# Cray Performance Tools

Gene Wagenbreth
Performance Engineer
gwagenbret@cray.com

# Cray Performance Tools

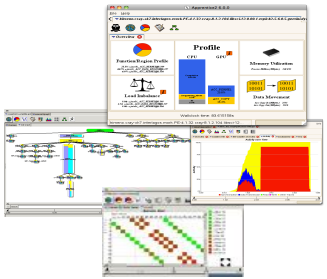
Heidi Poxon
Sr. Principal Engineer
Cray Programming Environment

# Agenda

- Overview
- Using Cray Performance Tools
- Bottleneck Detection
- Imbalance
- MPI rank reordering

# Using Cray Performance Tools

**Load modules to access software**



**Build and instrument program**

**Run program and view results**

# Cray Performance Analysis Tools

- **Whole program performance analysis with**
  - Novice and advanced user interfaces
  - Support for MPI, SHMEM, OpenMP, UPC, CAF, OpenACC, CUDA
  - Load imbalance detection
  - HW counter metrics (hit rates, computational intensity, etc.)
  - Observations and inefficiencies
  - Data correlation to user source
  - Minimal program perturbation
- **Sampling, tracing with runtime summarization, full trace (timeline) modes available**
- **Supports CCE, Intel and GCC compilers**

# Cray Performance Analysis Tools

- Convenient to make many small runs to optimize performance
- Should make some large runs to verify that observations carry over to large runs
- Apprentice2 provides visual interface to performance data

# Two Modes of Use

- **CrayPat-lite** for novice users, or convenience
- **CrayPat** for in-depth performance investigation and tuning assistance
- **Both offer:**
  - Whole program analysis across many nodes
  - Indication of causes of problems
  - Suggestions of modifications for performance improvement

# “Lite” Mode

## Load performance tools instrumentation module

```
$ module load perftools-base
$ module load perftools-lite
```

## Build program
## (no modification to makefile)

```
$ make
```

a.out (instrumented program)

## Run program
## (no modification to batch script)

```
$ aprun a.out
```

Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
files

# Example CrayPat-lite Output

```
########################################################################
#                                                                      #
#            CrayPat-lite Performance Statistics                       #
#                                                                      #
########################################################################

CrayPat/X:  Version 6.3.2.461 Revision 56930ff  02/01/16 15:31:33
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):     64
Numbers of PEs per Node:       32  PEs on each of  2  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:    16
Execution start time:  Tue Feb  2 18:53:50 2016
System name and speed:  kay  2301 MHz (approx)

Avg Process Time:         64.38 secs
High Memory:              1,563 MBytes  24.43 MBytes per PE
MFLOPS:           Not supported (see observation below)
I/O Read Rate:        48.514130 MBytes/sec
I/O Write Rate:       22.281350 MBytes/sec
Avg CPU Energy:          41,820 joules 20,910 joules per node
Avg CPU Power:           649.53 watts  324.77 watts per node
```

```
Table 1:  Profile by Function Group and Function (top 10 functions
shown)

  Samp% |    Samp |  Imb. |  Imb. |Group
        |         |  Samp | Samp% | Function
        |         |       |       |  PE=HIDE

 100.0% | 6,156.2 |    -- |    -- |Total
|-----------------------------------------------------------
|  66.3% | 4,082.5 |    -- |    -- |USER
||----------------------------------------------------------
||  11.8% |   729.2 |  48.8 |  6.4% |mult_su3_mat_vec_sum_4dir
||  10.2% |   629.4 |  49.6 |  7.4% |mult_adj_su3_mat_4vec
||   6.1% |   377.1 |  28.9 |  7.2% |mult_su3_nn
||   5.9% |   365.4 |  42.6 | 10.6% |mult_su3_na
||   5.4% |   329.4 |  37.6 | 10.4% |scalar_mult_add_lathwvec_proj
||   3.8% |   232.9 |  39.1 | 14.6% |mult_su3_sitelink_lathwvec
||==========================================================
|  25.3% | 1,557.0 |    -- |    -- |MPI
||----------------------------------------------------------
||  12.8% |   789.3 | 163.7 | 17.5% |MPI_Wait
||   6.7% |   411.9 |  74.1 | 15.5% |MPI_Isend
||   4.9% |   300.2 |  95.8 | 24.6% |MPI_Allreduce
||==========================================================
|   5.9% |   365.4 |  44.6 | 11.1% |STRING
||----------------------------------------------------------
||   5.9% |   365.4 |  44.6 | 11.1% |memcpy
|==========================================================
```

# Identify High Time Consuming Areas

```
Table 2:  Profile by Group, Function, and Line
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  Source
        |          |         |       |   Line
        |          |         |       |    PE=HIDE
 100.0% | 55,605.7 |      -- |    -- |Total
|-------------------------------------------------------------------------------
|  56.5% | 31,412.8 |      -- |    -- |USER
||------------------------------------------------------------------------------
||  19.7% | 10,944.1 |      -- |    -- |create_boundary$boundary_
3|        |          |         |       | source.omp_removed/test/compile/boundary.f90
||||----------------------------------------------------------------------------
4|||   7.8% |  4,355.9 |   175.1 |  3.9% |line.265
4|||   1.8% |    977.0 |    98.0 |  9.1% |line.268
4|||   1.1% |    617.0 |    94.0 | 13.2% |line.273
4|||   2.0% |  1,133.6 |   101.4 |  8.2% |line.549
4|||   1.1% |    590.0 |    66.0 | 10.1% |line.557
||||============================================================================
||  10.7% |  5,937.8 |      -- |    -- |get_block$blocks_
3|        |          |         |       | source.omp_removed/test/compile/blocks.f90
||||----------------------------------------------------------------------------
4|||   4.1% |  2,305.7 |   145.3 |  5.9% |line.221
4|||   1.0% |    569.5 |    77.5 | 12.0% |line.243
4|||   2.9% |  1,610.1 |   134.9 |  7.7% |line.246
```

# Guidance: How Can I Learn More?

```
MPI utilization:

    The time spent processing MPI communications is relatively high.
    Functions and callsites responsible for consuming the most time can
    be found in the table generated by pat_report -O callers+src (within
    the MPI group).
```

# Guidance: Reduce Shared Resource Contention

Metric-Based Rank Order:

When the use of a shared resource like memory bandwidth is unbalanced across nodes, total execution time may be reduced with a rank order that improves the balance.

A file named MPICH_RANK_ORDER.USER_Time was generated along with this report and contains usage instructions and the custom rank order from the following table.

| Rank Order | Node Metric Imb. | Reduction in Max Value | Maximum Value | Average Value |
|---|---|---|---|---|
| Current | 15.46% | | 1.134e+03 | 9.588e+02 |
| Custom | 1.46% | 14.202% | 9.731e+02 | 9.588e+02 |

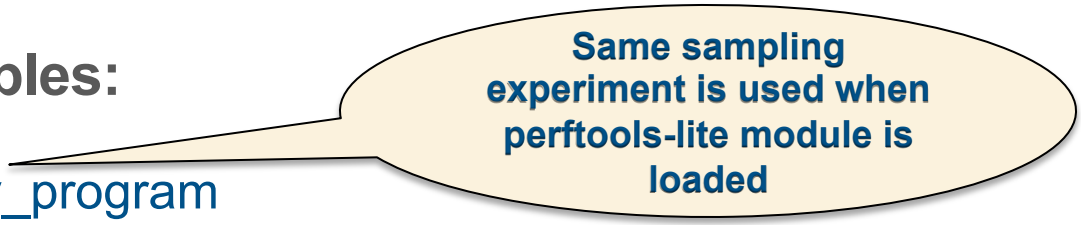
# Data from pat_report

- **Default reports are intended to be useful for most applications**
- **Don't need to rerun program to get more detailed data**
- **Different aggregations, or levels of information available**
  - Get fined-grained thread-imbalance information for OpenMP program built with CCE
    - $ pat_report -s pe=ALL –s th=ALL

# More In-depth Analysis and Bottleneck Detection

# How to Use CrayPat

- **Make sure the following modules are loaded:**
  - $ module load perftools-base perftools
- **2 instrumentation examples:**
  - $ pat_build my_program
  - $ pat_build –u –g mpi  my_program
- **Run application**
  - $ aprun -n …  my_program+pat
- **Create report**
  - $ pat_report my_program.xf > my_report



# Sample vs Trace

- **Sample mode**
  - Checks program counter and call stack 60 times per second
  - Minimal effect on execution
- **Trace mode**
  - Trace code inserted
  - Other information such as MPI message size
  - Cray compiler only – loops and loop lengths
  - Trace of small routines affects runtime
- **Trace routines from sample run**
  - 2 step – sample then trace gives best of both

# Predefined Trace Wrappers (-g tracegroup)

- blas Basic Linear Algebra subprograms
- caf Co-Array Fortran (Cray CCE compiler only)
- hdf5 manages extremely large data collection
- heap dynamic heap
- io includes stdio and sysio groups
- lapack Linear Algebra Package
- math ANSI math
- mpi MPI
- omp OpenMP API
- pthreads POSIX threads
- shmem SHMEM
- sysio I/O system calls
- system system calls
- upc Unified Parallel C (Cray CCE compiler only)

For a full list, please see pat_build(1) man page

# Control Data Collection with Runtime Options

- **Runtime controlled through PAT_RT_XXX environment variables**
- **See intro_craypat(1) man page**
- **Examples of control**
  - Enable full trace
  - Change number of data files created
  - Enable collection of CPU, network or power counter events
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)

# Performance Counters Overview

- **Cray supports raw counters, derived metrics and thresholds for:**
  - Processor (core and uncore)
  - Network
  - Accelerator
  - Power
- **Counters restricted for Haswell and KNL**
- **Predefined groups**
  - Groups together counters for experiments
- **See *hwpc, nwpc, accpc, and rapl* man pages**

# How to Get List of Events for a Processor

- **Run the following utility on a compute node:**
  - papi_native_avail
- **To collect performance counters**
  - Set PAT_RT_PERFCTR environment variable to list of events or group prior to execution

# MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced (pat_build –g mpi)

# Motivation for Load Imbalance Analysis

- **Increasing system software and architecture complexity**
  - Current trend in high end computing is to have systems with tens of thousands of processors
    - This is being accentuated with multi-core processors
- **Applications have to be very well balanced In order to perform at scale on these MPP systems**
  - Efficient application scaling includes a balanced use of requested computing resources
- **Desire to minimize computing resource "waste"**
  - Identify slower paths through code
  - Identify inefficient "stalls" within an application

# Find Program Load Imbalance

```
Table 1:  Profile by Function Group and Function

  Time% |     Time |     Imb. |   Imb. |    Calls |Group
        |          |     Time |  Time% |          | Function
        |          |          |        |          |  PE=HIDE

 100.0% | 1.957703 |       -- |     -- | 42,970.8 |Total
|-----------------------------------------------------------------------
|  60.0% | 1.174021 |       -- |     -- |  3,602.0 |USER
||----------------------------------------------------------------------
||  30.8% | 0.603850 | 0.176924 |  23.0% |  1,198.0 |calc3_
||  19.2% | 0.375117 | 0.128748 |  26.0% |  1,200.0 |calc2_
||   9.1% | 0.178111 | 0.081880 |  32.0% |  1,200.0 |calc1_
||======================================================================
|  36.0% | 0.704928 |       -- |     -- |  9,613.0 |MPI_SYNC
||----------------------------------------------------------------------
||  25.8% | 0.505174 | 0.385130 |  76.2% |  9,596.0 |mpi_barrier_(sync)
||  10.2% | 0.199537 | 0.199518 | 100.0% |      1.0 |mpi_init_(sync)
||======================================================================
|   4.0% | 0.078736 |       -- |     -- | 29,754.8 |MPI
||----------------------------------------------------------------------
||   2.3% | 0.045351 | 0.003531 |   7.3% |  9,596.0 |MPI_BARRIER
||   1.1% | 0.021520 | 0.051295 |  71.6% |  8,756.9 |MPI_ISEND
|======================================================================
```

# Load Imbalance May be Misleading

- Time may show in Barrier, Send or Recv
- May be due to cpu imbalance elsewhere

# Sort MPI Messages by Caller

```
   MPI |      MPI Msg |  MPI Msg |   MsgSz |  4KiB<= |Function
   Msg |        Bytes |    Count |     <16 |   MsgSz | Caller
 Bytes% |              |          |   Count |  <64KiB |  PE=[mmm]
       |              |          |         |   Count |
 100.0% | 34,940,767.4 |  8,771.9 |   258.6 | 8,513.3 |Total
|-----------------------------------------------------------------
| 100.0% | 34,940,647.4 |  8,756.9 |   243.6 | 8,513.3 |MPI_ISEND
||----------------------------------------------------------------
||  56.2% | 19,622,700.0 |  4,837.5 |    56.2 | 4,781.2 |calc2_
3|       |              |          |         |         | shalow_
||||--------------------------------------------------------------
4|||  56.4% | 19,718,400.0 |  7,200.0 | 2,400.0 | 4,800.0 |pe.0
4|||  56.4% | 19,699,200.0 |  4,800.0 |     0.0 | 4,800.0 |pe.32
4|||  42.3% | 14,784,000.0 |  4,800.0 | 1,200.0 | 3,600.0 |pe.63
||||==============================================================
||  42.5% | 14,851,950.0 |  3,693.8 |    75.0 | 3,618.8 |calc1_
3|       |              |          |         |         | shalow_
||||--------------------------------------------------------------
4|||  56.4% | 19,718,400.0 |  7,200.0 | 2,400.0 | 4,800.0 |pe.0
4|||  42.3% | 14,774,400.0 |  3,600.0 |     0.0 | 3,600.0 |pe.31
4|||  42.3% | 14,774,400.0 |  3,600.0 |     0.0 | 3,600.0 |pe.62
```

# Analyze MPI Message Sizes

```
Total
-----------------------------------------------------------------
  MPI Msg Bytes%                            100.0%
  MPI Msg Bytes                   4,465,684,125.8
  MPI Msg Count                         13,057.0 msgs
  MsgSz <16 Count                         719.0 msgs
  16<= MsgSz <256 Count                    28.0 msgs
  256<= MsgSz <4KiB Count                   0.7 msgs
  4KiB<= MsgSz <64KiB Count               279.8 msgs
  64KiB<= MsgSz <1MiB Count             12,029.6 msgs
=================================================================
  MPI_Send
-----------------------------------------------------------------
  MPI Msg Bytes%                            100.0%
  MPI Msg Bytes                   4,465,680,353.8
  MPI Msg Count                         12,318.0 msgs
  MsgSz <16 Count                           8.0 msgs
  16<= MsgSz <256 Count                     0.0 msgs
  256<= MsgSz <4KiB Count                   0.7 msgs
  4KiB<= MsgSz <64KiB Count               279.8 msgs
  64KiB<= MsgSz <1MiB Count             12,029.6 msgs
```

```
MPI_Send / LAMMPS_NS::Comm::reverse_comm
-----------------------------------------------------------------
  MPI Msg Bytes%                             48.6%
  MPI Msg Bytes                   2,171,466,150.3
  MPI Msg Count                          6,006.0 msgs
  MsgSz <16 Count                            0.0 msgs
  16<= MsgSz <256 Count                      0.0 msgs
  256<= MsgSz <4KiB Count                    0.0 msgs
  4KiB<= MsgSz <64KiB Count                  0.0 msgs
  64KiB<= MsgSz <1MiB Count              6,006.0 msgs
=================================================================
  MPI_Send / LAMMPS_NS::Comm::reverse_comm / LAMMPS_NS::Verlet::run
-----------------------------------------------------------------
  MPI Msg Bytes%                             48.6%
  MPI Msg Bytes                   2,169,218,110.3
  MPI Msg Count                          6,000.0 msgs
  MsgSz <16 Count                            0.0 msgs
  16<= MsgSz <256 Count                      0.0 msgs
  256<= MsgSz <4KiB Count                    0.0 msgs
  4KiB<= MsgSz <64KiB Count                  0.0 msgs
  64KiB<= MsgSz <1MiB Count              6,000.0 msgs
=================================================================
```

# Maximize On-node Communication by Reordering MPI ranks

# When Is Rank Re-ordering Useful?

- Maximize on-node communication between MPI ranks
- **Physical system topology agnostic**
- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication
- Relieve on-node shared resource contention by pairing threads or processes that perform different work on the same node
  - for example: computation with off-node communication

# MPI Rank Reorder – Two Interfaces Available

- **CrayPat**
  - Available with sampling or tracing
  - Include –g mpi when instrumenting program
  - Run program and let CrayPat determine if communication is dominant, detect communication pattern and suggest MPI rank order if applicable

- **grid_order utility**
  - User knows communication pattern in application and wants to quickly create a new MPI rank placement file
  - Available when perftools-base module is loaded

# MPI Rank Order Observations

```
Table 1:  Profile by Function Group and Function

 Time%  |      Time  |     Imb.   |  Imb.   |  Calls   |Group
        |            |     Time   | Time%   |          | Function
        |            |            |         |          |  PE=HIDE

 100.0% | 463.147240 |        -- |     -- | 21621.0 |Total
|-------------------------------------------------------------------------
|  52.0% | 240.974379 |         -- |     -- | 21523.0 |MPI
||------------------------------------------------------------------------
||  47.7% | 221.142266 | 36.214468 |  14.1% | 10740.0 |mpi_recv
||   4.3% |  19.829001 | 25.849906 |  56.7% | 10740.0 |MPI_SEND
||========================================================================
|  43.3% | 200.474690 |         -- |     -- |    32.0 |USER
||------------------------------------------------------------------------
||  41.0% | 189.897060 | 58.716197 |  23.6% |    12.0 |sweep_
||   1.6% |   7.579876 |  1.899097 |  20.1% |    12.0 |source_
||========================================================================
|   4.7% |  21.698147 |         -- |     -- |    39.0 |MPI_SYNC
||------------------------------------------------------------------------
|   4.3% |  20.091165 | 20.005424 |  99.6% |    32.0 | mpi_allreduce_(sync)
||========================================================================
|   0.0% |   0.000024 |         -- |     -- |    27.0 |SYSCALL
|=========================================================================
```

# MPI Rank Order Observations (2)

```
MPI Grid Detection:

    There appears to be point-to-point MPI communication in a 96 X 8
    grid pattern. The 52% of the total execution time spent in MPI
    functions might be reduced with a rank order that maximizes
    communication between ranks on the same node. The effect of several
    rank orders is estimated below.

    A file named MPICH_RANK_ORDER.Grid was generated along with this
    report and contains usage instructions and the Custom rank order
    from the following table.

         Rank     On-Node      On-Node    MPICH_RANK_REORDER_METHOD
        Order    Bytes/PE    Bytes/PE%
                             of Total
                             Bytes/PE

       Custom  2.385e+09       95.55%   3
          SMP  1.880e+09       75.30%   1
         Fold  1.373e+06        0.06%   2
    RoundRobin 0.000e+00        0.00%   0
```

# Auto-Generated MPI Rank Order File

```
# The 'USER_Time_hybrid' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00023/malice/craypat/WORKSHOP/bh2o-demo/Rank/sweep3d/src/sweep3d
# Ap2 File:     sweep3d.gmpi-u.ap2
# Number PEs:   768
# Max PEs/Node: 16
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
0,532,64,564,32,572,96,540,8,596,72,524,40,604,24,588
104,556,16,628,80,636,56,620,48,516,112,580,88,548,120,612
1,403,65,435,33,411,97,443,9,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,113,491,49,387,89,451,121,483
6,436,102,468,70,404,38,412,14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,118,420,22,452,94,388,126,484
129,563,193,531,161,571,225,539,241,595,233,523,249,603,185,555
153,587,169,627,137,635,201,619,177,515,145,579,209,547,217,611
7,405,71,469,39,437,103,413,47,445,15,509,79,477,31,501
111,397,63,461,55,429,87,421,23,493,119,389,95,453,127,485
134,402,198,434,166,410,230,442,238,466,174,506,158,394,246,474
190,498,254,426,142,458,150,386,182,418,206,490,214,450,222,482
128,533,192,541,160,565,232,525,224,573,240,597,184,557,248,605
168,589,200,517,152,629,136,549,176,637,144,621,208,581,216,613
5,439,37,407,69,447,101,415,13,471,45,503,29,479,77,511
53,399,85,431,21,463,61,391,109,423,93,455,117,495,125,487
2,530,34,562,66,538,98,522,10,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,106,634,90,578,114,618,122,610
135,315,167,339,199,347,259,307,231,371,239,379,191,331,247,299
175,363,159,323,143,355,255,291,207,275,183,283,151,267,215,223
133,406,197,438,165,470,229,414,245,446,141,478,237,502,253,398
157,510,189,462,173,430,205,390,149,422,213,454,181,494,221,486
130,316,260,340,194,372,162,348,226,308,234,380,242,332,250,300
202,364,186,324,154,356,138,292,170,276,178,284,210,218,268,146
4,535,36,543,68,567,100,527,12,599,44,575,28,559,76,607
52,591,20,631,60,639,84,519,108,623,92,551,116,583,124,615
3,440,35,432,67,400,99,408,11,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,107,416,91,488,115,448,123,480
132,401,196,441,164,409,228,433,236,465,204,473,244,393,188,497
252,505,140,425,212,457,156,385,172,417,180,449,148,489,220,481
131,534,195,542,163,566,227,526,235,574,203,598,243,558,187,606
251,590,211,630,179,638,139,622,155,550,171,518,219,582,147,614
761,660,737,652,705,668,745,692,673,700,641,684,713,644,753,724
729,732,681,756,721,716,764,676,697,748,689,657,740,665,649,708
760,528,736,536,704,560,744,520,672,568,712,592,752,552,640,600
728,584,680,624,720,512,696,632,688,616,664,544,608,656,648,576
762,659,738,651,706,667,746,643,714,691,674,699,754,683,730,723
722,731,763,658,642,755,739,675,707,650,682,715,698,666,690,747
257,345,265,313,281,305,273,337,609,369,577,377,617,329,513,529
545,297,633,361,625,321,585,537,601,289,553,353,593,521,569,561
256,373,261,341,264,349,280,317,272,381,269,309,285,333,277,365
352,301,320,325,288,357,328,304,360,312,376,293,296,368,336,344
258,338,266,346,282,314,274,370,766,306,710,378,742,330,678,362
646,298,750,322,718,354,758,290,734,662,686,670,726,702,694,654
262,375,263,343,270,311,271,351,286,319,278,342,287,350,279,374
294,318,358,383,359,310,295,382,326,303,327,367,366,335,302,334
765,661,709,663,741,653,711,669,767,655,743,671,749,695,679,703
677,727,751,693,647,701,717,687,757,685,733,725,719,735,645,759
```

# Using New MPI Rank Order

- Save grid_order output to file called **MPICH_RANK_ORDER**
- `$ export MPICH_RANK_REORDER_METHOD=3`
- Run non-instrumented binary with and without new rank order to check overall wallclock time for performance improvements
- Can be used for all subsequent executions of **same job size**

# Summary

- Users continue to need tools to help find critical performance bottlenecks within a program
- Cray performance tools offer functionality that reduces the time investment associated with porting and tuning applications on new and existing Cray systems


CRAY
COMPUTE | STORE | ANALYZE
Arigatou
gozaimashita

# Legal Disclaimer

*Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.*

*Cray Inc. may make changes to specifications and product descriptions at any time, without notice.*

*All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.*

*Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.*

*Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.*

*Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.*

*The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.*