>
<td>• Node Optimization</td>
<td>• Cray XK6</td>
<td></td>
</tr>
<tr>
<td>• Lustre</td>
<td>• External Services</td>
<td>• XE MPI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Sonexion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Programming Environment</td>
<td>• Programming Environment</td>
<td>• DDT</td>
<td>• Cray XK6</td>
<td></td>
</tr>
<tr>
<td>• Debugging Tools</td>
<td>• Debugging Tools</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• PerfTools</td>
<td>• PerfTools</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This schedule illustrates only the order or sequence of topics. The duration of a particular instructional segment may change, depending on the progress of the class and the availability of machine resources.
Reference Documents

• TR-XEP_BW - Student workbook

• The following Cray software manuals are available
  ▪ Cray Linux Environment (CLE) Release Overview S-2425
  ▪ Cray Application Developer’s Environment User’s Guide S-2396
  ▪ Workload Management and Application Placement for the Cray Linux Environment S-2496
  ▪ Using Cray Performance Analysis Tools S-2376
  ▪ Cray Fortran Reference Manual S-3901
  ▪ Cray C and C++ Reference Manual S-2179
  ▪ Getting started on MPI I/O S-2490

  • A complete document number includes the software release number (for example: S-2376-31)

  – Visit www.cray.com and select Customer Support tab
  – Or your local site documentation repository
Cray System Overview

Customer Documentation and Training
Overview Topics

- Cray XE and Cray XK System Overview
- Cabinets, Chassis, and Blades
- Compute and Service Nodes
- Components of a Node
- System Networks
- Interconnection Topologies
This slide illustrates a class 2 Cray XE system that has 2 rows of 8 cabinets. The topology of the system \((X \times Y \times Z)\) is \(8 \times 24 \times 16 = 3,072\) Nodes (if all are compute nodes) the topology of each row is \(8 \times 24 \times 8\). Using mc12 sockets that could accommodate 73,728 cores.
Cray XT and Cray XE systems are circular (toroidal) in all three dimensions.
Service nodes run a full Linux distribution and can be configured to provide login, network, system, or I/O services:

- Login nodes provide a Linux-based full programming environment with all of the standard Linux commands, and shells. Load leveling distributes users over multiple login nodes.
- Network nodes provide high-speed connectivity to other systems.
- I/O nodes provide scalable connectivity to the Lustre file system.
  - Each I/O node has one or two Fibre Channel host bus adapter (HBA) cards(s) or InfiniBand host channel adapter (HCA) cards that connect to the system’s RAID storage.
  - I/O nodes provide Metadata services (MDS) and Object storage services (OSS) for the Lustre file.
- The boot node boots the rest of the system.
  - A system has one primary boot node and an optional second boot node.
- The database node manages the system database (SDB) and contains the compute PE states.
- The syslog node stores information from syslog daemons on other service nodes.
- The SMW (System Maintenance Workstation) is part of the Cray RAS (Reliability, Accessibility, Serviceability) and Management System (CRMS). The SMW communicates to all nodes and RAID disk controllers via a private Ethernet network.
Cray XT and Cray XE systems are circular (toroidal) in all three dimensions.

Service nodes run a full Linux distribution and can be configured to provide login, network, system, or I/O services:

- Login nodes provide a Linux-based full programming environment with all of the standard Linux commands, and shells. Load leveling distributes users over multiple login nodes.
- Network nodes provide high-speed connectivity to other systems.
- I/O nodes provide scalable connectivity to the Lustre file system.
  - Each I/O node has one or two Fibre Channel host bus adapter (HBA) cards(s) or InfiniBand host channel adapter (HCA) cards that connect to the system’s RAID storage.
  - I/O nodes provide Metadata services (MDS) and Object storage services (OSS) for the Lustre file.
- The boot node boots the rest of the system.
  - A system has one primary boot node and an optional second boot node.
- The database node manages the system database (SDB) and contains the compute PE states.
- The syslog node stores information from syslog daemons on other service nodes.
- The SMW (System Maintenance Workstation) is part of the Cray RAS (Reliability, Accessibility, Serviceability) and Management System (CRMS). The SMW communicates to all nodes and RAID disk controllers via a private Ethernet network.
Early system documentation sometimes speaks of a chassis as a cage. In the original plan, a cage was an unconfigured chassis. Some components, such as the cage controller, still retain the old naming conventions. Partially populated chassis are not approved; all eight slots must contain service or compute blades. An unconfigured chassis contains filler plates to ensure proper air flow.
Chapter 1: System Overview

Liquid Cooled Cabinets
An SIO service blade has four SeaStar Application Specific Integrated Circuits (ASICs) of which only two are in use at any location. Because of this arrangement, you can install a service blade anywhere in the system, without concern for its orientation within the high speed network. The SIO blade is configured with either PCI-X or PCIe risers.

An XIO service blade has two Gemini ASICs and four nodes. Each node has a single PCIe Gen 2 riser, except on a blade configured as “Boot node”. In a boot node configuration, the PCI riser for node 0 is used by node 1 allowing for two PCI cards for the boot node (Fibre Channel and GigE); in this configuration node 0 has no PCI connections.
**Cray XT and Cray XE Systems**

<table>
<thead>
<tr>
<th></th>
<th>2003 Cray XT3</th>
<th>2005 Cray XT4</th>
<th>2007 Cray XT5 Barcelona</th>
<th>2008 Cray XT5 Shanghai</th>
<th>2009 Cray XT5 Istanbul</th>
<th>2010 Cray XE6 Magny-Cours</th>
<th>2011 Cray XE6 Interlagos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>130nm SOI</td>
<td>90nm SOI</td>
<td>65nm SOI Socket F</td>
<td>45nm SOI Socket F</td>
<td>45nm SOI Socket F</td>
<td>45nm SOI Socket G34</td>
<td>32nm SOI Socket G34</td>
</tr>
<tr>
<td>CPU Core</td>
<td>K8</td>
<td>K8</td>
<td>Greyhound</td>
<td>Greyhound+</td>
<td>Greyhound+</td>
<td>Greyhound+</td>
<td>Bulldozer</td>
</tr>
<tr>
<td></td>
<td>2.0 - 2.8 GHz</td>
<td>2.1 - 2.8 GHz</td>
<td>2.1 - 2.3 GHz</td>
<td>2.1 - 2.3 GHz</td>
<td>2.2 GHz</td>
<td>MC8 2.4 GHz MC12 2.2 GHz</td>
<td>IL16 2.1 GHz</td>
</tr>
<tr>
<td>L2/L3</td>
<td>1MB/0</td>
<td>1MB/0</td>
<td>512kB/2MB</td>
<td>512kB/6MB</td>
<td>512kB/6MB</td>
<td>512kB/12MB</td>
<td>512kB/12MB</td>
</tr>
<tr>
<td>Hyper-Transport™ Technology</td>
<td>3x 1.6GT/s</td>
<td>3x 1.6GT/s</td>
<td>3x 2GT/s</td>
<td>3x 4.0GT/s</td>
<td>3x 4.8GT/s</td>
<td>4x 6.4GT/s</td>
<td>4x 6.4GT/s</td>
</tr>
<tr>
<td>Memory</td>
<td>2x DDR1 300</td>
<td>2x DDR1 400</td>
<td>2x DDR2 667</td>
<td>2x DDR2 800</td>
<td>2x DDR2 800</td>
<td>4x DDR3 1333</td>
<td>4x DDR3 1333</td>
</tr>
<tr>
<td>Interconnect</td>
<td>SeaStar</td>
<td>SeaStar</td>
<td>SeaStar</td>
<td>SeaStar</td>
<td>SeaStar</td>
<td>Gemini</td>
<td>Gemini</td>
</tr>
</tbody>
</table>

SOI - Silicon on insulator
A Cray XT5 compute blade has four logical nodes. Each logical node contains two quad- or six-core AMD Opteron processors. The quad-core processors are Barcelona or Shanghai, the six-core processors are Istanbul. The logical node acts as an 8- or 12-way SMP compute node. The compute nodes run CNL.
Chapter 1: System Overview

The AMD Opteron processor has the following features:
- 64-bit addressing
- x86 instruction set
- Out-of-order execution
- Multiple floating-point units
- Issues 9 instructions simultaneously
- 16 64-bit registers
- Full 64-bit IEEE floating-point arithmetic
- Full 64-bit integer arithmetic

The Cray SeaStar ASIC has the following features:
- HyperTransport interface to Opteron
- Message passing interface
- High-speed network router
- System management
Five 52Vdc-to-12Vdc VRMs (there are six sockets; one is currently unused)
Eight node VRMs
Eight G34 processor sockets with individual heat sinks
32 memory DIMMs
L0 controller (integrated in blade PCA)
SeaStar or Gemini mezzanine (SeaStar VRM is replaceable, Gemini VRM is integrated into mezzanine)
No hot-swap switch
The AMD Opteron processor has the following features:

- 64-bit addressing
- x86 instruction set
- Out-of-order execution
- Multiple floating-point units
- Issues 9 instructions simultaneously
- 16 64-bit registers
- Full 64-bit IEEE floating-point arithmetic
- Full 64-bit integer arithmetic
NVIDIA Tesla™ X2090 GPU

- CUDA Core count 512
  - Peak Performance (DP) 665GFlops
- Memory type and size
  6GB of GDDR5 (ECC)
  - Memory bandwidth 177 GBs
- Custom heat sink for XK6 blade
XIO Service Blades

- Four nodes; each node contains:
  - One AMD Opteron processor with up to 16 GB of DDR2 memory
    - Processor is a six-core Opteron
  - A connection to a Gemini ASIC
  - Voltage regulating modules (VRMs)
- L0 controller
- Gemini mezzanine card
  - Contains two Gemini ASICs
- Four PCIe risers
  - One riser per node
    - GigE, 10-GigE, Fibre Channel, or InfiniBand
Chapter 1: System Overview

XIO Blade

node 3 connects to the inner mezzanine

Node 3

Node 2

Gemini 1

Node 1

Gemini 0

Node 0

node 1 connects to the inner mezzanine
SeaStar ASIC

- Direct HyperTransport connection to Opteron
- The DMA engine transfers data between the host memory and the network
  - Separate read and write sides in the DMA engine
  - The Opteron does not directly load/store to network
  - On the receive side, a set of content addressable memory (CAM) locations are used to route incoming messages
    - There are 256 CAM entries
- Designed for Sandia Portals API

*Portals* is the low-level communications protocol that supports systems that contain large numbers of nodes. MPI is implemented on top of portals.
Chapter 1: System Overview

The above slide represents an XT3 or XT4 compute blade and a service blade; it does not represent an XT5 compute blade, an XT5 would have two sockets.

**Node Characteristics**

- 1- to 8-GB DRAM on compute and service nodes
  - Some systems do have 16-GB service nodes, but the 4-GB DIMMs are no longer available
- 384 KB of RAM for the SeaStar firmware
- HyperTransport links
- Router x/y/z ports have peak bidirectional bandwidth of 9.6 GBps
- Static routing

**The L0 controller**

- Runs a realtime Linux operating system
- Accesses the SeaStar memory and status registers
- Provides booting, monitoring, and maintenance
- Connects to the L1 controller for the cabinet via 100BaseT Ethernet
  - All L1s connect to the SMW through a tree of Ethernet switches
- The L0 controller connects to the node through the SSI (synchronous serial interface) bus.
Latency is the overhead associated with sending a zero-length message between two MPI tasks. Total latency is a combination of both hardware and software factors; the software contribution is generally much greater than that of the hardware.

Bandwidth is the rate at which data can be transmitted between two MPI tasks. Like latency, bandwidth is a combination of both hardware and software factors.
### Portals API Design

- Designed to message among thousands of nodes
  - Performance is critical only in terms of scalability
  - Success is measured by how large an application is allowed to scale, not by a 2-node ping-pong time
- Designed to avoid scalability limitations
  - Is network independent
  - Bypasses OS – no memory copies into or out of the kernel; few interrupts
  - Bypasses application – does not require activity by the application to ensure forward progress of the message
  - Reliable, ordered delivery of messages between two nodes

Portals API Design

- Receiver (target) managed – no reserved memory for thousands of potential senders
  - Is connectionless; no explicit connection is established with other processes in the application
    - When connected, maintains a minimum amount of state
  - The target process determines how to respond to the incoming message
    - The target node can choose to accept or ignore a message from any node
    - Destination of any message is not an address
      - Instead “Match bits” enable the receiver to place it
  - The unexpected message queue stores unexpected messages
  - All buffers are in user space
Gemini

- Gemini is designed to pass data to and from the network with less control
  - Gemini is suited for SMP
  - Gemini provides for adaptive routing
  - Hardware support for PGAS (Partitioned Global Address Space) languages
    - Programming language extensions such as Unified Parallel C (UPC), Co-array Fortran (CAF), and Cray SHMEM
    - Remote memory address translations, as with Cray X2 systems
  - Atomic memory operations
- The Gemini chip uses HyperTransport 3.0
  - The HT3 links support automatic link-level retries and improved CRC
Performance and Resiliency Features

- The Gemini interconnect provides:
  - Congestion feedback to enable routing around network bottlenecks
    - Packets are reordered in the receive buffer
    - Separate completion notification when all are stored
  - Automatic link failure handling with software help
    - To route around a failed link, system traffic must be quiesced to prevent deadlock and preserve ordering
    - This feature is also used to support hot blade swap

quiesce – to temporarily deactivate or disable
Chapter 1: System Overview

Terminology

- FMA – Fast Memory Access
- BTE – Block Transfer Engine
- DMAPP – Distributed Memory Application
- GNI – Generic Network Interface
  - uGNI – user level GNI
  - kGNI – kernel level GNI
- GHAL – Generic Hardware Abstraction Layer
Chapter 1: System Overview

Gemini vs SeaStar – Topology

XT6 Module with SeaStar

XE6 Module with Gemini

2 links in x,z

1 link in y
### Node Types

- **Compute nodes**
  - No user login
  - Only for execution of parallel applications
  - Diskless - no permanent storage
  - CNL operating system
    - Linux symmetrical multiprocessing (SMP)
    - Supports OpenMP programming model
    - `apinit` part of ALPS (Application Level Placement Scheduler)

- **Service Nodes**
  - Provide services based on hardware/software configuration
  - Node names: boot node, SDB, syslog, login, I/O, and network nodes
  - Run Linux operating system - SLES

---

SLES - SuSE Linux Enterprise Server

Although underlying support for TCP/IP sockets is available on CNL, rsh and ssh are not supported on CNL and a user cannot rsh or ssh into any compute nodes with their sockets.
PBS Pro behaves differently on Cray XT/XE systems than on other clusters (discussed in the “Component Interactions” Section of this course). Most clusters have one `pbs_mom` per compute node, but XT/XE systems run only a copy of `pbs_mom` on each login node.

Some limits that PBS can normally impose on jobs are not meaningful for Cray systems. For example, PBS supports the option to limit CPU seconds or memory that a job consumes; `pbs_mom` supports these limits. However, in regard to Cray systems, `pbs_mom` runs only on the login node and applies the limits only to the job script, which is not useful.
### Service Nodes

- **I/O nodes**
  - Attach to RAID storage devices
  - Perform remote DMA I/O through the HSN
  - Function as a metadata server (MDS) and object storage servers (OSSs) for Lustre file systems

- **Network nodes**
  - 10 Gigabit Ethernet (10GbE) adapter
  - Connect to other file servers

Lustre documentation is available at:
http://www.oracle.com/us/products/servers-storage/storage/storage-software/031855.htm or
Service Node Software Components

- ALPS - Application Level Placement Scheduler
  - Application placement, launch, and management for CNL
  - Includes several daemons and client processes
- RCA – Resiliency Communication Agent
  - Part of the kernel, on all nodes
  - Sends a heartbeat message to the SMW (CRMS/HSS/CMS)
  - Detects and responds to failing application launchers

aprun is the application launcher that initiates user compute node processes, similar to mpirun on other systems.
Topology is defined as the arrangement in which the nodes of a LAN are connected to each other. The HSS network, formerly named the CRMS (that is, Cray RAS [Reliability, Availability, and Serviceability] Maintenance System) Network, is examined in the Systems Administration class.

**System Networks**

- The High-speed Network (HSN) or interconnect network
  - Applications communicate to other compute and service nodes via the HSN
    - The network topology class defines the HSN configuration
      - The preferred topology is a folded torus
        - A torus is a circle
        - Folding enables the use of the shortest cables possible to connect all nodes within a dimension
        - In some configurations a dimension may be a mesh

- The Hardware Supervisory System (HSS) network
  - Monitors and controls the system
  - Multilevel Ethernet network
  - Connects the System Management Workstation (SMW) to the L0s
Network Topology Classes

- Configurations are based on topology classes:
  - Class 0 contains 1 to 9 chassis (1 to 3 cabinets)
    - 1 row
  - Class 1 contains 4 to 16 cabinets
    - 1 row
  - Class 2 contains 16 to 48 cabinets
    - 2 equal length rows
  - Class 3 contains 48 to 320 cabinets
    - Three rows of equal length
      - A mesh in one or more dimensions
    - Even number of 4 or more equal length rows
      - Torus in all dimensions
  - Mini is a class 0 with a 2D torus, no X dimension
    - 1 to 18 chassis, single row
Cray XT systems:

- The four SeaStar ASICs in a blade connect in the mezzanine board in the Y dimension.
- The eight blades in a chassis connect in the chassis backplane in the Z dimension (SeaStar 0 to 0, 1 to 1, 2 to 2, and 3 to 3).
- Only the X dimension and the ends of the Y and Z dimensions must be cabled.
- Service blades also have four SeaStar ASICs; therefore, the eight blades can be a combination of compute and service blades.

Cray XE systems:

- The two Gemini ASICs in a blade connect in the mezzanine board in the Y dimension.
- The eight blades in a chassis connect in the chassis backplane in the Z dimension (Gemini 0 to 0, and 1 to 1).
- Only the X dimension and the ends of the Y and Z dimensions must be cabled.
Class 0 topology

- 1 to 9 chassis (1 to 3 cabinets)
- All cabinets are in a single row
- The system scales in units of chassis
- $N \times 4 \times 8$ topology
  - $N$ is the number of populated chassis in the system
- Torus in all 3 dimensions

In a Class 0 topology, one cable per chassis connects the two Z-dimension connectors to complete an 8x torus (across the blades in the chassis).

Each chassis also uses one Y-dimension cable set (two cables) to complete a 4x torus (across the four SeaStars in a blade).

The X dimension is cabled as a single torus with a radix equal to the number of populated chassis in the system.
Class 1 topology

- 4 to 16 cabinets
- All cabinets are in a single row
- The system scales in units of fully populated cabinets
- \( N \times 12 \times 8 \) topology
  - \( N \) is the number of cabinets in the system
  - Torus in all 3 dimensions

In a Class 1 topology, the Z-dimension cables connect to a single chassis, the Y-dimension cables connect the three chassis within a single cabinet, and the X-dimension cables connect the same chassis numbers across the cabinets.

One cable per chassis connects the two Z-dimension connectors to complete an 8x torus (across the blades in the chassis).

Three Y-dimension cable sets (two cables per set) are required per cabinet to connect the three chassis within the cabinet in a 12x torus (4 SeaStars per blade x 3 chassis). The Y-dimension cabling is identical in each cabinet.

The X dimension is cabled across the cabinets as three folded tori. All chassis 0s form one torus, the chassis 1s form a second torus, and the chassis 2s form the third torus. The radix of each torus is the same and is equal to the number of cabinets in the system. Each chassis in the system requires one X-dimension cable set of eight cables (three sets per cabinet).
In a Class 2 system, the Z-dimension cables connect two chassis in a cabinet column, the Y-dimension cables remain within a single cabinet, and the X-dimension cables connect the same chassis numbers across cabinets in a cabinet row.

Three sets of Y-dimension cables connect the three chassis within each cabinet to complete a 12x torus (4 SeaStars per blade x 3 chassis). One of two cabling schemes is used inside the cabinet. All cabinets in row 0 use one Y-dimension cabling scheme, and all cabinets in row 1 use an alternate cabling scheme.

Six Z-dimension cables per cabinet column build three 16x tori (two chassis per torus). The Z-dimension tori connect: chassis 0 in row 1 to chassis 2 in row 0; chassis 1 in row 1 to chassis 1 in row 0; and chassis 2 in row 1 to chassis 0 in row 0. This cabling method uses a single cable length for all Z-dimension connections. All Z-dimension cabling is identical across each cabinet column.

The X dimension is cabled across the cabinets as 3 folded tori in each cabinet row. All chassis 0s form one torus, the chassis 1s form a second torus, and the chassis 2s form a third torus. The radix of each torus is the same and is equal to the number of cabinets in the row.
Class 3 Topology

- 48 to 320 cabinets
- Three rows of equal length
  - A mesh in one or more dimensions
- 4 to 10 rows; even number of rows
  - 12 to 32 cabinets per row
  - The number of cabinets in each row must be equal
  - Torus in all 3 dimensions
- The system scales in units of one fully populated cabinet times the number of rows in the system
- $N \times (4 \times \text{number of cabinet rows}) \times 24$
  - $N$ is the number of cabinets per row

In a Class 3 system, the Z-dimension cables connect the three chassis within a cabinet, the Y-dimension cables connect one chassis in each cabinet within a cabinet column, and the X-dimension cables connect the same chassis numbers across cabinets in a cabinet row.

This slide illustrates a Class 3 Cray XT system that has 6 rows of 18 cabinets. The topology of this system is $18 \times 24 \times 24$ (10,368 Nodes if all are compute nodes).
HSN Cabling Photo
Cray XT5m Topology

- 1 to 18 chassis, 1 to 6 cabinets
- All cabinets are in a single row
- The system scales in units of chassis
- 2D torus topology
  - y and z dimensions
  - \((Y \times 4) \times 8\) topology
    - \(Y\) is the number of populated chassis in the system
## Identifying Components

- **System components are labeled according to physical ID (HSS Identification), node ID, IP address, or class**

<table>
<thead>
<tr>
<th>Component</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>s0, all</td>
<td>All components attached to the SMW.</td>
</tr>
<tr>
<td>Cabinet</td>
<td>cX-Y</td>
<td>Cabinet number and row; this is the L1 host name.</td>
</tr>
<tr>
<td>Cage</td>
<td>cX-Yc#</td>
<td>Physical cage in cabinet: 0, 1, 2. Cages are numbered from bottom to top.</td>
</tr>
<tr>
<td>Blade or slot</td>
<td>cX-Yc#s#</td>
<td>Physical blade slot in cage: 0 – 7, numbered from left to right; this is the L0 hosts name.</td>
</tr>
<tr>
<td>Node</td>
<td>cX-Yc#s#n#</td>
<td>Opteron Chip on a blade: 0 - 3 for compute and XIO blades, 0 and 3 for SIO blades</td>
</tr>
<tr>
<td>SeaStar ASIC</td>
<td>cX-Yc#s#s#</td>
<td>Cray SeaStar ASIC on a blade: 0 – 3</td>
</tr>
<tr>
<td>Gemini ASIC</td>
<td>cX-Yc#s#g#</td>
<td>Cray Gemini ASIC on a blade: 0 – 1</td>
</tr>
<tr>
<td>Link</td>
<td>cX-Yc#s#s#1#</td>
<td>Link port of a SeaStar ASIC: 0 – 5</td>
</tr>
<tr>
<td></td>
<td>cX-Yc#s#g#1#</td>
<td>Link port of a Gemini ASIC: 0 – 5</td>
</tr>
</tbody>
</table>

**Examples:**
- c0-0: Cabinet 0-0, X position 0 within row 0.
- c12-3: Cabinet 12-3, X position 12 within row 3.
- c*-*: Wildcard: All cabinets within the system.
- c0-0c2: Cage 2 of cabinet 0-0.
- c0-0c*: Wildcard: All cages within cabinet c0-0.
- c0-0c0s4: Slot (module) 4 of cage 0 of cabinet c0-0.
- c0-0c0s*: All slots (0..7) of cage 0 of cabinet c0-0.
- c0-0c0s0n3: CPU 3 on module slot 0, cage 0, cabinet c0-0.
- c0-0c0s0n*: All CPUs (0..3) on module slot 0, cage 0, cabinet c0-0.
- c0-0c0s0s2: SeaStar 2 on module slot 0, cage 0, cabinet c0-0.
- c0-0c0s0s*: All SeaStars (0..3) on module slot 0, cage 0, cabinet c0-0.
- c0-0c0s0s0l4: Port 4 (port E) of SeaStar 0 on module slot 0, cage 0, cabinet c0-0.
- c0-0c0s0s0l*: All ports (0..5) of SeaStar 0 on module slot 0, cage 0, cabinet c0-0.
Node ID (NID)

- The Node ID is a unique hexadecimal or decimal number that identifies each CLE node
- The NID reflects the node location in the network
  - In a SeaStar based system, the NIDs are assigned on 128-number boundaries for each cabinet
  - In a Gemini based system, NIDs are sequential
  - NID format is \texttt{nidnnnn} (decimal) when it refers to a hostname of a node, for example: \texttt{nid00003}
# Cabinet 0 NID Numbering Example

<table>
<thead>
<tr>
<th>Cray XT (SeaStar based) system</th>
<th>Cray XE (Gemini based) system</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 2, 3</td>
<td>0, 1, 30, 31</td>
</tr>
<tr>
<td>4, 5, 6, 7</td>
<td>2, 3, 28, 29</td>
</tr>
<tr>
<td>8, 9, 10, 11</td>
<td>4, 5, 26, 27</td>
</tr>
<tr>
<td>12, 13, 14, 15</td>
<td>6, 7, 24, 25</td>
</tr>
<tr>
<td>16, 17, 18, 19</td>
<td>8, 9, 22, 23</td>
</tr>
<tr>
<td>20, 21, 22, 23</td>
<td>10, 11, 20, 21</td>
</tr>
<tr>
<td>24, 25, 26, 27</td>
<td>12, 13, 18, 19</td>
</tr>
<tr>
<td>28, 29, 30, 31</td>
<td>14, 15, 16, 17</td>
</tr>
<tr>
<td>32, 33, 34, 35</td>
<td>62, 63, 32, 33</td>
</tr>
<tr>
<td>36, 37, 38, 39</td>
<td>60, 61, 34, 35</td>
</tr>
<tr>
<td>40, 41, 42, 43</td>
<td>58, 59, 36, 37</td>
</tr>
<tr>
<td>44, 45, 46, 47</td>
<td>56, 57, 38, 39</td>
</tr>
<tr>
<td>48, 49, 50, 51</td>
<td>54, 55, 40, 41</td>
</tr>
<tr>
<td>52, 53, 54, 55</td>
<td>52, 53, 42, 43</td>
</tr>
<tr>
<td>56, 57, 58, 59</td>
<td>50, 51, 44, 45</td>
</tr>
<tr>
<td>60, 61, 62, 63</td>
<td>48, 49, 46, 47</td>
</tr>
<tr>
<td>64, 65, 66, 67</td>
<td>64, 65, 94, 95</td>
</tr>
<tr>
<td>68, 69, 70, 71</td>
<td>66, 67, 92, 93</td>
</tr>
<tr>
<td>72, 73, 74, 75</td>
<td>70, 71, 88, 89</td>
</tr>
<tr>
<td>76, 77, 78, 79</td>
<td>72, 73, 86, 87</td>
</tr>
<tr>
<td>80, 81, 82, 83</td>
<td>74, 75, 84, 85</td>
</tr>
<tr>
<td>84, 85, 86, 87</td>
<td>76, 77, 82, 83</td>
</tr>
<tr>
<td>88, 89, 90, 91</td>
<td>78, 79, 80, 81</td>
</tr>
</tbody>
</table>
DVS, DSL (Dynamic Shared Libraries), and Cluster Compatibility Mode (CCM)

Customer Documentation and Training
DVS

• Data Virtualization Services (DVS)
  – Projects files system mounted on service nodes to the compute nodes
  – Operating modes
    ▪ Cluster parallel mode
      • A single file is written to a single server
    ▪ Stripe parallel mode
      • A single file is written across multiple DVS servers
    ▪ Load balance mode
      • Work is distributed across the DVS servers
      • Supports failover
Example `fstab` file entries, with comments:

```
# stripe parallel mode
# mount /user3 from DVS servers c0-0c2s2n0 and c0-0c2s2n1 to /user3
# the lack of a maxnodes option means that files will be striped across
# multiple servers
# data is striped in 1MB chunks, since blksize=1048576 is specified
# (the default is 16k)
/user3 /user3 dvs path=/user3,nodename=c0-0c2s2n0:c0-0c2s2n1, \ 
blksize=1048576
```

```
# cluster parallel mode
# mount /user2 from DVS servers c0-0c2s2n0 and c0-0c2s2n1 to /user2
# maxnodes=1 means each file hashes to one and only one server from
# the list
/user2 /user2 dvs path=/user2,nodename=c0-0c2s2n0:c0-0c2s2n1,maxnodes=1
```
Dynamic Shared Objects and Libraries (DSL)

- DSL provides an alternate compute node root file system
  - Uses DVS to project shared root to the compute nodes
    - Projects shared root to the compute nodes
    - DSL uses Loadbalance mode to balance the workload between multiple DSL nodes
      - Loadbalance mode accommodates DSL failover
  - Allows users to link and load dynamic shared libraries
    - Requires user to relink or recompile their application
      \% cc -dynamic sample.c -o sample
      - Alternate method:
        \% export XTPE_LINK_TYPE=dynamic
        \% cc sample.c -o sample
    - New libraries are used without relinking their application
Chapter 2: DVS, DSL, and CCM

DSL Diagram

- The compute nodes have a DVS client running.

- The DSL nodes can be "service" or "compute" nodes.
- They are configured to mount shared root.
- They have a DVS server running.
- Used to project shared root to the compute nodes.

Shared Root file system
In load balanced mode, two or more DVS servers provide access to the same underlying file system. The DVS server used is determined by the NID number of the node making the request. If a DVS server fails, the client automatically fails over to the next server in the list.
Cluster Compatibility Mode (CCM)

- CCM was created to allow customers to run standard HPC software
  - CNL allows for *extreme scalability*, but it does not permit standard HPC software to run out of-the-box
- CCM provides the compute node runtime environment (CNRTE) expected by ISV applications
  - Dynamically allocates and configures compute nodes for the job duration
  - “shared root” distribution visible on compute nodes
    - Built on top of DSL (Dynamic Shared Libraries)

Under CCM, everything the application can “see” attempts to be identical to a standard Linux cluster: Linux OS, x86 processor, and MPI

- Third party MPI implementations runs over TCP/IP over HSN
- When the system is configured with CCM:
  - Nodes are allocated like any other batch job (on demand)
  - Nodes are not permanently dedicated to CCM
    - This avoids permanent changes that might add latency or affect performance of our traditional applications
    - CCM would be a small percentage of total system usage
  - Temporarily enable specific services for the lifetime of the job. Currently this is limited to `rsh` with options to configure `ssh`, `portmap`, `nis`, and `nscd`.
  - Additional fixes done through local loopback mounts (sysv5 `ipc` mechanisms and random device support)
Chapter 2: DVS, DSL, and CCM

**CCM Batch Integration**

- Integrated with PBSPro and Torque/Moab
  - Uses batch prologue / epilogue hooks to associate CCM capabilities with a specific batch queue
  - **Prologue**
    - Checks for valid environment (batch system, non-root user, DSL, valid queue (queue with prologue/epilogue))
    - Sets up ssh keys (if non-existent) and empty `.rhosts` file
    - Creates temporary files with `nodelist` and `rhosts`
    - Starts up a series of init scripts on the compute nodes using `xtxqtcmd`
  - **Epilogue**
    - Shuts down a series of init scripts using `xtxqtcmd`
    - Attempts to clean up job container
    - Runs node health tests

Batch system allocates the nodes prior to the start of the prologue/epilogue. This is a change in PBS, which was introduced in the 10.2 release.
I/O Support

- The compute nodes usually hand off I/O to the service nodes
- Lustre provides a parallel file system for application use
  - A Lustre file system consists of a Metadata server (MDS) and one or more Object Storage Targets (OSTs)
    - If you want to create another Lustre file system, you must configure it on separate disk devices
  - Other file system may be available to the compute nodes, but may not perform as well as Lustre
- The aprun application launcher handles stdin, stdout, and stderr for the application
The application launcher `aprun` provides I/O for standard input, output, and error.
The high-performance file system is Lustre, which provides a scalable, POSIX compliant file system.
Lustre Terminology:
MDS – metadata server (a service node)
  MDT – metadata target
  The software interface to the backend storage. Controls the file system metadata (inodes) and locking mechanism. The backend storage is an ldiskfs (ext3) file system. On Cray systems, this is a RAID volume
OSS – object storage server (a service node)
  Supports multiple OSTs
OST – object storage target
  The software interface to the backend storage
LOV – logical object volume, a combined structure of OSTs in a filesystem.
NAL – network abstraction layer
LNET – Lustre Network
LND – Lustre network device
Node Interaction

- **Clients**: File open, directory operations, metadata and concurrency
- **LOV**: File I/O and file locking
- **MDS**: Recovery, file status, file creation
- **OSS**:
Lustre Terminology

- **Stripe width** - the number of OSTs to write the file across
  - Cray recommends a default stripe width of one to four
    - Recommends against striping across all OSTs
  - Can be set either at the file or directory level
    - When you stripe at the directory level, new files inherit the stripe width of the parent directory
Lustre Diagram

In normal operation, Lustre distributes files across the OSTs in the file system.

For stripe count = 1
- File A
- File C

For stripe count = 2
- File B
- File D
- File E
- File F
Using the df Command

• Use the standard df command to locate the mount point for a Lustre file system

```
% df -t lustre
Filesystem   1K-blocks    Used Available Use% Mounted on
@ptl:/nid00008_mds/client  5644895560 4509828872 848322232 85% /lus/nid00008
36@ptl:/nid00036_mds/client 1128979112 447543988 624086236 42% /lus/nid00036
%
```
Chapter 3: Lustre File Systems

Lustre Commands

• `lfs` is a Lustre utility that can:
  - Provide file system configuration information
    `lfs df`
  - Create a file or directory with a specific striping pattern
    `lfs setstripe`
  - Display file striping patterns
    `lfs getstripe [directory | file name]`
  - Find file locations
    `lfs find [directory | file name]`
    ▪ For example, to find directories or files on a particular OST
    `lfs find -r -obd ost5_UUID /work/rns`
  - Display quota information
    `lfs quota -u|g <name> file system`

The stripe utility, `lfs`, enables you to stripe files after Lustre is configured. Normally, you stripe files by using the `lmc` command when you create the configuration file.
Client View of File System Space

- To view the individual OSTs

```
% lfs df

<table>
<thead>
<tr>
<th>UUID</th>
<th>1K-blocks</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>nid000008_mds_UUID</td>
<td>1003524776</td>
<td>579066856</td>
<td>945617920</td>
<td>5</td>
<td>/lus/nid000008[MDT:0]</td>
</tr>
<tr>
<td>ost0_UUID</td>
<td>1128979112</td>
<td>1094326220</td>
<td>34652892</td>
<td>96</td>
<td>/lus/nid000008[OST:0]</td>
</tr>
<tr>
<td>ost1_UUID</td>
<td>1128979112</td>
<td>1076393372</td>
<td>52585740</td>
<td>95</td>
<td>/lus/nid000008[OST:1]</td>
</tr>
<tr>
<td>ost2_UUID</td>
<td>1128979112</td>
<td>894139784</td>
<td>234839328</td>
<td>79</td>
<td>/lus/nid000008[OST:2]</td>
</tr>
<tr>
<td>ost3_UUID</td>
<td>1128979112</td>
<td>1006132924</td>
<td>122846188</td>
<td>89</td>
<td>/lus/nid000008[OST:3]</td>
</tr>
<tr>
<td>ost4_UUID</td>
<td>1128979112</td>
<td>725581028</td>
<td>403398084</td>
<td>64</td>
<td>/lus/nid000008[OST:4]</td>
</tr>
<tr>
<td>filesystem summary:</td>
<td>5644895560</td>
<td>4796573328</td>
<td>848322232</td>
<td>84</td>
<td>/lus/nid000008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UUID</th>
<th>1K-blocks</th>
<th>Used</th>
<th>Available</th>
<th>Use%</th>
<th>Mounted on</th>
</tr>
</thead>
<tbody>
<tr>
<td>nid000036_mds_UUID</td>
<td>1003524776</td>
<td>57871880</td>
<td>945652896</td>
<td>5</td>
<td>/lus/nid000036[MDT:0]</td>
</tr>
<tr>
<td>ost0_UUID</td>
<td>1128979112</td>
<td>504892876</td>
<td>624086236</td>
<td>44</td>
<td>/lus/nid000036[OST:0]</td>
</tr>
<tr>
<td>filesystem summary:</td>
<td>1128979112</td>
<td>504892876</td>
<td>624086236</td>
<td>44</td>
<td>/lus/nid000036</td>
</tr>
</tbody>
</table>
%
```
File Striping and Inheritance

- Lustre distributes files across all OSTs
- The default stripe width is set in the configuration file
- Users can create files and directories with various striping characteristics
  - New files inherit the striping of the parent directory
  - Striping across more OSTs generally leads to higher peak performance on large files, but may not be best for small files
  - CANNOT change the stripe pattern on an existing file
  - CAN change the stripe pattern on a directory
- Improper striping, such as in the following list, may result in inefficient use of your Lustre file system:
  - Writing a very large file to a single OST
  - Creating a directory where files do not circle through the OSTs
  - Striping a small file across many OSTs
**lfs Command**

- **Use the `lfs setstripe` command to manage striping characteristics**
  - To define striping for a file or directory:

    ```
lfs setstripe [--count c] [--pool p] [--offset o] [--size s] <dir|filename>
    ```

    | Option     | Description                                      |
    |------------|--------------------------------------------------|
    | --count    | stripe count, 0 means use the default           |
    | --pool     | name of OST pool                                |
    | --offset   | starting ost, -1 means use the default (round robin) |
    | --size     | stripe-size, 0 means use the default            |

    Defaults are defined in the Lustre configuration file

- **To view striping for a file or directory:**

  ```
lfs getstripe <file-name|dir-name>
  ```
Chapter 3: Lustre File Systems

### lfs Command Example

```
/rns> mkdir rick
/nid00008/rns> lfs getstripe rick
OBDS:
0: ost0_UUID ACTIVE
1: ost1_UUID ACTIVE
2: ost2_UUID ACTIVE
3: ost3_UUID ACTIVE
4: ost4_UUID ACTIVE
rick
(Default) stripe_count: 2 stripe_size: 1048576 stripe_offset: 0
/nid00008/rns>
/nid00004:/work # cd rick
/nid00004:/work/rick # touch file_one
   /rns/rick> lfs getstripe file_one
OBDS
0: ost0_UUID ACTIVE
1: ost1_UUID ACTIVE
2: ost2_UUID ACTIVE
3: ost3_UUID ACTIVE
4: ost4_UUID ACTIVE
file_one
       obdidx  objid       objid        group
            3   22169877  0x1529735    0
            4   23084122 0x1603c5a    0
```
Chapter 4: External Services

External Services

Customer Documentation and Training
Why External Services for Cray Systems

- To address customer requirements:
  - More flexible user access
  - More options for data management, data protection
  - Leverage commodity components in customer-specific implementations
  - Provide faster access to new devices and technologies
  - Repeatable solutions that remain open to custom configuration
  - Enable each solution to be used, scaled, and configured independently

![Image of Cray Systems components]
External Services

- esFS
  - Provides globally shared data between multiple systems
    - Cray XT/XE systems and others
  - Provides access to other file systems
    - DVS is used to project Panasas or StorNext to the compute nodes

- esLogin
  - Increased availability of data and system services to users
  - An enhanced user environment
    - larger memory, swap space, and more horsepower
    - Dell 905, 4 socket, quad-core and 128 GB of memory

- esDM
  - More options for data management and data protection
Direct Attached Lustre

File open, directory operations, metadata, and concurrency

Clients

LOV

File I/O and file locking

MDS

Recovery, file status, file creation

OSS
Basic Elements of an esFS Configuration

SIO or XIO
Blade
LNET Router

IB HCA

IB HCA

InfiniBand SW A

InfiniBand SW B

OSS1

OSS2

OSS3

OSS4

MGS, MDS1

MDS2

Ctrl A

Ctrl B

MDS
3992

1+1

1+1

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2

8+2
esLogin Typical HW Configuration

- esLogin hardware configuration:
  - Multi socket, multicore processors
  - Internal memory of 128 GB or more
  - Local disks for local root, swap, and tmp
  - NICs for connection to Cray system and customer network

- esLogin software configuration:
  - Connections to file systems, including External Lustre
  - Torque/Moab or PBS for job submission
    - The same version as the Cray system uses
  - Cray libraries, build tools, performance tools
  - Third-party compiler(s) and debugger(s)
Chapter 4: External Services

**es Data Mover (esDM)**

- High performance file sharing and backup/archive of Lustre data
- External client mounts Cray internal Lustre file systems

**external services Data Mover**

The standard implementation of a Lustre file system on Cray supercomputer nodes can limit the options available to customers for sharing and protecting data. Network File System (NFS) transfer over a 1GbE network to a Network Attached Storage (NAS) system has been the most popular solution, but this approach can be constrained by both bandwidth and cost considerations. Recent qualification of a 10GbE system interface addresses the bandwidth constraint, but the cost of a high performance NAS system can still be out of scope for the requirements of some customers.

Cray Custom Engineering now offers a lower cost alternative for sharing and backup of Lustre data, using a specially configured client known as External Services Data Mover (esDM). Cray Linux Environment™ (CLE) 2.2 and later releases support Lustre LNET routing on a specially configured service blade. The Lustre LNET router allows an external Lustre client to mount the Cray Lustre file systems, to share data, and copy data to other media including disk, tape and VTL. The service blade is configured with an InfiniBand (IB) Host Channel Adapter (HCA).
CRAY SONEXION™
Integrated Storage Systems for High Performance Computing
Customer Documentation and Training
Cray Sonexion

- Cray Sonexion™ is a new storage BRAND for High Performance Computing
  - Cray Sonexion (pronounced so-nex-ee-on) is an integrated storage system designed from scratch to run the Lustre file system in an optimal manner based on best practices experience following an appliance model
    - Lustre based for optimal performance
    - The hardware is integrated and modular to minimize complexity
    - The software tools are intuitive and provide instant monitoring of the system
Cray System
Cray Sonexion Hardware Overview

- Cray Sonexion systems have three major components:
  - A 42U rack
    - Contains power distribution, cabling, and switches
      - Two 18 or 36-port Mellanox InfiniBand (IB) switches
      - One 24 or 48-port Ethernet switch
      - Power options for US and EMEA
      - Optional water-cooled doors
  - A metadata unit (MDU)
    or metadata management unit (MMU)
    - Contains Lustre MDSs and MDT drives
      - Two 1U MDS (MDS and MGS in a failover pair)
      - One 2U storage tray
  - A Scalable Storage Unit (SSU)
    - One integrated unit containing two Lustre OSSs and OST drives

The base 18-port Mellanox switch is a Mellanox IS5023Q switch
An optional 36-port Mellanox IS5035Q switch can be used instead of the 18-port IB Switches.
Cray Sonexion Hardware Overview

- Each file system contains:
  - A base rack
    - The base rack contains 1 MDU and 1 to 7 SSUs
  - Optional expansion racks
    - Contain 1-7 SSUs
      - SSUs may be added up to Lustre stripe width
      - Every SSU in a file system must have the same disk capacity
- All components are managed via a single, separate, web-based GUI system
**Sonexion Base Rack Example**

- **Base Rack**
  - 2 1U InfiniBand Switches
  - 1 1U GigE Switch
  - 1 Cray Sonexion 1300 MMU
    - 2 1U external MDSs
    - 1 2U storage tray
  - 3 Cray Sonexion 1300 SSUs
    - Configured with 2 TB drives

> 360TB usable capacity
9GB/s IOR write
Sonexion Expansion Rack Example

- **Base Rack**
  - 2 36-port QDR Switches
  - 1 48-port GigE Switch
  - 1 Cray Sonexion 1300 MMU
  - 6 Cray Sonexion 1300 SSUs
  - Estimated Power Per Rack
    - Max: 18.8KW
    - Peak Operating: 14.4KW

- **Expansion Racks:**
  - 2 InfiniBand Switches
  - 1 GigE Switch
  - 6 Cray Sonexion 1300 SSUs
  - Estimated Power Per Rack
    - Max: 17.7KW
    - Peak Operating: 13.3KW
Sonexion MDU

- Two 1U external MDS
  - Dual QDR IB network links per node
- 2U24 storage tray enclosure
  - High availability configuration, no single point of failure
  - 22 Dual-ported 2.5” 10k SFF SAS Drives
    - 146GB, 450GB or 600GB drive options
  - 2 100GB SSD
    - Used for journaling
  - RAID10 data protection on SAS drives
  - One to three expansion enclosures with 600GB drives
- Power loss protection module
  - Protects data held in a volatile memory
Scalable Storage Unit (SSU) Hardware

- **Ultra high Density 5U84 enclosure with embedded OSSs**
  - Two high-availability Lustre OSSs per SSU
    - Each OSS has QDR IB cable to a rack IB switch
  - Two trays of 42 HDD’s each with 6Gb SAS interconnect
    - 80 Dual-ported 3.5” FatSAS 7.2K RPM Drives
      - Eight RAID6 8+2 OSTs
      - 4 OSTs per OSS
      - 1TB, 2TB or 3TB drives
    - 2 Dual-ported 3.5” 100GB SSD Configured as RAID1 (1x1)
      - One SSD drive per tray, located in the innermost left slot
      - Used for OST journaling for increased performance
    - 2 optional global hot spares
  - Optional expansion enclosure per SSU
    - Adds capacity, not bandwidth
Simple Routing

- A single LNET allows all routers on the XIO nodes to access all OSSs
- LNET communication requires inter-switch links between the leaf switches or a large core switch (not shown)
- Additional switch hops and inter-switch traffic causes InfiniBand congestion
Round-Robin Fine Grained Routing

- LNET groups link routers and OSSs connected to the same leaf switch.
- Aggregate bandwidth from the OSSs matches the aggregate bandwidth from the routers.
Programming Environment

Customer Documentation and Training
Cray Programming Environment

- A cross-compiler environment
  - Compiler runs on a login node
  - Executable runs on the compute nodes
- Modules utility
  - Consists of the module command and module files
  - Initializes the environment for a specific compiler
  - Allows easy swapping of compilers and compiler versions
- Cray written compiler driver scripts
  - CNL compiler options
  - CNL system libraries and header files
  - Compiler specific programming environment libraries

Module files, usually referred to as modules, are written in the Tool Command Language (tcl). Module files contain commands to configure the shell environment for a particular compiler, library, or utility. `ssh` is normally used to connect to the system.

User account information is maintained through an LDAP or Kerberos server.

You can set up passwordless `ssh` to access a system. You can also set up a pass phrase for a more secure session.
Cray Compilers

- Cray compilers (Cray Compiling Environment (CCE))
  - Provides additional support for Fortran 2003, Fortran with coarrays, and UPC (Unified Parallel C)
- GNU Compiler Collection (GCC)
- Portland Group Inc. (PGI) Compilers
- PathScale compiler support
- Intel compiler support
- All provide Fortran, C, C++, and OpenMP support
- Cray also provides the Chapel compiler
- So Which Compiler Do I Choose?
  - If your site offers you a choice, experiment with the various compilers
  - Mixing binaries created by different compilers may cause issues

PathScale has limited support for libraries.
PathScale and Intel compilers must be obtained from third-party sources, although Cray provides the modules and drivers necessary to invoke them.

SSE – Streaming SIMD Extensions
AVX – Advanced Vector Extensions
FMA4 – Fused Multiply-Add extensions (four operand, AMD specific)
XOP – SSE and AVX extensions (AMD specific)
SIMD – Single Instruction, Multiple Data
If the vendor compiler commands are called and the relevant module is not loaded, a login node binary is produced.

To find version information for the various compilers use \texttt{-V} with the PGI compilers and \texttt{-version} with the GCC and Pathscale compilers.

To see all of the information associated with compiling an application using a Fortran compiler you can use the option \texttt{--show} (when using the \texttt{show} option no binary is produced)

For example:

\begin{verbatim}
users/rns> ftn --show samp261.F
\end{verbatim}

The above command outputs a long list of information, too long to show here.

---

**Compiler Driver Scripts**

- **Do not call compilers directly; use Cray compile drivers**
  - \texttt{ftn}
  - \texttt{cc}
  - \texttt{CC}

- **Driver actions:**
  - Select compiler version
  - Add system libraries and header files
  - Add compiler-specific programming environment libraries
  - Execute the actual compiler command with added options
  - **Without DSL configured, executables are statically linked.**

- **Use vendor man pages for details of compiler options**
  - Cray man pages: crayftn, craycc, crayc++
  - GCC man pages: gfortran, gcc, g++
  - PGI man pages: pgf90, pgcc, pgCC
  - PathScale man pages: pathf90, pathcc, pathCC, eko
  - Intel man pages: icc, icpc, fpp, and ifort

---
## PE Summary

<table>
<thead>
<tr>
<th>PrgEnv-cray, PrgEnv-gnu, PrgEnv-pgi, PrgEnv-pathsclae, PrgEnv-intel</th>
<th>CCE, GCC, PGI, PathScale, and Intel compilers. All provide: C, C++, Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>xt-mpich2</td>
<td>MPICH2</td>
</tr>
<tr>
<td>xt-shmem</td>
<td>SHMEM</td>
</tr>
<tr>
<td>xt-mpt</td>
<td>MPICH2 and SHMEM (deprecated)</td>
</tr>
<tr>
<td>xt-libsci (Cray scientific libraries)</td>
<td>BLAS, LAPACK, BLACS, FFT, FFTW, CRAFFT, IRT, ScaLAPACK, and SuperLU_DIST</td>
</tr>
<tr>
<td>acml (AMD Core Math Libraries)</td>
<td>BLAS, LAPACK, FFT, RNG</td>
</tr>
<tr>
<td>Debuggers</td>
<td>Igdb, TotalView, and DDT (Allinea - Distributed Debugging Tool)</td>
</tr>
<tr>
<td>Performance tools</td>
<td>CrayPat, Apprentice2, PAPI library</td>
</tr>
<tr>
<td>Other Libraries</td>
<td>Fast_mv, PETSc, and Trilinos</td>
</tr>
</tbody>
</table>

BLAS (Basic Linear Algebra Subprograms) are used for vector-vector, matrix-vector and matrix-matrix operations and are tuned for a particular architecture. For more information refer to the man pages: intro_blas1, intro_blas2, and intro_blas3

LAPACK (Linear Algebra PACKage) solves systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigen value problems, and singular value problems.

The BLAS and LaPACK libraries include libGoto form the University of Texas. C programmers must use the Fortran interface to these libraries.

FFT (Fast Fourier Transforms) is package of Fortran subprograms for the fast Fourier transform of periodic and other symmetric sequences. For more information refer to the man pages: intro_fft, intro_fftw2 and intro_fftw3

ScaLAPACK (Scalable Linear Algebra PACKage) contains High-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations that support PVM and/or MPI.

BLACS (Basic Linear Algebra Communication Subprograms) is a message-passing library, designed for linear algebra. The computational model consists of a one- or two-dimensional process grid, where each process stores pieces of the vectors and matrices.

SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high-performance machines. Functions are written in C and callable from either C or Fortran. Three different versions exist for various machine architectures; Cray XT systems are distributed memory systems

UPC - Unified Parallel C
Your site determines a default module set that loads when your login shell starts. The following list actually exists on a Cray system running CLE 3.1.UP03:

```
users/rns> module list
Currently Loaded Modulefiles:

1) modules/3.2.6.6
2) nodestat/2.2-1.0301.23102.11.16.gem
3) sdb/1.0-1.0301.25351.21.29.gem
4) MySQL/5.0.64-1.0301.2899.20.4.gem
5) lustre-cray_gem_s/1.8.4_2.6...
6) udreg/2.2-1.0301.2966.16.2.gem
7) ugni/2.1-1.0301.2967.10.23.gem
8) gni-headers/2.1-1.0301.2931.19.1.gem
9) dmapp/3.0-1.0301.2968.22.24.gem
10) xpmem/0.1-2.0301.25333.20.2.gem
11) Base-opts/1.0.2-1.0301...gem
12) xtpe-network-gemini
13) pg/11.4.0
14) totalview-support/1.1.2
15) xt-totalview/8.9.0
16) xt-libsci/10.5.02
17) pmi/2.1.3-1.0000.8416.14.5.gem
18) xt-mpich2/5.2.3.2
19) xt-asyncpe/5.00.25
20) at/1.1.3
21) PrgEnv-pgi/3.1.61
22) pbs/10.4.0.101257
23) xtpe-mcl2
```

Notice the version numbers that follow the module name. Other versions may exist on your system, especially previous or pre-released versions. Consult with your system administrators to determine which modules you can swap; some modules must be used in concert.
Processor Modules

- General module additions for various processors
  - module load xtpe-barcelona
  - module load xtpe-shanghai
  - module load xtpe-istanbul
  - module load xtpe-mc8  (Magny-cours)
  - module load xtpe-mc12  (Magny-cours)
  - module load xtpe-interlagos
Chapter 6: CNL Programming Environment

Compiling Fortran

users/rns> module swap PrqEnv-cray PrqEnv-pqi
users/rns> module list
Currently Loaded Modulefiles:
   1) modules/3.2.6.6
   ...
   6) ucreg/2.3.1-1.0400.3052.6.12.gem
   7) ugni/2.2-1.0400.3340.9.29.gem
   8) gni-headers/2.1-1.0400.3411.8.1.gem
   9) dmapp/3.1-1.0400.3387.13.25.gem
  10) xpmem/0.1-2.0400.27172.6.5.gem
  11) Base-opts/1.0.2-1.0400.25719.5.1.gem
  12) xtpe-network-gemini
  13) pg4/11.7.0
  14) totalview-support/1.1.2
  15) xt-totalview/8.9.1
  16) xt-libsci/11.0.01
   ...
  19) atp/1.3.0
  20) PrqEnv-pgi/4.0.12A
  21) pbs/10.4.0.101257
  22) xtpe-mcl2
  23) xt-mpich2/5.3.2.3
users/rns> ftn -o samp264 samp264.f
users/rns> file samp264
samp264: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), for GNU/Linux
2.6.4, statically linked, not stripped

Your site may load the relevant modules during the login shell start-up; issue the command `module list` to determine what is loaded.

In earlier software releases the message:
/opt/cray/xt-asyncepe/2.5.5/bin/ftn: INFO: linux target is being used
may have appeared, this an informational message. You can suppress these messages either by including the `ftn/cc/CC option -target=[ catamount | linux]` or by setting the environment variable `XTPE_INFO_MESSAGE_OFF`. 
OpenMP

- OpenMP is a shared-memory parallel programming model that application developers can use to create and distribute work using threads
  - The Cray systems support version 3.0 or later of the OpenMP API standard
  - OpenMP provides library routines, Fortran directives, C and C++ pragmas, and environment variables
  - OpenMP applications can be used in hybrid OpenMP/MPI applications, but may not cross node boundaries
    - In OpenMP/MPI applications, MPI calls can be made from master or sequential regions, but not parallel regions
  - CCE also provides early support for OpenMP with accelerator extensions
OpenMP

- With Cray compilers, it is on by default (-h omp)
  - You can disable openmp with -h noomp; you can also use the flag -0 omp or -0 noomp

- With GCC compilers, use the -fopenmp option

- With PGI compilers, use the -mp option

- With PathScale compilers, use the -mp option
  - PathScale applications, set the PSC_OMP_AFFINITY environment variable to FALSE, the default value is TRUE
  - Note: The number of threads should not exceed the number of cores in the processor.

- With the Intel compilers, use the -openmp option

- To execute OpenMP programs:
  - Set the OMP_NUM_THREADS environment variable
  - Use the aprun -d threads option
CCE automatically chooses 128- or 256-bit SIMD width on a loop-by-loop basis based on performance projections.

PGI has command line options allowing the user to choose 128- or 256-bit SIMD width.

Although Intel compilers do support AVX generation, run-time checks in the Intel library prevent execution on Interlagos hardware. Intel compilers have no support for FMA or XOP instruction generation.
Intel Fortran does support coarrays, but their implementation does not leverage the Gemini PGAS support. Intel C does not provide UPC support.

GNU and Pathscale compilers do not provide UPC or Fortran with coarray support, although there is a third-party effort to add UPC support to GCC (typically based on GASNET).
Launching a Compute Node Program

- CLE supports both interactive and batch methods of launching a compute node program
  - Administrators can limit you to a single choice
  - Interactive - use the launcher commands
    - aprun (CNL)
  - Batch -
    - PBS Pro or Torque/MOAB
    - Submits job script via qsub and queues
    - Selects job and executes script on a login node
    - Script runs launcher command (aprun) to start the compute node program

PBS Pro also permits the scheduling of interactive jobs; refer to the TotalView debugger chapter for instructions.
Chapter 6: CNL Programming Environment

Executing Programs on CNL

- All application executables on the Cray systems are malleable (adaptable)
  - The number of processors to run on is determined at runtime
- Use the `aprun` command to run an application
  - `aprun` handles stdin (for PE 0), stdout and stderr
- Must be in a directory accessible by the compute nodes
  - Normally this is a Lustre file system
  - Can be a DVS mounted file system
    - Performance may be an issue
    - This is not a DVS issue, the exact system configuration determines performance

To successfully execute `aprun`, you must direct file operations to paths within the mount point of the file system that provides compute node application I/O.

`aprun` provides user identity and environment information, as part of the application launch, so that the user's login node session can be replicated for the application on the assigned set of compute nodes. This information includes the `aprun` current working directory that must be accessible from the compute nodes.

```
% df -t lustre
Filesystem 1K-blocks  Used  Available Use% Mounted on
43@ptl:/work/user 1664914272 60085752 127919700 2% /scratch
```
The ALPS Suite

- The Application Level Placement Scheduler (ALPS) provides the interface for compute node job placement and execution, forwards the users environment, and manages stdin (PE 0 only), stdout, and stderr

<table>
<thead>
<tr>
<th>User commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aprun</td>
<td>Application submission (job launcher)</td>
</tr>
<tr>
<td>apstat</td>
<td>Application status</td>
</tr>
<tr>
<td>apkill</td>
<td>Signal delivery</td>
</tr>
<tr>
<td>xtnodestat</td>
<td>Provides current job and node status summary information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Servers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apsys</td>
<td>Client interaction on login nodes</td>
</tr>
<tr>
<td>apinit</td>
<td>Process management on compute nodes</td>
</tr>
<tr>
<td>apsched</td>
<td>Reservations and placement, on SDB node</td>
</tr>
<tr>
<td>apbasil</td>
<td>Workload manager interface, on login or SDB node</td>
</tr>
</tbody>
</table>

Only the first three clients are of interest to users.
Running Batch Jobs

**Login node**
- `qsub`
- `shell`
- `PBS_mom`
- `apbasil`
- `aprue`
- `apsys`

**SDB node**
- `PBS_server`
- `Scheduler`
- `apsched`

**Compute node**
- `apinit`
- User application

Diagram showing the interaction between login, SDB, and compute nodes.
## ALPS Basics

- **Terminology**
  - **Node**
    - All resources managed by a single CNL instance
  - **Processing Element (PE)**
    - PEs are instances of the executable
  - **NUMA node**
    - A multi-core socket (XT5) or die (X6)
Chapter 6: CNL Programming Environment

**ALPS**

- **ALPS basic job control**
  - **Width (aprun -n)**
    - Number of PEs to launch
    - A PE is an instance of a binary copied to a node
    - Determines the number of MPI Ranks an application uses
    - The default value is 1
  
  - **Node List (aprun -L)**
    - A user-supplied list of candidate nodes to constrain placement
      - List must be specified in an increasing range

  - **To run in Multiple Program Multiple Data (MPMD) mode, a colon (:) separates programs**
    - There must be a space on both sides of the colon

An example of a MPMD job launch with two executables:

```bash
aprun [-n procs][-d depth][-N pes] executable [options] [: -n procs [-d depth] [-N pes] executable_2]
```
ALPS Multi-core Control

- PEs Per Node / PPN (aprun -N)
  - Number of PEs per CNL instance
  - Not part of the MPI rank assignment
  - When specified:
    - Places specified number of PEs per node
    - A sufficient number of cores must exist on each node
  - When unspecified:
    - Allows ALPS to pack PEs tightly
    - Behavior dependent upon application and system resources
  - ALPS assigns the same number of PEs to all nodes, regardless of whether PPN is specified. (This is required for distributed memory (DM).)
ALPS Multi-core Control

- **Depth** (`aprun -d`)
  - Specifies number of threads per PE
    - The meaning of this option depends on the programming model. Specify this option when you use OpenMP code.
    - Compute nodes must have at least `depth` cores
  - **ALPS reserves (width * depth) processor cores for the use of the application**
    - Not part of MPI rank assignment
  - **ALPS invokes width (-n) instances of the binary**
    - Application spawns `(depth - 1)` threads per PE
    - Running two threads on a core is not a good idea
### ALPS Multi-socket Control

<table>
<thead>
<tr>
<th>aprun</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-S #</td>
<td>Defines the number of PEs per NUMA node</td>
</tr>
<tr>
<td>-sl #</td>
<td>Defines the NUMA node to use: XT5 (0 or 1) X6 (0, 1, 2, or 3)</td>
</tr>
<tr>
<td>-sn #</td>
<td>Defines the number of NUMA nodes to use: XT5 (1 or 2) X6 (1 - 3)</td>
</tr>
<tr>
<td>-ss</td>
<td>Specifies strict memory containment per NUMA node</td>
</tr>
<tr>
<td>-cc</td>
<td>Defines a CPU list for binding, can also use keywords: cpu (default) binds a PE to a CPU in the NUMA node numa_node binds a PE to the CPUs within a NUMA node any threads created by the PE are bound to the same NUMA node</td>
</tr>
<tr>
<td>-cp</td>
<td>Defines a file name to use for CPU binding</td>
</tr>
</tbody>
</table>
Chapter 6: CNL Programming Environment

ALPS Memory Control

- Memory allocation is controlled with the -m option
  - The option is in Megabytes per PE

<table>
<thead>
<tr>
<th>aprun</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m size</td>
<td></td>
</tr>
<tr>
<td>-m sizeh</td>
<td>Requests size of huge pages to be allocated to each processing element. All nodes use as much huge page memory as they are able to allocate and 4 KB pages afterward.</td>
</tr>
<tr>
<td>-m sizehs</td>
<td>Requires size of huge pages to be allocated to each processing element. If aprun cannot satisfy the request, it issues an error message and the application terminates.</td>
</tr>
</tbody>
</table>

Specifies the per-processing-element required Resident Set Size (RSS) memory size in megabytes. K, M, and G suffixes are supported (16M = 16 megabytes, for example). Any truncated or full spelling of unlimited is recognized. If you do not include the -m option, the default value assigned equals the minimum compute node memory size divided by the maximum number of CPUs on any compute nodes. For example, on Cray XT5 compute nodes with 32 GB of memory, the default per-PE memory size is 32 GB / 8 CPUs = 4 GB.

- Memory requirements are per PE
- Aggregate OS memory usage is not enforced by CNL
- Restriction of one application per node due to a lack of enforceability
- Tries to adjust to systems with different node memory sizes
Using Huge Pages

- Huge pages are:
  - 2 Megabytes on a Cray XT system
  - 128KB, 512KB, 2MB, 8MB, 16MB, or 64MB on a Cray XE system
  - Default or “base” page size is 4 KB

- Using Huge pages can improve the performance of your application
  - To use huge page module and compile your application
    ```
    $ module load craype-hugepages128K (512k, 2m, 8m, 16m or 64m)
    $ cc -o my_app my_app.c
    ```
  - Add the `-m` option along with the huge page suffix (for example `-m 700h` or `-m 700hs`) and launch the application
    ```
    aprun -n 16 -N 2 -m 700hs ./hugepage_app
    ```
  - If you link (compile) your application with huge pages, run your application with the same module loaded

See the `intro_hugepages` man page for more information.
Using Huge Pages

• The memory available for huge pages is less than the total amount of memory available to the PE
  ▪ The operating system and I/O buffers reduce available memory
  ▪ Memory fragmentation can reduce available memory
    • Fragmentation usually increases with time
    • This could affect running multiple runs of the application
  – If you do not use the \texttt{--m sizes\#s} (requires) option the application may start, but perform slower
Additional aprun Options

- **aprun -B**
  - Uses some batch reservation options directly
    - aprun \(-n\) (mppwidth) \(-N\) (mppnpnn) \(-d\) (mppdepth) \(-m\) (mppmem)
    - Cannot use \(-B\) with the \(-n\), \(-N\), \(-d\), \(-m\) options on the aprun command line

- **aprun -r**
  - Core specialization
    - Allows user to designate a core per node to handle OS work
    - Must account for the extra core in the application launch, the following requires two X6 nodes

```bash
$ aprun -n 24 -r 1 a.out
```
• apcount
  - "scale up" the width of the batch reservation when using core specialization (the aprun –r option)

  apcount [-d depth] [-N PEs_per_node] [-r cores] [-v] appcount hwsise

  crayadm@xt:~> apcount -d 6 -r 1 24 12 48
  crayadm@xt:~>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d depth</td>
<td>The number of threads per processing element (PE)</td>
</tr>
<tr>
<td>-N #</td>
<td>The maximum number of PEs per node</td>
</tr>
<tr>
<td>-r cores</td>
<td>The number of cores that are specialized</td>
</tr>
<tr>
<td>-v</td>
<td>Display verbose output</td>
</tr>
<tr>
<td>appcount</td>
<td>The number of total application instances running</td>
</tr>
<tr>
<td>hwsise</td>
<td>The number of cores-per-node for the given processor type</td>
</tr>
</tbody>
</table>
In this example, C indicates the cabinet (using the conventional cabinet and row nomenclature), c indicates the cage (chassis), s indicates the slot number, and n indicates the node. There are 4 cabinets; each has 3 chassis with 8 slots per chassis. Cabinets 0 and 1 have service blades in chassis 0, slots 0-3. Note that there are only 2 nodes on a service blade, so nodes 1 and 2 do not exist.

Cabinet C3-0 has a node down at c0s4n1

This is from a heavily loaded system with a rapidly changing mixture of large and small batch jobs
Four interactive and 136 batch nodes are free.
Several nodes are allocated (A) but idle, waiting for batch jobs to run an application.
Lower case letters show individual jobs
Job h has found space by using non-contiguous nodes.
In addition to the preceding information, a list of jobs matching the code letter to the job ID, user, start time, number of nodes and command line can be produced.
ALPS Status and Control

- The `apstat` command provides status information for:
  - Applications
  - Compute nodes
  - Reservations
  - Scheduler statistics
  apstat [-a] [-n] [-p] [-r][-s] [-v] [apid [apid...]]

- ALPS `apkill` command sends a signal to a specified application
  apkill [-signal] apid
  - The `apkill` command sends a signal to all processes that are part of the specified application (apid).
    - `-signal` Specifies the name or integer of the signal to send.
      - Signal 15 (SIGTERM) is sent by default. Usually, this will kill processes that do not catch or ignore the signal.
      - See the signal(7) man page for details.

`apstat` accepts the following options:

- `-a` [apid] Displays applications.
- `-n` Displays nodes.
- `-p` Displays pending applications.
- `-r` Displays confirmed and claimed reservations.
- `-s` Displays scheduler statistics.
- `-v` Displays verbose detail (applies to all options).
## Application Status `apstat`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apstat -a</code></td>
<td>Displays the applications and not the summary</td>
</tr>
<tr>
<td><code>apstat -a apid</code></td>
<td>Displays only the specified application</td>
</tr>
<tr>
<td><code>apstat -av apid</code></td>
<td>Displays more detail about the specified application</td>
</tr>
</tbody>
</table>

```bash
% apstat
Compute node summary: up: 352 idle: 61

No pending applications are present
```

<table>
<thead>
<tr>
<th>Placed</th>
<th>Apid</th>
<th>ResId</th>
<th>User</th>
<th>PEs</th>
<th>Nodes</th>
<th>Age</th>
<th>State</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>322549</td>
<td>36</td>
<td>user1</td>
<td>512</td>
<td>256</td>
<td></td>
<td>0h19m</td>
<td>run</td>
<td>mpirch</td>
</tr>
<tr>
<td>322558</td>
<td>41</td>
<td>user2</td>
<td>64</td>
<td>32</td>
<td></td>
<td>0h03m</td>
<td>run</td>
<td>echam5</td>
</tr>
<tr>
<td>322560</td>
<td>42</td>
<td>user3</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0h01m</td>
<td>run</td>
<td>OpsProg_Ex</td>
</tr>
<tr>
<td>322563</td>
<td>44</td>
<td>xtpo0</td>
<td>4</td>
<td>2</td>
<td></td>
<td>0h00m</td>
<td>run</td>
<td>OMP_where</td>
</tr>
</tbody>
</table>

The `apstat -a` command displays the following column headings for placed applications:

- **Apid**: Application id
- **User**: Application owner
- **PEs**: Application total number of PEs
- **Nodes**: Total number of nodes assigned to this application
- **Age**: Amount of time since the placement was made and the resource reservation was claimed
- **Command**: Application name
When a batch job is executed, an ALPS reservation is made (confirmed). When the batch job's embedded aprun is executed, the reservation is claimed and the line with the 'A' flag shows the allocation. The number of PEs in the allocation must be less than or equal to those of the reservation.

```
users/rns> apstat
Compute node summary
          arch  config  up  use  held  avail  down
          XT     140  140   1    0   147    0

No pending applications are present

Total placed applications: 1
Placed  ApId  ResId  User  PEs  Nodes  Age  State  Command
      1479335 1049  rns    4    1  0h04m  run  sfreduce

users/rns> apstat -r
ResId  ApId  From  Arch  PEs  N  d  Memory  State
    1049  1479334  batch:1879368  XT  4  4  1  2000  NID list, conf, claim
A  1049  1479335  batch:1879368  XT  4  --  2000  conf, claim

users/rns> qstat
Job id            Name             User           Time Use S  Queue
------------- ---------------- -------------- -------- - ----- -----
1023356.sdb      STDIN            user1          00:00:00 R  workq
1023370.sdb      STDIN            user2          00:01:03 R  workq
1023379.sdb      run_64           user3          00:00:03 R  workq
1023397.sdb      signal_check user4          00:00:00 R  workq
users/rns>
```

When a batch job is executed, an ALPS reservation is made (confirmed). When the batch job's embedded aprun is executed, the reservation is claimed and the line with the 'A' flag shows the allocation. The number of PEs in the allocation must be less than or equal to those of the reservation.

```
users/rns> qstat
Job id            Name             User           Time Use S  Queue
------------- ---------------- -------------- -------- - ----- -----
1879368.sdb      sfreduce.pbs     rns          00:00:00 R  qnodes
```

```
users/rns>
```
The `apstat -n` command displays the following column headings for node information:

- **NID**: Compute node identifier
- **Arch**: Compute node architecture (XT or X2)
- **State**: Compute node state; includes both status (UP or DN) and allocation mode (B for batch or I for interactive)
- **HW**: The number of cores in the node
- **Rv**: The number of cores held in a reservation
- **Pl**: The number of cores being used by an application
- **PgSz**: Compute node memory page size in bytes. PgSz is 4K for 4 KB base pages and 2M for 2 MB huge pages
- **Avl**: Compute node pages available
- **Conf**: Total pages confirmed
- **Placed**: Total pages placed
- **PEs**: Number of PEs reserved on the node
- **Apids**: Placed application IDs (or blank if none placed)
Using `cnselect` to Select Nodes

- Convenient MySQL interface to attributes table
- Returns a list of compute nodes based on user-specified criteria
- Must be run from a login node
- Pass node list to `aprun` via the `-L` parameter. Nodes used will be selected from the list but only as many as needed will be reserved.

```
% module load MySQL
% NODES=$(cnselect availmem.lt.2000 .and. coremask.eq.1)
% echo $NODES
44-55
% export OMP_NUM_THREADS=1
% aprun -n 2 -d 1 -L $NODES ./OMP_where
Rank = 0  Thread = 0  Processor = nid00044
Rank = 1  Thread = 0  Processor = nid00045
```

The first example selects any available node. The second two select single- or dual-core nodes. The next one select s clock speed. The remaining ones select on memory size; the final one also chooses the number of cores.

```
% module load MySQL
% cnselect
44-63
% cnselect coremask.eq.1
44-55
% cnselect coremask.gt.1
56-63
% cnselect clockmhz.ge.2400
56-63
% cnselect availmem.lt.2000
44-55
% cnselect availmem.eq.2000
44-55
% cnselect availmem.gt.2000
56-59
% cnselect availmem.gt.2000
56-59,60-63
% cnselect availmem.lt.3000.and.coremask.eq.1
44-55
```
Batch Access

- Cray systems use three batch systems: PBS Pro, Torque/MOAB, and LSF
  - LSF is not discussed in this course
  - To use PBS, you must load the `pbs` module
    ```bash
    % module load pbs
    ```
  - To use Torque/MOAB, you must load the `moab` module
    ```bash
    % module load moab
    ```
  - PBS Pro and Torque/MOAB use these commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qsub</td>
<td>Submits your job script</td>
</tr>
<tr>
<td>qstat</td>
<td>Monitors job queues</td>
</tr>
<tr>
<td>qstat -a</td>
<td>Enhanced qstat</td>
</tr>
<tr>
<td>qstat -f jobid</td>
<td>Provides information on a job</td>
</tr>
<tr>
<td>qdel</td>
<td>Delete (kill) your queued or running job</td>
</tr>
</tbody>
</table>
Chapter 6: CNL Programming Environment

Simple PBS Job Script

```bash
#!/bin/sh
#PBS -l mppwidth=48
#PBS -l mppdepth=6
#PBS -l mppnppn=4
#PBS -N jobname
#PBS -j oe
# Change to the directory where job was submitted
cd $PBS_O_WORKDIR
aprun -n 48 -d 6 -N 4 program [options]

On a CLE 3.1 System

aprun -B program [options]
```

PBS Script Options, refer to the qsub man page for details.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PBS_O_WORKDIR</td>
<td>Identifies the directory you were in when you submitted the job, this must be a Lustre file system or a file system the compute nodes have access to.</td>
</tr>
<tr>
<td>-a date-time</td>
<td>Time at which a job becomes eligible to run</td>
</tr>
<tr>
<td>-A account</td>
<td>Run job under the specified account</td>
</tr>
<tr>
<td>-j oe</td>
<td>eo</td>
</tr>
<tr>
<td>-l</td>
<td>Specifies a list of resources required for the job</td>
</tr>
<tr>
<td>-N</td>
<td>Specifies a name for the job, the default is the name of the script</td>
</tr>
<tr>
<td>-o</td>
<td>e path</td>
</tr>
<tr>
<td>-p priority</td>
<td>Priority in the queue relative to your other jobs; this is only a hint</td>
</tr>
<tr>
<td>-q queue</td>
<td>Destination queue</td>
</tr>
<tr>
<td>-v [list]</td>
<td>Pass environment variables to job</td>
</tr>
<tr>
<td>-V</td>
<td>Pass all environment variables to job</td>
</tr>
<tr>
<td>:</td>
<td>When used at the beginning of a line the line is ignored by PBS</td>
</tr>
</tbody>
</table>
## PBS Pro MPP Resources

Specialized PBS resources map to aprun parameters

<table>
<thead>
<tr>
<th>aprun</th>
<th>qsub</th>
<th>Resource Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n 4</td>
<td>-l mppwidth=4</td>
<td>Width</td>
</tr>
<tr>
<td>-d 2</td>
<td>-l mppdepth=2</td>
<td>Depth</td>
</tr>
<tr>
<td>-N 1</td>
<td>-l mppnppn=1</td>
<td>Number of PEs Per Node</td>
</tr>
<tr>
<td>-L 5,6,7</td>
<td>-l mppnodes=&quot;5,6,7&quot;</td>
<td>Node List</td>
</tr>
<tr>
<td>-m 1000m</td>
<td>-l mppmem=1000mb</td>
<td>Memory Per PE</td>
</tr>
<tr>
<td>-a xt</td>
<td>-l mpparch=XT</td>
<td>Target Architecture</td>
</tr>
<tr>
<td></td>
<td>-l mpplabels=&quot;foo,bar&quot;</td>
<td>Node Attribute Labels</td>
</tr>
</tbody>
</table>
Chapter 6: CNL Programming Environment

**Batch Job Submission**

- `qsub` returns a Job ID of `258909.nid00003`
- Use `qstat -a` to show the status of job `258909`
- Default filename for `stdout` is `runjob.o258909`
  - Script contains `-oe` option so only one output file

```bash
% qsub runjob
258909.nid00003
% ls -l *258909
-rw------- 1 rns hwpt 460 Nov 21 12:01 runjob.o258909
%
```

- Submitting an interactive batch job

```bash
% qsub -I -l mppwidth=4 -l mppdepth=2 -l mppnppn=1
qsub: waiting for job 148535.sdb to start
qsub: job 148535.sdb ready
%
```

See the next page for the `qstat` output.

The following is an example of running an interactive batch job:

```bash
% qsub -I -l mppwidth=4 -l mppdepth=2 -l mppnppn=1
qsub: waiting for job 148535.sdb to start
qsub: job 148535.sdb ready
%
% export OMP_NUM_THREADS=2
% aprun -n 4 -N 1 -d 2 -m7G ./OMP_where
rank = 0 thread = 0 processor = nid00384
rank = 0 thread = 1 processor = nid00384
rank = 1 thread = 0 processor = nid00385
rank = 2 thread = 0 processor = nid00386
rank = 3 thread = 0 processor = nid00387
rank = 3 thread = 1 processor = nid00387
rank = 2 thread = 1 processor = nid00386
rank = 1 thread = 1 processor = nid00385
Application 323280 resources: utime 0, stime 0
```
If no output is returned from a job, check `/var/spool/PBS/undelivered/`. 

---

### Batch Job Status

```
% qstat -a
nid00003:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>Queue</th>
<th>Nodes</th>
<th>Time</th>
<th>State</th>
<th>Req'd</th>
<th>Req'd</th>
<th>Flap</th>
</tr>
</thead>
<tbody>
<tr>
<td>258908</td>
<td>user</td>
<td>workq</td>
<td>runjob</td>
<td>18359</td>
<td>000:00</td>
<td>4</td>
<td>--</td>
<td>R</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>258909</td>
<td>user</td>
<td>workq</td>
<td>runjob</td>
<td>18389</td>
<td>000:00</td>
<td>4</td>
<td>--</td>
<td>R</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Total generic compute nodes allocated: 8

**Legend:**
- E - Job is exiting after having run
- H - Job is held
- Q - Job is queued, eligible to run or routed
- R - Job is running
- T - Job is being moved to new location
- W - Job is waiting for its execution time (-a option) to be reached
- S - Job is suspended
Torque/MOAB

- Torque/MOAB is an alternate work load manager
  - Developed and maintained by Cluster Resources, Inc.

- MOAB Terminology
  - Backfill – allows jobs to run out-of-order
    - Provided they do not impact high-priority jobs
    - Criteria for backfill is wall clock time
      - Users must provide good estimates of their time requirements
        - If too small, a job may be killed before completion
        - If too large, a job may not be scheduled
  - Allocation management – allowing portions of the system to be assigned to a given account
    - Associates users with accounts
    - User specifies the account to use at job submission
MOAB Terminology

• Advanced reservations – reserves specific resources for a given period of time
  ▪ Access is controlled by an access control list (ACL)
  ▪ By default, does not constrain a user to the reserved resources
  ▪ Can be changed by the administrator

• Quality of service (QOS) – enables administrators to grant special privileges to particular users
  ▪ Special privileges may include:
    • Access to additional resources
    • Exemptions from specific policies
    • Access to special capabilities
    • Improved job prioritization
  ▪ Users must specify the QOS in the job submission script
## MOAB Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| **checkjob** | Provides users with a detailed status of each job they submitted  
  • Also provides an analysis of why the job is not running |
| **showstart** | Provides an estimated start time  
  • Time is more accurate if you are near the top of the list of jobs to execute  
  • Does not account for higher-priority jobs being submitted after you run the command |
| **showq** | Provides additional information about job prioritization  
  • Supplements what the `gstat` command provides |
| **showbf** | Identifies the resources that are immediately available |
Debugging Tools

Customer Documentation and Training
Cray Debugging Support Tools

- STAT (Stack Trace Analysis Tool)
- ATP (Abnormal Termination Processing)
  - MRNet (Multicast/Reduction Network)
- FTD (Fast Track Debugging)
  - Supported in l gdb and DDT
Stack Trace Analysis Tool (STAT)

- Stack trace sampling and analysis for large scale applications
  - Reduce number of tasks to debug
  - Discover equivalent process behavior
- Sample application stack traces
  - Merge/analyze traces:
    - Discover equivalent process behavior
    - Group similar processes
    - Facilitate scalable analysis/data presentation
- Merging Stack Traces
  - Create call graph prefix tree
    - Compressed representation
    - Scalable visualization
    - Scalable analysis
Abnormal Termination Processing (ATP)

- System of light weight back-end monitor processes on compute nodes
  - Coupled together as a tree with MRNet
  - Automatically launched by aprun
  - Leap into action on any application process trapping
  - stderr backtrace of first process to trap

- STAT like analysis provides merged stack backtrace tree
  - Leaf nodes of tree define a modest set of processes to core dump
  - Or, a set of processes to attach to with a debugger
ATP 1.2.1

• Automatic
  – ATP module loaded by default
    ▪ Signal handler added to application and registered
  – aprun launches ATP in parallel with application launch
    ▪ In CLE 2.2 a user must overtly launch
  – Enabled/disabled via ATP_ENABLED environment variable

• Provides:
  – Backtrace of first crash to stderr
  – Merged backtrace tree
  – Dumps core file set (if limit/ulimit allows)

• Tested at 10K PEs
Fast Track Debugging

- Compile an application such that both a debug and non-debug (optimized) version of each routine is created
  - Linkage such that optimized versions are used by default
  - Debugger overrides default linkage when setting breakpoints and stepping into functions
- Currently prototyped in gdb and exercised through lgdb and is used with Allinea’s DDT
Cray delivers an integrated set of performance tools that provide automatic program instrumentation, without requiring source code or file modifications. Before you can use these tools, ensure that your code compiles cleanly, runs to completion, and produces expected results.
CrayPat is a performance analysis tool that collects performance information from a user application:

- CrayPat supports two types of experiments: sampling and tracing
  - Sampling experiments capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - Tracing counts an event, such as the number of times an MPI call is executed
- CrayPat uses PAPI to read the performance counters of the Opteron processor

Cray Apprentice2 displays graphical reports from the .ap2 file.

Trace-based or synchronous experiments count every entry into and out of each function that is called in the application. Build (pat_build) options can reduce the number of functions to include in the experiment. Further experimentation on a fine-grained portion of the application can occur through source code modifications, where a user uses CrayPat pat_region API in the source code. Normally this is not required.
**pat_build Sampling**

- The default mode of instrumentation with no options on the `pat_build` command line is **sampling**.
  - Sampling is controlled by the environment variable `PAT_RT_EXPERIMENT`
    - Supported sampling functions are: `samp_pc_time`, `samp_pc_ovfl`, `samp_cs_time`, `samp_cs_ovfl`, `samp_ru_time`, `samp_ru_ovfl`, `samp_heap_time`, `samp_heap_ovfl`
    - **Caution**: Do not collect hardware counter information when you sample by overflow (for example `< samp_pc_ovfl`)
  - Use **sampling** to obtain a profile and **then trace** functions of interest
Chapter 8: Performance Tools

Using CrayPat

- To instrument a program:
  - Load the **perftools** module
    
    ```
    % module load perftools
    ```
  - The executable and object (.o) files are required
    ```
    % ftn -c prog.f90
    % cc -c work.c
    % ftn -o program1 prog.o work.o
    ```
  - Or
    ```
    nid00008/rns> ftn -o samp264 samp264.f
    /opt/cray/xt-asyncpe/3.4.4/bin/ftn: INFO: linux target is being used
    WARNING: CrayPat is saving object files from temporary locations
    into directory '/home/users/rns/.craypat/samp264/15976'
    nid00008/rns>
    ```

In the example above, `%pat_build program1` examines the program `program1` and relinks its object and library files with files from the CrayPat run-time library to produce `program1+pat`. This operation requires the continued availability of the object files that were used to link `program1` (either in their locations at the time `program1` was linked or in a directory specified by the `PAT_BUILD_LINK_DIR` environment variable).
Using CrayPat

- **Run `pat_build` to instrument the program**

  ```bash
  nid00008/rns> pat_build samp264
  nid00008/rns> ls -l samp26*
  
  -rw-r-xr-x 1 rns hwpt 2001593 Sep 24 19:21 samp264
  -rw-r-xr-x 1 rns hwpt 3592486 Sep 24 19:22 samp264+pat
  ```

- **Execute the instrumented program**

  ```bash
  nid00008/rns> aprun -n 4 samp264+pat
  ```
Automatic Profiling Analysis (APA)

- The instrumented program generates a .xf file
  - Depending on the size of the job and environmental variables:
    - either a single .xf file (default) is created
    - or a subdirectory with a .xf file for each processor used
  - Run `pat_report`
    - `pat_report` will generate an .ap2 and .apa file, as well as run a text report to stdout
    - The .ap2 is used to generate additional text reports or is used by Apprentice2
    - The .apa file is used (optionally) to assist you in creating a trace based experiment file

```
nid00008/rns> pat_report samp264+pat+2885-203sdt.xf

nid00008/rns> ls -l samp264+*
  -rw------- 1 rns hwpt  444 Sep 24 19:25 samp264.pbs.o1079481
  -rw-r----- 1 rns hwpt  7240 Sep 24 19:25 samp264+pat+2885-203sdt.xf
  -rw-r--r-- 1 rns hwpt  1568 Sep 24 19:26 samp264+pat+2885-203sdt.apa
  -rw-r--r-- 1 rns hwpt  36864 Sep 24 19:26 samp264+pat+2885-203sdt.ap2
```

By default, for jobs with 255 PEs or less, a single .xf file is created. If the job uses 256 PEs or more, the square root number of PEs .xf files are created.

The user had to instrument their program with `pat_build --O apa` in order for `pat_report` to generate the .apa file.
Chapter 8: Performance Tools

Automatic Profiling Analysis (APA)

- Use the .apa file to build a trace experiment file
  - No need to specify the executable
  - You should get an instrumented program samp264+apa

```
nid00008/rns> pat build -O samp264+pat+2885-203sdt.apa
INFO: Trace intercept routine created for the 883 byte
function 'use data '

nid00008/rns> ls -ltr
...
-rwxr-xr-x 1 rns hwpt 3599891 Sep 24 19:49 samp264+apa
```

- Run application to get top time-consuming routines
  \% aprun -n 4 samp264+apa

- Use pat_report to process the .xf file
- The .apa file can be modified and used again by you

The top time-consuming routines comes from the initial pat_build -O apa, which performs a form of sampling to get an initial profile. Then further information can be obtained for those top time consuming routines (identified in the .apa file) with the program instrumented using the .apa, and rerun.

Use pat_report to process the .xf file, not view the .xf file. View the text report generated to stdout or through Apprentice2.
**pat_build Trace Options**

- To trace functions and create the instrumented executable, use the following `pat_build` options:
  - `-g` traces non-user library functions for one of the predefined groups, like `[caf|cuda|gni|...|upc]`
    - Refer to the `pat_build` man page for a complete list.
  - `-t` `tracefile` to specify a file containing a lists of functions to trace.
  - `-T` `tracefunc` where `tracefunc` is a comma-separated list of function names to trace; `!tracefunc` excludes function.
  - `-u` trace user functions.
  - `-w` is used to trace MAIN. There are only trace points to collect performance data inserted at the beginning and end of MAIN.
    - This is helpful if the user wants to collect some data that has high collection overhead and wants to minimize additional tracing overhead.

You can specify the name of resulting instrumented program with the `-o` option or by the final argument. If neither are specified, the program name is appended with `+pat`.

- `-f` overwrites an existing instrumented binary with a `+pat` suffix.

*Note: pat_build does not enable you to instrument a program that is also using the PAPI interface directly.*
The experiment output file (or data file) is:
- A single .xf file (default)
  - Requires that output be written to a file system that supports locking (such as a Lustre file system)
- A directory with a file (.xf) for each process
- The file named reduce+pat+3496-12t.xf contains the following information: name of the instrumented program, reduce+pat; the process ID 3496; the physical node—the application started on 12; and the type of experiment performed

- The pat_report command reads the experiment file(s) and produces a text report and an .ap2 file
- The .ap2 file is used as input to Apprentice
- The .ap2 file can be used by pat_report to produce text output
  - The .ap2 file is portable; it does not require the source or .xf files
# Environment Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| PAT_RT_SUMMARY               | 0 Turn off summary  
1 Enable summary (default) |
| PAT_RT_EXPFILE_PER_PROCESS   | 0 Write experiment data to a single file  
Requires a file system capable of locking  
1 Write a separate file for each process  
   • An application may abort if the number of processes exceeds the number of open files permitted |
| PAT_RT_EXPFILE_NAME          | The experiment file name                                                     |
| PAT_RT_EXPFILE_DIR           | The directory that contains the experiment output file  
   • Specify a Lustre directory when you create a single experiment output file |
| PAT_RT_HWPC                  | Define the HWPC group                                                        |
A Sequence of Commands

```
# Loaded the CrayPat module
rns/crayPatExample> module load perftools

# compiled the code - simple application
rns/crayPatExample> ftn -o samp264 samp264.f

# Created the experiment executable
rns/crayPatExample> p4t_build samp264

# modify the job script to run samp64+pat
rns/crayPatExample> vi samp264.pbs

# run the job
rns/crayPatExample> qsub samp264.pbs

# Made sure the job ran
rns/crayPatExample> cat samp264.pbs.o1879623

# view samp264+pat+15346-43s.xf > samp264+pat+15346.report
rns/crayPatExample> view samp264+pat+15346.report

# run the job
rns/crayPatExample> ls -ltr

-rw-r-x-x-- 1 rns hwpt  5411 Sep 25 13:34 samp264.f
-rw-r--r-- 1 rns hwpt  306 Sep 25 13:34 samp264.pbs
-rw-r--r-- 1 rns hwpt  990967  Sep 25 13:35 samp264
-rw-r-xr-x  1 rns hwpt 3592502 Sep 25 13:35 samp264+pat
-rw-r-----  1 rns hwpt   459 Sep 25 13:36 samp264.pbs.o1879623
-rw-r-----  1 rns hwpt   7240 Sep 25 13:36 samp264+pat+15346-43s.xf
-rw-r-----  1 rns hwpt   5248 Sep 25 13:37 samp264+pat+15346.report
-rw-r-----  1 rns hwpt   1613 Sep 25 13:37 samp264+pat+15346-43s.apa
-rw-r-----  1 rns hwpt  36868 Sep 25 13:37 samp264+pat+15346-43s.ap2
-rw-r-xr-x  1 rns hwpt 3599971 Sep 25 13:53 samp264+apa

# modify the job script to run samp64+apa
rns/crayPatExample> vi samp264.pbs

# run the job
rns/crayPatExample> qsub samp264.pbs

# run the job
rns/crayPatExample> p4t_report samp264+apa+8557-142t.xf > samp264+apa+8557.report

rns/crayPatExample> vi samp264+apa+8557.report
```
The table is a portion of the output of `program1.rpt1`.

The fifth column, labelled **Calls**, contains the count for all 4 PEs.

The second column, **Time**, lists the maximum time used by any PE per function.

The third column, **Imb. (Imbalance) Time**, lists the average time required by all PEs per function.

The fourth column, **"Imb. Time %**, a value of 100% indicates that a single PE executed the function. A value of 0% would indicate that all PEs spent equal time performing the function. (Refer to the man page for information about the math used to calculate the percentage.)
Table 2: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Group / Function / PE='HIDE'</th>
<th>Time%</th>
<th>Time</th>
<th>Imb.Time</th>
<th>Imb.Time%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totals for program (OMITTED BY INSTRUCTOR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER (OMITTED BY INSTRUCTOR)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

USER / use_data

<table>
<thead>
<tr>
<th>Calls</th>
<th>2.0 /sec</th>
<th>30.0 calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAPI_L1_DCM</td>
<td>12.355M/sec</td>
<td>182917104 misses</td>
</tr>
<tr>
<td>FAPI_TLB_DM</td>
<td>5.849M/sec</td>
<td>86592140 misses</td>
</tr>
<tr>
<td>FAPI_L1_DCA</td>
<td>208.397M/sec</td>
<td>3085410313 ref</td>
</tr>
<tr>
<td>FAPI_FP_OPS</td>
<td>145.973M/sec</td>
<td>2161200000 ops</td>
</tr>
</tbody>
</table>

User time (approx) 14.805 secs 34052604860 cycles 100.0%Time

Average time per call: 0.493511 sec

CrayPat Overhead : Time 0.0%

HW FP Ops / User time 145.973M/sec 2161200000 ops 1.6%peak(DP)

HW FP Ops / WCT 145.973M/sec

Computational intensity 0.06 ops/cycle 0.70 ops/ref

MLOPS (aggregate) 583.89M/sec

TLB utilization 35.63 refs/miss 0.070 avg uses

DI cache hit,miss ratios 94.1% hits 5.9% misses

DI cache utilization (misses) 16.87 refs/miss 2.108 avg hits

This is the report generated after `pat_build -o samp264+apa+153&6-43&4r samp apa` was executed and the executable `samp264+apa` was run. The APA file suggested HWPC value 1 be used. This is where the performance counter data comes from Table 2.

---

TR-XEP_BW

Cray Private

159
Hardware Performance Counters

- The APA file suggests which hardware performance counters you should use
  - To use different performance counters, set the PAT_RT_HWPC ENVIRONMENTAL variable and rerun the job.

```
[rns/crayPatExample] cat samp264+pat15346 43sdt.apa
[clipped]
# ------------------------------
#   HWPC group to collect by default.
-Drunenv=PAT_RT_HWPC=1 # Summary with TLB metrics.
# ------------------------------

[rns/crayPatExample] cat samp264.pbs
#!/bin/ksh
PBS -j oe
PBS -l mppwidth=4
PBS -l walltime=00:30:00
export PAT_RT_HWPC=0
cd $PBS_O_WORKDIR
#aprun -n 4 ./samp264
#aprun -n 4 ./samp264+pat
aprun -n 4 ./samp264+apa
```

An event set is a group of PAPI preset or native events

- CrayPat defines 20 groups (sets)
  - Select a set by using the environment variable
    PAT_RT_HWPC

Profiling - counting specified events

- Used in CrayPat

Overflow - testing events and alerting the application when a count is exceeded

- Requires modification of the user application
Looking Closer

- Load the `perf tools` modules
  - Use the CrayPat `pat_region` API to identify the region of interest
    - In Fortran
      ```fortran
      include <pat_apif.h>
      call PAT_region_begin(1,"halo_loop",stat);
      ...
      call PAT_region_end(1,stat);
      ```
  - Compile your code
    - Use `pat_build` to create an instrumented binary
    - Use the environment variable `PAT_RT_HWPC` to select the hardware counters that you want to collect. `PAT_RT_HWPC=0`
    - You can also save your favorite counters in a file and pass them to CrayPat
      - Add file name to `PAT_RT_HWPC_FILE` environment variable

In C/C++
```
#include <pat_api.h>

PAT_region_begin(1,"halo_loop");
...
PAT_region_end(1);
```
Looking Closer

```
! first find the mean
! (walk thru memory as sequentially as possible)
call PAT_region_begin(1, "halo_loop", istat)
total = 0.0
do k = 1, nz
  do j = 1, ny
    do i = 1, nx
      total = total + array(i, j, k)
    enddo
  enddo
enddo
call PAT_region_end(1, istat)
```

Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time %</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>16.066011</td>
<td>--</td>
<td>--</td>
<td>65.0</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>16.065887</td>
<td>--</td>
<td>--</td>
<td>63.0</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>87.9%</td>
<td>14.121553</td>
<td>0.012971</td>
<td>0.1%</td>
<td>30.0</td>
<td>use_data</td>
<td></td>
</tr>
<tr>
<td>8.6%</td>
<td>1.005652</td>
<td>0.001772</td>
<td>0.2%</td>
<td>30.0</td>
<td>#1.Halo_loop</td>
<td></td>
</tr>
<tr>
<td>3.5%</td>
<td>0.558584</td>
<td>0.011731</td>
<td>2.7%</td>
<td>1.0</td>
<td>MAIN_</td>
<td></td>
</tr>
</tbody>
</table>
## Looking closer

### USER / #1.halo_loop

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Time</td>
<td>1.385652 s</td>
</tr>
<tr>
<td>Imb.Time</td>
<td>0.001772 s</td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>0.2%</td>
</tr>
<tr>
<td>Calls /sec</td>
<td>21.6</td>
</tr>
<tr>
<td>PAPI_L1_DCM /sec</td>
<td>0.791 M</td>
</tr>
<tr>
<td>PAPI_TOT_INS /sec</td>
<td>1969.6 M</td>
</tr>
<tr>
<td>PAPI_L1_DCA /sec</td>
<td>923.2 M</td>
</tr>
<tr>
<td>PAPI_FP_OPS /sec</td>
<td>130.8 M</td>
</tr>
<tr>
<td>User time (approx) /sec</td>
<td>1.386</td>
</tr>
<tr>
<td>100.0%Time</td>
<td></td>
</tr>
<tr>
<td>Average Time per Call /sec</td>
<td>0.046188 s</td>
</tr>
<tr>
<td>CrayPat Overhead Time /sec</td>
<td>0.0%</td>
</tr>
<tr>
<td>HW FP Ops / User time /sec</td>
<td>130.8 M</td>
</tr>
<tr>
<td>HW FP Ops / WCT /sec</td>
<td></td>
</tr>
<tr>
<td>HW FP Ops / Inst</td>
<td>6.6%</td>
</tr>
<tr>
<td>Computational intensity /sec</td>
<td>0.06 ops/cycle</td>
</tr>
<tr>
<td>Instr per cycle</td>
<td>0.86 inst/cycle</td>
</tr>
<tr>
<td>MFLOPS /sec</td>
<td>7678.77 M</td>
</tr>
<tr>
<td>MFLOPS (aggregate) /sec</td>
<td>523.03 M</td>
</tr>
<tr>
<td>Instructions per LD &amp; ST /sec</td>
<td>46.9% refs</td>
</tr>
<tr>
<td>DL cache hit,miss ratios</td>
<td>99.9% hits</td>
</tr>
<tr>
<td>DL cache utilization (misses)</td>
<td>1167.9 refs/miss</td>
</tr>
<tr>
<td>DL cache utilization (avg hits)</td>
<td>145.973 avg hits</td>
</tr>
</tbody>
</table>
PAPI

- PAPI provides a common interface for the performance counters in various processors, including the Opteron
  - PAPI defines a set of *Preset* counters that map to a common performance counter in various processors
  - The *Preset* name matches as closely as possible to the *Native* event
    - Using the *Preset* name provides portability between processors when user code is modified to collect performance data
  - A *Native* event is an actual hardware counter in the processor
    - See the PAPI documentation available at: http://icl.cs.utk.edu/papi/docs/
    - `papi_avail`, and `papi_native_avail` are commands that can be executed on the compute node to determine the available counters
      aprun -n 1 papi_avail
Rank Order and CrayPAT

- One can also use the CrayPat performance measurement tools to generate a suggested custom ordering.
  - Available if MPI functions traced (-g mpi or -0 apa)
    - `pat_build -O apa my_program`
    - see Examples section of `pat_build` man page
  - `pat_report` options:
    - `mpi_sm_rank_order`
      - Uses message data from tracing MPI to generate suggested MPI rank order. Requires the program to be instrumented using the `pat_build -g mpi` option.
    - `mpi_rank_order`
      - Uses time in user functions, or alternatively, any other metric specified by using the `-s mro_metric` options, to generate suggested MPI rank order.
Rank Order and CrayPAT

- module load perftools

- Rebuild your code
  - pat_build -O apa a.out
  - Run a.out+pat
  - pat_report -Ompi_sm_rank_order
    a.out+pat+...sdt/ > pat.report
    - Creates MPICH_RANK_REORDER_METHOD.x file
  - Then set environment variable
    MPICH_RANK_REORDER_METHOD=3 and link the file
    MPICH_RANK_ORDER.x to MPICH_RANK_ORDER
  - Rerun code
Rank Order and CrayPAT Example

<table>
<thead>
<tr>
<th>Rank Order</th>
<th>USER Samp</th>
<th>Max</th>
<th>Avg</th>
<th>Max Node Ranks</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>17062</td>
<td>97.6%</td>
<td>16907</td>
<td>100.0%</td>
</tr>
<tr>
<td>2</td>
<td>17213</td>
<td>98.4%</td>
<td>16907</td>
<td>100.0%</td>
</tr>
<tr>
<td>0</td>
<td>17282</td>
<td>98.8%</td>
<td>16907</td>
<td>100.0%</td>
</tr>
<tr>
<td>1</td>
<td>17489</td>
<td>100.0%</td>
<td>16907</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

- This suggests that:
  - The custom ordering “d” might be the best
  - Folded-rank next best
  - Round-robin 3rd best
  - Default ordering last

- The utility grid_order can be used to statically generate MPI rank order
Gemini Counters

- Gemini counters are accessed through CrayPat and environment variables
  - See the `intro_craypat` and `nwhc` man pages
  - See: `$CRAYPAT_ROOT/share/CounterGroups.Gemini`

<table>
<thead>
<tr>
<th>PAT_RT_NWPC</th>
<th>Specifies individual Gemini performance counter event names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAT_RT_NWPC_CONTROL</td>
<td>Specifies parameters that control various aspects of the Gemini networking performance counters.</td>
</tr>
<tr>
<td>PAT_RT_NWPC_FILE</td>
<td>Specifies a file or list of files containing individual Gemini performance counter event names.</td>
</tr>
<tr>
<td>PAT_RT_NWPC_FILE_GROUP</td>
<td>Specifies a file or list of files containing specifications of Gemini performance counter groups.</td>
</tr>
<tr>
<td>PAT_RT_NWPC_FILE_TILE</td>
<td>Specifies a file or list unset of files containing specifications of Gemini performance counters that use the filtering counters to define new events.</td>
</tr>
<tr>
<td>PAT_RT_NWPC_TILE_DISPLAY</td>
<td>If set to nonzero value, writes the filtered tile NWPC event specifications to stdout.</td>
</tr>
</tbody>
</table>
The left screen appears during data collection; later, the pie charts appear.
Apprentice2 call tree display
Chapter 8: Single-node Optimization

Single-node Optimization

Customer Documentation and Training
Chapter 8: Single-node Optimization

Single CPU Performance: Vectorization

- If you want your application to perform well, vectorize the code
  - Strongly recommended for single precision arithmetic
- What vectorizes
  - Inner stride-1 loops
    - Problems may occur with large body loops
      - Mvect=nosizelimit
- What does not vectorize
  - Loops that have if statements or any indirect or non-unit stride references
  - Outer loops

“The single most important impediment to good parallel performance is still poor single-node performance.”

-William Gropp
Argonne National Laboratory
Chapter 8: Single-node Optimization

Vectorization

- The vectorizer scans code searching for loops that are candidates for high-level transformations, such as
  - Loop distribution
  - Loop interchange
  - Cache tiling
  - Idiom recognition
    - Replacement of recognizable code sequences, such as a reduction loop, with an optimized code sequence or function call
- In addition, the vectorizer produces extensive data dependence information for use by other phases of compilation and detects opportunities to use vector or packed Streaming SIMD Extensions
  - In 32-bit mode, the vector length is 4
  - In 64-bit mode, the vector length is 2

SSE means "Streaming SIMD Extensions" that enhance the x86 instruction set

SSE2 – for single/dual-core processors
  - SSE2 enhancements include:
    - 8 new 128-bit SIMD floating-point registers that are directly addressable (XMM8-15)
    - 50 new instructions for packed floating-point data
    - MMX instructions for multimedia, video, mpeg

SSE3 – for Revision E processors (dual-core)

SSE4a – for quad-core processors
The write path to memory in the quad-core processors has write combining registers. The Opteron processor has four 64-byte registers that combine data from different operations (being written to the same 64-byte line) to a single write operation.
Processor Enhancements

- Floating-point is 128 bits wide
  - 1 cycle per vector
    - Previous processors consumed two cycles per vector
- Out-of-order loads, ordered retirement
- Can do 2 128-bit loads or 2 64-bit stores
- Other improvements
  - Improved performance of shuffle instructions
  - Improved transfers between floating-point and general purpose registers
  - Improved transfers between floating-point registers
  - Optimization of repeated move instructions
  - More efficient PUSH/POP stack operations
  - Adaptive prefetch, automatically advances the fetch ahead distance if the demand stream catches up to the prefetch stream
  - Write combining buffers improve performance


Note: AMD Web pages change often and the link may not be accurate. Use the publication number for searching; it may take a bit of work to find the manual.
Processor Information

- Stride-1 memory access patterns are important
  - Stride-1 uses all of the cache line before it can be evicted
  - The compiler often performs prefetch for stride-1 loops, not as often for non stride-1 loops
- It is advisable to align hot loops on 32-byte blocks
- PREFETCH versus PREFETCHW
  - PREFETCHW is recommended if you expect to modify the data
  - PREFETCHW sets hints of intent to write
- Use streaming instructions instead of PREFETCHW
  - If the code will overwrite one or more complete lines with new data
  - If the new data will not be used again
- Maximum of eight outstanding cache-misses
  - References to the same cache line are merged
- Align floating point data on 16-byte boundaries
In this slide, the logic to the left of the Bus Unit (such as the L3 cache, memory control, and HyperTransport) is shared by the four cores of the processor. The logic to the right of the Bus Unit is unique to a single core and is replicated for each core on the processor.

The instructions for the Opteron processors are variable length and perform multiple primitive operations. At the decoder, the instructions are converted to fixed-length macro-ops. Macro-ops contain one load/store and one integer or floating point operation. At the scheduler, the macro-ops are converted to micro-ops that include single fixed-length instructions that perform a single load, store, integer, or floating-point operation.

The cache-line in the L1 instruction cache and L1 data caches is 64-bytes wide. The L1 instruction cache is naturally aligned on 64-byte boundaries. On a fetch from memory, it will prefetch the next line. The replacement algorithm for the instruction cache is least recently-used. For the data cache, address bits 16 through 4 determine which set is referenced.

The L1, L2 and L3 caches are mutually exclusive; that is, data will not be duplicated in all three caches. The L2 cache stores victims from the L1 instruction and data caches. The L3 cache stores victims from the L2 caches of all four cores. Under certain conditions, when the data is being used by all four cores, that data will remain in the L3 cache. These cache restrictions permit more data to be stored closer to the processor, but may incur more write-backs than occur in a system where the caches are ranked.

Hardware prefetch can be detected on incrementing and decrementing streams. The data prefetched by PREFETCHNTA instructions is not saved to the L2 cache when it is evicted from the L1 cache unless it existed in the L2 cache before the prefetch occurs.
Core

- In order to optimize the utilization of the shared and dedicated resources on the chip for different types of applications, modern x86 processors offer flexible options for running applications. As a result, the definition of a core has become ambiguous.

- Definition of a Core for Blue Waters:
  - Equivalent to an AMD “Interlagos” Compute Unit, which is an AMD Interlagos “Bulldozer” core consisting of: one instruction fetch/decode unit, one floating point scheduler with two FMA execution units, two integer schedulers with multiple pipelines and L1 Dcache, and a L2 cache. This is sometimes also called a “Core Module.” A “core” = “compute unit” = “core module.”
Interlagos Processor Architecture

- Interlagos is composed of a number of “Bulldozer Modules” or “Compute Units”
  - A bulldozer module has shared and dedicated components
  - There are two independent integer units; shared L2 cache, instruction fetch, Icache; and a shared, 256-bit Floating Point resource
  - A single Integer unit can make use of the entire Floating Point resource with 256-bit AVX instructions
Building an Interlagos Processor

- Each processor die is composed of 4 bulldozer modules
  - The 4 modules share a memory controller and 8MB L3 data cache
  - Each processor die is configured with two DDR3 memory channels and multiple HT3 links
Interlagos Die Floorplan
Interlagos Processor

- Two die are packaged on a multi-chip module to form an Interlagos processor
  - Processor socket is called G34 and is compatible with Magny Cours
  - Package contains
    - 8 bulldozer modules
    - 16 MB L3 Cache
    - 4 DDR3 1333 or 1600 memory channels
Two MPI Tasks on a Bulldozer Module
("Dual-Stream Mode")

- An MPI task is pinned to each integer unit
  - Each integer unit has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit, instruction fetch, and the L2 Cache are shared between the two integer units
    - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code is highly scalable to a large number of MPI ranks
  - Code can run with 2GB per task memory footprint
  - Code is not well vectorized
One MPI Task on a Bulldozer Module
("Single Stream Mode")

- Only one integer unit is used per core module
  - This unit has exclusive access to the 256-bit FP unit and is capable of 8 FP results per clock cycle
  - The unit has twice the memory capacity and memory bandwidth in this mode
  - The L2 cache is effectively twice as large
  - The peak of the chip is not reduced

- When to use
  - Code is highly vectorized and makes use of AVX instructions
  - Code benefits from higher per task memory size and bandwidth
One MPI Task per Module with Two OpenMP Threads ("Dual-Stream Mode")

- An MPI task is pinned to a bulldozer module
- OpenMP is used to run a thread on each integer unit
  - Each OpenMP thread has exclusive access to an integer scheduler, integer pipelines and L1 Dcache
  - The 256-bit FP unit and the L2 Cache is shared between the two threads
  - 256-bit AVX instructions are dynamically executed as two 128-bit instructions if the 2nd FP unit is busy
- When to use
  - Code needs a large amount of memory per MPI rank
  - Code has OpenMP parallelism at each MPI rank
AVX (Advanced Vector Extensions)

- Max Vector length doubled to 256 bit
- Much cleaner instruction set
  - Result register is unique from the source registers
  - Old SSE instruction set always destroyed a source register
- Floating point multiple-accumulate
  - A(1:4) = B(1:4)*C(1:4) + D(1:4) ! Now one instruction
- Next gen of both AMD and Intel will have AVX

- Vectors are becoming more important, not less
Compilers Where to start

• For users who want to get the best performance they can the following gives you some information on compilers and settings to try

• Load the proper xtpe-<arch>
  – For the X6 systems use xtpe-mc8, xtpe-mc12, or xtpe-interlagos
    ▪ If the module is not loaded and no ‘arch’ is specified in the compiler options, the compilers default to the node type on which the compiler is running; which is not the same as the compute nodes!
    ▪ When using these modules the OpenMP threaded BLAS/LAPACK libraries are used
      • The serial version is used if ‘OMP_NUM_THREADS’ is not set or set to 1
GNU Compiler

- GNU’s Relative Strengths…from Cray’s Perspective
  - So-so Fortran, outstanding C and C++ (if you ignore vectorization)
    - Obviously, the best for gcc compatibility
    - Scalar optimizer was recently rewritten and is very good
    - Vectorization capabilities focus mostly on inline assembly
      - Note the last three releases have been incompatible with each other (4.3, 4.4, and 4.5) and required recompilation of Fortran modules
  - Options: -O3 -ffast-math -funroll-loops
  - Compiler feedback: -ftree-vectorizer-verbose=2
    - man gfortran; man gcc; man g++
CCE Compiler

- CCE’s Relative Strengths…from Cray’s Perspective
  - Outstanding Fortran, very good C, and good C++
    - Very good vectorization
    - Very good Fortran language support; only real choice for Coarrays
    - C support is quite good, with UPC support
    - Very good scalar optimization and automatic parallelization
    - Clean implementation of OpenMP 3.0, with tasks
    - Sole delivery focus is on Linux-based Cray hardware systems
    - Best bug turnaround time
    - Cleanest integration with other Cray tools (performance tools, debuggers, upcoming productivity tools)
    - No inline assembly support
Recommended CCE Options

• Use default optimization levels
  ▪ It’s the equivalent of most other compilers –O3 or –fast
  – Use –O3,fp3 (or –O3 –hfp3, or some variation)
    ▪ –O2 only gives you slightly more than –O2
    ▪ –hfp3 gives you a lot more floating point optimization, esp. 32-bit
  – If an application is intolerant of floating point reassociation, try a lower –hfp number, like –hfp1
    first, only use –hfp0 if absolutely necessary
    ▪ Might be needed for tests that require strict IEEE conformance
    ▪ Or applications that have ‘validated’ results from a different compiler
Recommended CCE Options

• Do not recommend using –Oipa5, –Oaggress, and so on
  – higher numbers are not always correlated with better performance
• Compiler feedback: –rm (Fortran) –hlist=m (C)
• If you know you do not want OpenMP: –xomp or
  –Othread0
• man pages: crayftn, craycc, crayCC
Loopmark: Compiler Feedback

- Compiler can generate an filename.lst file.
  - Contains annotated listing of your source code with letter indicating important optimizations

```
%%% Loopmark Legend %%%
Primary Loop Type  Modifiers
-------------  -------------
A - Pattern matched  b - blocked
C - Collapsed  f - fused
D - Deleted  i - interchanged
E - Cloned  m - streamed but not partitioned
I - Inlined  p - conditional, partial and/or computed
M - Multithreaded  r - unrolled
P - Parallel/Tasked  s - shortloop
V - Vectorized  t - array syntax temp used
W - Unwound  w - unwound
```
Loopmark: Compiler Feedback

```fortran
ftn -rm ...  or  cc -hlist=m ...

29. b------< do i3=2,n3-1
30. b b------< do i2=2,n2-1
31. b b Vr--< do i1=1,n1
32. b b Vr   ul(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
33. b b Vr   + u(i1,i2,i3-1) + u(i1,i2,i3+1)
34. b b Vr   u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
35. b b Vr   + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
36. b b Vr--> enddo
37. b b Vr--< do i1=2,n1-1
38. b b Vr   r(i1,i2,i3) = v(i1,i2,i3)
39. b b Vr   - a(0) * u(i1,i2,i3)
40. b b Vr   - a(2) * (u2(i1) + u1(i1-1) + u1(i1+1))
41. b b Vr   - a(3) * (u2(i1-1) + u2(i1+1))
42. b b Vr--> enddo
43. b b------> enddo
44. b-------> enddo
```
PGI Compiler

- The best compiler is not the same for every application
- PGI’s Relative Strengths… from Cray’s Perspective
  - Very good Fortran, okay C and C++
    - Good vectorization
    - Good functional correctness with optimization enabled
    - Good manual and automatic prefetch capabilities
    - Very interested in the Linux HPC market, although that is not their only focus
    - Excellent working relationship with Cray, good bug responsiveness
PGI compiler

- Suggested options: -fast
- Interprocedural analysis allows the compiler to perform whole-program optimizations, -Mipa=fast
- If you can be flexible with precision, try -Mfprelaxed
- Option -Msmartalloc call to the routine mallocopt in the main routine; this can have a dramatic impact on the performance of programs that dynamically allocate memory.
- Compiler feedback: -Minfo=all
- man pages: pgf90; pgcc; pgCC; or pgf90 -help
  - The -help option displays command line options
  - Request information about either a single option or groups of options: % pgcc -fast -help
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-default64</td>
<td>Fortran driver option for -i8 and -r8</td>
</tr>
<tr>
<td>-i8, -r8</td>
<td>Treats INTEGER and REAL variables in Fortran as eight bytes (Caution: Use the ftn -default64 option to link the right libraries, NOT -i8 or -r8)</td>
</tr>
<tr>
<td>-byteswapio</td>
<td>Reads big-endian files in Fortran. (Cray XT systems are little endian)</td>
</tr>
<tr>
<td>-Mnomain</td>
<td>Uses the ftn driver to link programs with the main program (written in C or C++) and one or more subroutines (written in Fortran)</td>
</tr>
</tbody>
</table>

Note: The PGI Fortran stop statement writes a FORTRAN STOP message to standard out. In a parallel application, this may not scale well. To turn off this message, use the environment variable NO_STOP_MESSAGE.

Note: Only the PGI compiler provides the -default64 option.
Other PGI Compiler Information

- It is possible to disable optimizations included with 
  -fast, for example -fast -Mnolre enables -fast and 
  then disables loop redundant optimizations
- The PGI option -Mconcur, -mprof=mpi, -Mmpi, and 
  -Mscalapack are not supported
- Fortran interfaces can be called from a C program by 
  inserting an underscore to the respective name
  - Pass arguments by reference rather than by value
  - For example to call dgetrf()
    
    dgetrf_(&uplo, &m, &n, a, &lda, ipiv, work, &lwork, &info);

- To debug an optimized code, the -gopt flag will insert 
  debugging information without disabling optimizations
Other PGI Compiler Information

• Some compiler options may affect both performance and accuracy
  - Lower accuracy is often higher performance, but it’s also able to enforce accuracy
    ▪ -Kieee All Floating Point (FP) math strictly conforms to IEEE 754
      • Off by default
    ▪ -Ktrap Turns processor trapping of FP exceptions
    ▪ -Mdaz Treat all denormalized numbers as zeros
    ▪ -Mflushz Set SSE to flush-to-zero (on with -fast)
    ▪ -Mfprelaxed Allow the computer to use relaxed (reduced) precision to speed up some floating point optimizations
  • Some other compilers turn this on by default, PGI chooses to favor accuracy to speed by default
<table>
<thead>
<tr>
<th>PathScale Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pathscale’s Relative Strengths…from Cray’s Perspective</strong></td>
</tr>
<tr>
<td>– Good Fortran, C, probably good C++</td>
</tr>
<tr>
<td>▪ Outstanding scalar optimization for loops that do not vectorize</td>
</tr>
<tr>
<td>▪ Fortran front end uses an older version of the CCE Fortran front end</td>
</tr>
<tr>
<td>▪ OpenMP uses a non-pthreads approach</td>
</tr>
<tr>
<td>▪ Scalar benefits will not get as much mileage with longer vectors</td>
</tr>
<tr>
<td>– <strong>Option:</strong> <code>-Ofast</code></td>
</tr>
<tr>
<td>▪ Note: This is less precise with precision than other compilers</td>
</tr>
<tr>
<td>– <strong>Compiler feedback:</strong> <code>-LNO:simd_verbose=ON</code></td>
</tr>
<tr>
<td>– <strong>man pages:</strong> <code>pathcc, pathCC, pathf95, or eko</code> <em>(Every Known Optimization)</em></td>
</tr>
</tbody>
</table>
The Opteron L1 data cache is a 64KB 2-way set associative cache; each cache line is 64 bytes (512 bits)
- Latency from the L1 cache is 3 clocks (2/cycle)
  - Any stride that is a multiple of 64 causes L1 bank conflicts
  - Addresses 512*64 (or 32768 apart) come back to the same row; a refill from L2 has a latency of 11 or more clocks; a useful principle is to avoid large powers of 2

L1 data cache:
- 512 rows, 2 ways (512 x 2 way = 1024, 1024 x 64-bytes (per line) = 64K)
- Size: 64 KB
- Index: bits 14–6 (64-byte stride, 32768-byte wrap)
- 3-clock latency

Two identical L1 physical tags arrays (and L1 DTLBs) translate two virtual addresses simultaneously.
Each cache line resides on eight banks (indexed by bits 5:3). Two loads from the L1 conflict cannot occur at the same time if their bank number is the same (unless their line index is also the same).
The L1 has eight banks; virtual address bits 5:3 select the bank. Accessing the same bank causes a bank conflict. A loop that has fewer loads per cycle (performance counters) may indicate a bank conflict.
A rather obvious statement: Initialize data registers before you read them because reading uninitialized data registers can cause cache trashing.
The cache reference address is a virtual address. A virtual address is used because there are not enough untranslated addresses. Virtual addressing enables you to determine an address while address translation is occurring. The “tag” of the cache-line holds the physical address.
A detailed hardware reference:
L2 cache:
- 512 rows, 16 ways
- Size: 512KB
- Index: bits 14:6 (64-byte stride, 65536-byte wrap)
- Unified – caches both data and instructions

The L1 and L2 cache are mutually exclusive; that is, a cache line cannot exist in both caches simultaneously; data evicted from the L1 (to reuse an L1 line) is stored in the L2.

Read-ahead:
If the memory controller detects multiple consecutive cache line accesses, it will read a maximum of four cache lines and place them into the L2 cache.

The Nx bit (bit 63 of a page table entry) allows the OS to specify where executable code can NOT be; this provides a more efficient control of memory. Bit 47:0 are the virtual address, bit 63:47 are copies of bit 47; when the address is in the correct form it is said to be in “canonical form”.

---

Memory Access Patterns (L2)

- The Opteron L2 cache is a 512KB 16-way set associative cache; each cache line is 64 bytes
  - Latency from the L2 cache is 11 or more clocks
  - Addresses that are 1024*64 (or 65536) apart return to the same row; latency for a refill from memory is ~150cp; a useful principle is to avoid large powers of 2
Cache Optimization

• Cache effects may be more important than vectorization
  – References to main memory relatively expensive

• The AMD Opteron processors have 3 levels of cache
  – 6 MB level 3 cache – shared across cores
  – 512 KB level 2 cache
  – 64 KB level 1 cache

• The more memory references that are in cache helps performance in two ways
  ▪ Eliminates going out to main memory
  ▪ Helps lessen memory contention across the cores in a socket

• The most effective techniques are cache blocking and loop splitting
Two types of cache data reuse

• Loop-nest cache blocking
  – Cache blocking is a combination of strip mining and loop interchange within a single loop nest, designed to increase data reuse.

• Inter-loop/region/subroutine cache reuse
  – Effectively bring a working data set into the cache and then reuses that data many times across multiple loops or even subroutines
Cache Blocking

- Cache blocking is a combination of strip mining and loop interchange, designed to increase data reuse.
  - Takes advantage of temporal reuse: re-reference array elements already referenced
  - Good blocking will take advantage of spatial reuse: work with the cache lines
- Many ways to block any given loop nest
  - Which loops should get blocked?
  - What block size(s) should be used?
- Analysis can reveal which ways are beneficial
  - But trial-and-error is probably faster
- Good news – Cray compiler optimizations include loop blocking
Cache Use in Stencil Computations

- **2D Laplacian**
  
  ```
  do j = 1, 8
    do i = 1, 16
      a = u(i-1,j) + u(i+1,j) &
      - 4*u(i,j) &
      + u(i,j-1) + u(i,j+1)
    end do
  end do
  ```

- **Cache structure for this example:**
  - Each line holds 4 array elements
  - Cache can hold 12 lines of u data

- **No cache reuse between outer loop iterations**
Chapter 8: Single-node Optimization

Blocking to Increase Reuse

- Unblocked loop: 120 cache misses
- Block the inner loop

```
  do IBLOCK = 1, 16, 4  
    do j = 1, 8          
      do i = IBLOCK, IBLOCK + 3  
        a(i,j) = u(i-1,j) + u(i+1,j) &  
        - 4*u(i,j) &  
        + u(i,j-1) + u(i,j+1)  
      end do  
    end do  
  end do
```

- Now we have reuse of the “j+1” data
Blocking to Increase Reuse

- One-dimensional blocking reduced misses from 120 to 80

- Iterate over 4x4 blocks

```plaintext
do JBLOCK = 1, 8, 4
  do IBLOCK = 1, 16, 4
    do j = JBLOCK, JBLOCK + 3
      do i = IBLOCK, IBLOCK + 3
        a(i,j) = u(i-1,j) + u(i+1,j) &
        - 4*u(i,j) &
        + u(i,j-1) + u(i,j+1)
      end do
    end do
  end do
end do
```

- Better use of spatial locality (cache lines)
## Initial Test

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>42.4%</td>
</tr>
<tr>
<td>Time</td>
<td>12.397761</td>
</tr>
<tr>
<td>Imb.Time</td>
<td>0.000370</td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Calls</td>
<td>340</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>2719.188M/sec</td>
</tr>
<tr>
<td></td>
<td>33711498004</td>
</tr>
<tr>
<td>DC_L2_REFILL_MOESI</td>
<td>79.644M/sec</td>
</tr>
<tr>
<td></td>
<td>987402929</td>
</tr>
<tr>
<td>DC_SYS_REFILL_MOESI</td>
<td>4.059M/sec</td>
</tr>
<tr>
<td></td>
<td>50318116</td>
</tr>
<tr>
<td>BU_L2_REQ_DC</td>
<td>129.172M/sec</td>
</tr>
<tr>
<td></td>
<td>1601429574</td>
</tr>
<tr>
<td>User time</td>
<td>12.398 secs</td>
</tr>
<tr>
<td></td>
<td>32233848320</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>L1 Data cache misses</td>
<td>83.703M/sec</td>
</tr>
<tr>
<td></td>
<td>1037721045</td>
</tr>
<tr>
<td>LD &amp; ST per D1 miss</td>
<td>32.49 ops/miss</td>
</tr>
<tr>
<td>D1 cache hit ratio</td>
<td>96.9%</td>
</tr>
<tr>
<td>LD &amp; ST per D2 miss</td>
<td>669.97 ops/miss</td>
</tr>
<tr>
<td>D2 cache hit ratio</td>
<td>96.9%</td>
</tr>
<tr>
<td>L2 cache hit ratio</td>
<td>95.2%</td>
</tr>
<tr>
<td>Memory to D1 refill</td>
<td>4.059M/sec</td>
</tr>
<tr>
<td></td>
<td>50318116</td>
</tr>
<tr>
<td>Memory to D1 bandwidth</td>
<td>247.723MB/sec</td>
</tr>
<tr>
<td></td>
<td>3220359424</td>
</tr>
<tr>
<td>L2 to Dcache bandwidth</td>
<td>4861.112MB/sec</td>
</tr>
<tr>
<td></td>
<td>63193787456</td>
</tr>
</tbody>
</table>
# Chapter 8: Single-node Optimization

## After Cache Blocking

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>36.3%</td>
</tr>
<tr>
<td>Time</td>
<td>8.753226</td>
</tr>
<tr>
<td>Imb. Time</td>
<td>0.000596</td>
</tr>
<tr>
<td>Imb. Time%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Calls</td>
<td>340</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>3861.533M/sec</td>
</tr>
<tr>
<td>DC_L2_REFILL_MOESI</td>
<td>116.399M/sec</td>
</tr>
<tr>
<td>DC_SYS_REFILL_MOESI</td>
<td>2.755M/sec</td>
</tr>
<tr>
<td>BU_L2_REQ_DC</td>
<td>161.490M/sec</td>
</tr>
<tr>
<td>User time</td>
<td>8.753 secs</td>
</tr>
<tr>
<td>Utilization rate</td>
<td>100.0%</td>
</tr>
<tr>
<td>L1 Data cache misses</td>
<td>119.154M/sec</td>
</tr>
<tr>
<td>LD &amp; ST per D1 miss</td>
<td>32.41 ops/miss</td>
</tr>
<tr>
<td>D1 cache hit ratio</td>
<td>96.9%</td>
</tr>
<tr>
<td>LD &amp; ST per D2 miss</td>
<td>1401.70 ops/miss</td>
</tr>
<tr>
<td>D2 cache hit ratio</td>
<td>98.3%</td>
</tr>
<tr>
<td>L2 cache hit ratio</td>
<td>97.7%</td>
</tr>
<tr>
<td>Memory to D1 refill</td>
<td>2.755M/sec</td>
</tr>
<tr>
<td>Memory to D1 bandwidth</td>
<td>168.145MB/sec</td>
</tr>
<tr>
<td>L2 to Dcache bandwidth</td>
<td>7104.420MB/sec</td>
</tr>
</tbody>
</table>
Cray XE MPI Environment

Customer Documentation and Training
MPICH2 and Cray MPT

- Cray uses MPICH2
  - Provides a good, robust and feature rich MPI
  - Cray provides low level communication libraries
  - Point to point tuning
  - Collective tuning
  - Shared memory device is built on top of Cray XPMEM
- Many layers are straight from MPICH2
  - Error messages can be from MPICH2

See the man page intro_mpi for more information (note the same man page, but different information on SeaStar and Gemini systems)
How MPICH2 Uses GNI – Key Concepts

• Due to lack of messaging hardware on Gemini, a connection oriented approach is used (GNI SMSG mailboxes)
  – The relatively limited memory registration resources have a major impact on the MPICH2 GNI Netmod design. Using large pages generally helps alleviate problems associated with these limited memory registration resources.
  – All network transactions are tracked at some level. No fire-and-forget. Helps with dealing with transient network errors.
How MPICH2 Uses GNI – SMSG Mailboxes

- Uses put-with-notification hardware on Gemini. This allows implementing a one-way-through network messaging scheme.
- Can recover from transient network errors
- Flow control
- MPICH2 and GNILND (Lustre, DVS, etc.) share same mailbox code
- By default, connections (mailboxes) are established dynamically. Note mailboxes are actually allocated in blocks due to limited memory registration resources on NIC.
- Process private and shared SMSG mailboxes available. Current MPICH2 only uses private ones.
- For private SMSG mailboxes memory usage per-rank scales linearly with number of connections.
Implications of GNI SMSG Mailboxes for Apps

- Applications with scalable communication patterns benefit from high message rates and low latency of GNI private SMSG mailboxes without large memory consumption. (~1.7 μsec latency nn, ~1.4 MM/sec nn/rank*)

- Applications with dense communication graphs aren’t going to scale too well on Gemini or Aries using GNI private SMSG mailboxes, may be okay with shared SMSG mailboxes.

* For ranks on dieA

![Diagram showing network graphs for scalable and dense communication patterns.](image-url)
MPICH2 GNI Netmod Message Protocols

• Eager Protocol
  – For a message that can fit in a GNI SMSG mailbox (E0)
  – For a message that can’t fit into a mailbox but is less than MPICH_GNI_MAX_EAGER_MSG_SIZE in length (E1)

• Rendezvous protocol (LMT)
  – RDMA Get protocol – up to 512 KB size messages by default
  – RDMA Put protocol – above 512 KB
Max Message Size for E0 Varies with Job Size

- Protocol for messages that can fit into a GNI SMMSG mailbox
- The default varies with job size, although this can be tuned by the user to some extent

<table>
<thead>
<tr>
<th>ranks in job</th>
<th>maximum bytes of user data</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 1024</td>
<td>984</td>
</tr>
<tr>
<td>&gt;1024 &amp;&amp; &lt;=16384</td>
<td>472</td>
</tr>
<tr>
<td>&gt; 16384</td>
<td>216</td>
</tr>
</tbody>
</table>
MPICH_GNI_MAX_VSHORT_MSG_SIZE

- Can be used to control the maximum size message that can go through the private SMSG mailbox protocol (E0 eager path).

- Default varies with job size

- Maximum size is 1024 bytes. Minimum is 80 bytes.

- If you are trying to demonstrate an MPI_Alltoall at very high count, with smallest possible memory usage, may be good to set this as low as possible.

- If you know your app has a scalable communication pattern, and the performance drops at one of the edges shown on table on Slide 11, you may want to set this environment variable.

- Pre-posting receives for this protocol avoids a potential extra memcpy at the receiver.
**MPICH_GNI_MAX_EAGER_MSG_SIZE**

- Maximum size message that go through the eager (E1) protocol
  - Default is 8192 bytes
    - Maximum allowable setting is 131072 bytes
    - Note that a 40-byte Nemesis header is included in account for the message size
- May help for applications sending medium size messages and do better when loosely coupled
  - Does the application have a large amount of time in `MPI_Waitall`?
    - Setting this environmental variable higher may help
- Pre-posting receives can avoid potential double memcpy at the receiver
**MPICH_GNI_RDMA_THRESHOLD**

- Controls the threshold at which the GNI netmod switches from using FMA for RDMA read/write operations to using the BTE
  - Default is now 1024 bytes
- Since BTE is managed in the kernel, BTE initiated RDMA requests can progress even if the applications isn’t in MPI
  - Owing to Opteron/HyperTransport quirks, the BTE is often better for moving data to/from memories that are farther from the Gemini
  - But using the BTE may lead to more interrupts being generated
**MPICH_GNI_NDREG_LAZYMEM**

- Controls whether or not to use a lazy memory deregistration policy inside UDREG
  - Memory registration is expensive so this is usually a good idea
  - Important for applications using LMT (large message transfer)/rendezvous protocol and 4KB pages
    - Messages greater than `MPICH_GNI_MAX_EAGER_MSG_SIZE`.
  - Default is enabled

- To disable:
  ```
  export MPICH_GNI_NDREG_LAZYMEM=disabled
  ```
  - Disabling results in a significant drop in measured bandwidth for large transfers ~40-50 %
  - Would be better using huge pages than disabling this
MPICH_GNI_DMAPP_INTEROP

- Only relevant for mixed MPI/SHMEM/UPC/CAF codes
- For Danube systems, want to leave enabled so MPICH2 and DMAPP can share the same memory registration cache, reducing pressure on memory registration resources on the NIC
- May have to disable for SHMEM codes that call shmем_init after MPI_Init.
- May want to disable if trying to add SHMEM/CAF to an MPI code and notice a big performance drop.
- Syntax:

  export MPICH_GNI_DMAPP_INTEROP=disabled
MPICH_GNI_NUM_BUFS

- Controls the number of 32KB DMA buffers available for each rank to use in the GET-based Eager protocol (E1).
  - Default is 64 32K buffers (2M total)
- May help to modestly increase. But other resources constrain the usability of a large number of buffers
- Syntax: `export MPICH_GNI_NUM_BUFS=X`
**MPICH_GNI_DYNAMIC_CONN**

- Enabled by default
- Normally want to leave enabled so mailbox resources (memory, NIC resources) are allocated only when the application needs them
- If application does all-to-all or many-to-one/few, may as well disable dynamic connections. This will result in significant startup/shutdown costs though.
- Recent bugs have been worked around by disabling dynamic connections.
- Syntax for disabling:

  ```
  export MPICH_GNI_DYNAMIC_CONN=disabled
  ```
<table>
<thead>
<tr>
<th>MPICH_GNI_MBOX_PLACEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Provides a means for controlling which memories on a node are used for some SMSG mailboxes (private).</td>
</tr>
<tr>
<td>• Default is to place the mailboxes on the memory where the process is running when the memory for the mailboxes is faulted in.</td>
</tr>
<tr>
<td>• For optimal MPI message rates, better to place mailboxes on memory of die0 (where Gemini is attached).</td>
</tr>
<tr>
<td>• Only applies to first 4096 mailboxes of each rank on the node.</td>
</tr>
<tr>
<td>• Feature only available in very recent CLE versions (not all UP01s?) and very most recent MPT build (actually none yet due to build issues as of today).</td>
</tr>
<tr>
<td>• Syntax for enabling placement of mailboxes near the Gemini: <code>export MPICH\_GNI\_MBOX\_PLACEMENT=nic</code></td>
</tr>
</tbody>
</table>
MPI_Allgather

- With MPT 5.1 switched to using Seastar-style algorithm where for short transfers/rank: use MPI_Gather/MPI_Bcast rather than ANL algorithm
- Switchover from Cray algorithm to ANL algorithm can be controlled by the `MPICH_ALLGATHER_VSHORT_MSG` environment variable. By default enabled for transfers/rank of 1024 bytes or less
- The Cray algorithm can be deactivated by setting

  ```bash
  export MPICH_COLL_OPT_OFF=mpi_allgather
  ```

  ```tcsh
  setenv MPICH_COLL_OPT_OFF mpi_allgather
  ```

ANL = Argonne National Lab, birthplace of MPICH2
**MPI_Allgatherv**

- With MPT 5.1 switched to using Seastar-style algorithm where for short transfers/rank: use a specialized MPI_Gatherv/MPI_Bcast rather than ANL algorithm.

- Switchover from Cray algorithm to ANL algorithm can be controlled by the `MPICH_ALLGATHERV_VSHORT_MSG` environment variable. By default enabled for transfers/rank of 1024 bytes or less.

- The Cray algorithm can be deactivated by setting

  ```bash
echo MPICH_COLL_OPT_OFF=mpi_allgatherv
setenv MPICH_COLL_OPT_OFF mpi_allgatherv
tcsh
```
MPI_Alltoall

- Switchover from ANL’s implementation of Bruck algorithm (IEEE TPDS, Nov. 1997) is controllable via the MPICH_ALLTOALL_SHORT_MSG environment variable.
  - Larger transfers use an optimized pair-wise exchange algorithm

- New algorithm can be disabled by
  - export MPICH_COLL_OPT_OFF=mpi_alltoall

- Defaults are

<table>
<thead>
<tr>
<th>ranks in communicator</th>
<th>Limit (in bytes) for using Bruck</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 512</td>
<td>2048</td>
</tr>
<tr>
<td>&gt;512 &amp;&amp; &lt;=1024</td>
<td>1024</td>
</tr>
<tr>
<td>&gt; 1024</td>
<td>128</td>
</tr>
</tbody>
</table>
MPI_Allreduce/MPI_Reduce

- The ANL smp-aware MPI_Allreduce/MPI_Reduce algorithms can cause issues with bitwise reproducibility. To address this Cray MPICH2 has two new environment variables starting with MPT 5.1 -

  - `MPI_ALLREDUCE_NO_SMP` – disables use of smp-aware `MPI_Allreduce`
  
  - `MPI_REDUCE_NO_SMP` – disables use of smp-aware `MPI_Reduce`
MPI_Bcast

- Starting with MPT 5.1, all ANL algorithms except for binomial tree are disabled since the others perform poorly for communicators with 512 or more ranks

- To disable this tree algorithm-only behaviour, set the `MPICH_BCAST_ONLY_TREE` environment variable to 0, i.e.

```bash
export MPICH_BCAST_ONLY_TREE=0
```
Note that XPMEM is completely different from KNEM (INRIA) which is in use in MPICH2 deployed on other Linux cluster systems. Don’t get confused if following mpich-discuss, etc.
MPICH_SMP_SINGLE_COPY_OFF

- Specifies whether or not to use a XPMEM-based single-copy protocol for intra-node messages of size MPICH_SMP_SINGLE_COPY_SIZE bytes or larger
  - The default is enabled
  - May need to set this environment variable if
    - Finding XPMEM is kernel OOPses (check the console on the SMW)
    - Sometimes helps if hitting UDREG problems. XPMEM goes kind of crazy with Linux mmu notifiers and causes lots of UDREG invalidations (at least the way MPICH2 uses XPMEM).
Rank Placement

• The default ordering can be changed by using the following environment variable:
  – MPICH_RANK_REORDER_METHOD
  – Values that you can set it to:
    ▪ 0: Round-robin placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
    ▪ 1: (DEFAULT) SMP-style placement – Sequential ranks fill up each node before moving to the next.
    ▪ 2: Folded rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
    ▪ 3: Custom ordering. The ordering is specified in a file named MPICH_RANK_ORDER.
Rank Placement

- When is this useful?
  - When point-to-point communication consumes a significant fraction of program time and a load imbalance is detected
  - Also shown to help for collectives (alltoall) on subcommunicators
  - Spread out IO across nodes
Reordering example: GYRO

- **GYRO 8.0**
  - B3-GTC problem with 1024 processes
- Run with alternate MPI orderings

<table>
<thead>
<tr>
<th>Reorder method</th>
<th>Comm. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – SMP (Default)</td>
<td>11.26s</td>
</tr>
<tr>
<td>0 – round-robin</td>
<td>6.94s</td>
</tr>
<tr>
<td>2 – folded-rank</td>
<td>6.68s</td>
</tr>
</tbody>
</table>

**Note:**
- The rank reordering works only on nodes. If you want to pack within a node in a special way, use the `aprun -cc 'cpu list'`
- Hence, to get a bit more out of the folded-rank option use:
  ```
  aprun -cc 0,6,12,18,19,13,7,1,2,8,14,20,21,15,9,3,4,10,16,22,23,17,11,5
  ```
- This folds across nodes, and folds within dies on a node
**Reordering example: TGYRO**

- **TGYRO 1.0**
  - Steady-state turbulent transport code using GYRO, NEO, TGLF components

- **ASTRA test case**
  - Tested MPI orderings at large scale
  - Originally testing weak-scaling, but found reordering very useful

<table>
<thead>
<tr>
<th>Reorder method</th>
<th>TGYRO wall time (min)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20480 Cores</td>
<td>40960 Cores</td>
</tr>
<tr>
<td>Default</td>
<td>99m</td>
<td>104m</td>
</tr>
<tr>
<td>Round-robin</td>
<td>66m</td>
<td>63m</td>
</tr>
</tbody>
</table>

Huge win!
Rank order choices

- Many options, depends on pattern
  - Check out `pat_report`, `grid_order`, and `mgrid_order` for generating custom rank orders based on:
    - Measured data
    - Communication patterns
    - Data decomposition

- Nodes marked X use a shared resource heavily
  - If the shared resource is:
    - Memory bandwidth: scatter the X's
    - Network bandwidth to others, again scatter
    - Network bandwidth among themselves, concentrate