Cray Performance Tools

Heidi Poxon
Sr. Principal Engineer
Cray, Inc.

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 publicly 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.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. 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 on this website are the property of their respective owners.

Focus

- Improve your familiarity with the Cray performance tools
  - Add to your bag of tricks for analyzing code behavior

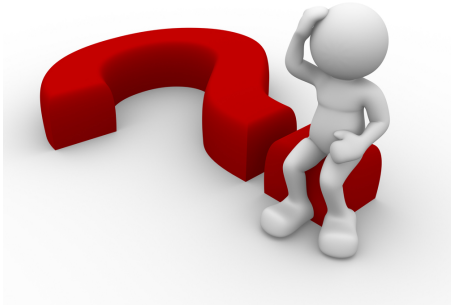
- Practice
  - The mechanics of using Cray performance tools
  - Learn how to identify problem areas and learn which experiment to use when

CRAY

Do Not Assume You Know Your Application Profile

General Profiling Tips

- Profile your *working* application
  - On the problem of interest at the scale of interest
  - Don’t think you know where the problem is and more importantly why it’s the most important bottleneck in the program
  - Performance on a single node is *not* necessarily representative of performance on 1000 nodes



- Which is dominant: computation or data movement and where?
- What size messages are frequently used in this program?
- Is the program suffering from load imbalance and if so, where?
- What is the percent of peak memory bandwidth achieved?
- Is there any insight from the tool on the performance data collected?

Cray Performance Tools

- Reduce the time investment associated with porting and tuning applications on Cray systems
- Analyze whole-program behavior across many nodes to identify critical performance bottlenecks within a program
- Improve profiling experience by using simple and/or advanced interfaces for a wealth of capability that targets analyzing the largest HPC jobs

Accessing perftools Software

- `perftools-base` provides access to pat_run, Reveal, Apprentice2, pat_report, and man pages without modification to applications

- perftools-base is often loaded by default (check your systems)

  - Load an instrumentation module to collect performance data

```
Examples:

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

Two Modes of Use

- CrayPat-lite: simple interface for convenience

Load module, build program, run, report generated to stdout

- CrayPat: advanced interface for in-depth performance investigation and tuning assistance

- Both offer:
  - Whole program analysis across many nodes
  - Indication of causes of problems
  - Ability to easily switch between the two interfaces

Cray Performance Tools have profiled production applications with over 256,000 MPI ranks

What About Different Compilers?

Cray Performance Tools support the following compilers

- Cray (CCE), Intel, GCC, and Arm Allinea compilers on Cray XC systems
- Cray (CCE) compiler on Cray CS systems

Using the Simple Interface

- `user@login> module load perftools-lite`

- Build program

- Run program

- View report sent to STDOUT (and .rpt file in experiment directory)
  - Example data directory: `stencil_order+49144-225s/`

Consolidated Performance Data

- Easily access performance data

- Unique directory name for each experiment

- Based on program name + unique number + ‘s’ or ‘t’
  - **vhone+73030-20s: program vhone with ‘s’ for sampling**
  - user@login> **pat_report *expdir* > *full_report***
  - user@login> **app2 vhone+73030-20s**

- Example directory:
  - user@login> **ls vhone+73030-20s**

```
ap2-files/  index.ap2  rpt-files/  xf-files/
```

Application Performance Information

- Job summary
- Application performance summary
- Profiles showing slowest code in application
- Key bottlenecks such as load imbalance
- Memory footprint and bandwidth
- Time spent performing I/O and Lustre file information
- Application energy and power usage
- Observations about application behavior

Example: perftools-lite Job Summary

```
CrayPat/X: Version 7.0.1 Revision 3714888 03/07/18 02:11:13
Experiment:                  lite lite/sample_profile
Number of PEs (MPI ranks):    36
Numbers of PEs per Node:      36
Numbers of Threads per PE:     1
Number of Cores per Socket:   18
Execution start time: Thu Mar 15 11:14:05 2018
System name and speed: nid00030 2.101 GHz (nominal)
Intel Broadwell CPU Family: 6 Model: 79 Stepping: 1

Avg Process Time:                    3.70 secs
High Memory:                     1,801.4 MBytes   50.0 MBytes per PE
Observed CPU clock boost:          117.2 %
Percent cycles stalled:             38.3 %
Vector intensity (packed instr):     2.6 %
Instr per Cycle:                     1.51
I/O Read Rate:                   3.676263 MBytes/sec
I/O Write Rate:                  0.293086 MBytes/sec
```

Example: perftools-lite Top Time Consumers

```
Table 1:  Profile by Function Group and Function (top 10 functions shown)
  Samp% |     Samp |    Imb. |  Imb. |Group
        |          |    Samp | Samp% | Function
        |          |         |       |  PE=HIDE
 100.0% | 55,605.7 |      -- |    -- |Total
|-------------------------------------------------------------------------
|  56.5% | 31,412.8 |      -- |    -- |USER
||------------------------------------------------------------------------
||  19.7% | 10,944.1 |   290.9 |  2.6% |create_boundary$boundary_
||  10.7% |  5,937.8 |   214.2 |  3.5% |get_block$blocks_
||   3.9% |  2,194.4 |     7.6 |  0.3% |create_distrb_balanced$distribution_
||   2.0% |  1,135.5 |   137.5 | 10.8% |impvmixt$vertical_mix_
||   1.9% |  1,064.8 |   124.2 | 10.5% |impvmixt_correct$vertical_mix_
||========================================================================
|  22.5% | 12,513.4 |      -- |    -- |ETC
||------------------------------------------------------------------------
||  20.1% | 11,151.4 | 2,758.6 | 19.9% |__cray_memcpy_KNL
||========================================================================
|  20.7% | 11,503.5 |      -- |    -- |MPI
||------------------------------------------------------------------------
||  11.1% |  6,171.6 | 1,785.4 | 22.5% |MPI_ALLREDUCE
||   7.9% |  4,377.8 | 3,254.2 | 42.7% |mpi_waitall
```

Example: perftools-lite Observations

```
MPI Grid Detection:
    There appears to be point-to-point MPI communication in a 32 X 32
    grid pattern. The 20.7% 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 Hilbert rank order
    from the following table.

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

       Hilbert  1.413e+12     81.94%  3
           SMP  1.053e+12     61.04%  1
          Fold  9.405e+11     54.53%  2
    RoundRobin  8.962e+11     51.96%  0
```

Example: perftools-lite Hot Spots by Line

```
Table 3:  Profile by Group, Function, and Line
  Samp% |      Samp |     Imb. |  Imb. | Group
        |           |     Samp | Samp% |  Function
        |           |          |       |   Source
        |           |          |       |    Line
        |           |          |       |     PE=HIDE
 100.0% | 60,665.8 |       -- |    -- | Total
|-----------------------------------------------------------------------
|  94.6% | 57,390.6 |       -- |    -- | USER
||----------------------------------------------------------------------
||  82.1% | 49,835.3 |       -- |    -- | LAMMPS_NS::PairLJCut::compute
|||---------------------------------------------------------------------
3||  80.7% | 48,970.1 |       -- |    -- | src/Obj_xc30intel/../pair_lj_cut.cpp
||||--------------------------------------------------------------------
4|||   3.9% |  2,359.8 |    100.2 |  4.1% | line.102
4|||   1.0% |    596.2 |     61.8 |  9.5% | line.105
4|||   8.3% |  5,022.4 |    683.6 | 12.1% | line.107
4|||   2.9% |  1,744.2 |    966.8 | 36.0% | line.108
```

Example: File I/O Information

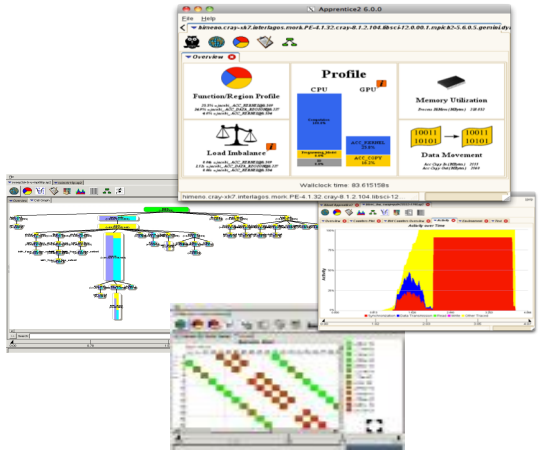
```
Table 5:  File Input Stats by Filename
     Read |     Read |   Read Rate |   Reads | Bytes/ | File Name=!x/^/(proc|sys)/
     Time |  MiBytes | MiBytes/sec |         |   Call |  PE=HIDE

 0.249811 | 0.686758 |    2.749114 | 6,338.0 | 113.62 | Total
|-------------------------------------------------------------------------------
| 0.225211 | 0.122314 |    0.543111 |   256.0 | 501.00 | clover.in.tmp
| 0.003741 | 0.003090 |    0.826022 |   220.0 |  14.73 | clover.in
|===============================================================================
```

Example: Lustre File Information

Table 7: Lustre File Information

File Path	Stripe size	Stripe offset	Stripe count	OST list
rationals_m001m05m5.test1	1,048,576	0	8	2,4,0,6,3,7,5,1
36x36x36x72.chklat	1,048,576	0	8	1,2,4,0,6,3,7,5

Data Movement



Focus on MPI Communication Bottlenecks

- `user@login> module load perftools`
- Build program
  - It's helpful to add **`–hlist=a`** when building with CCE for listing
- Instrument program, only focusing on MPI
  - `user@login>` **`pat_build –g mpi ./my_program`**
- Run instrumented program (my_program+pat)
- Create report
  - `user@login>` **`pat_report my_data_directory+12t/ > my_report`**

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).
```

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
```

Analyzing 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

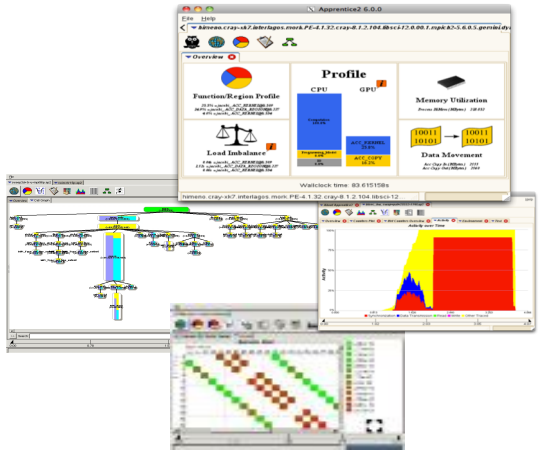
MPI_Send / LAMMPS_NS::Comm::reverse_comm

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

Load Imbalance

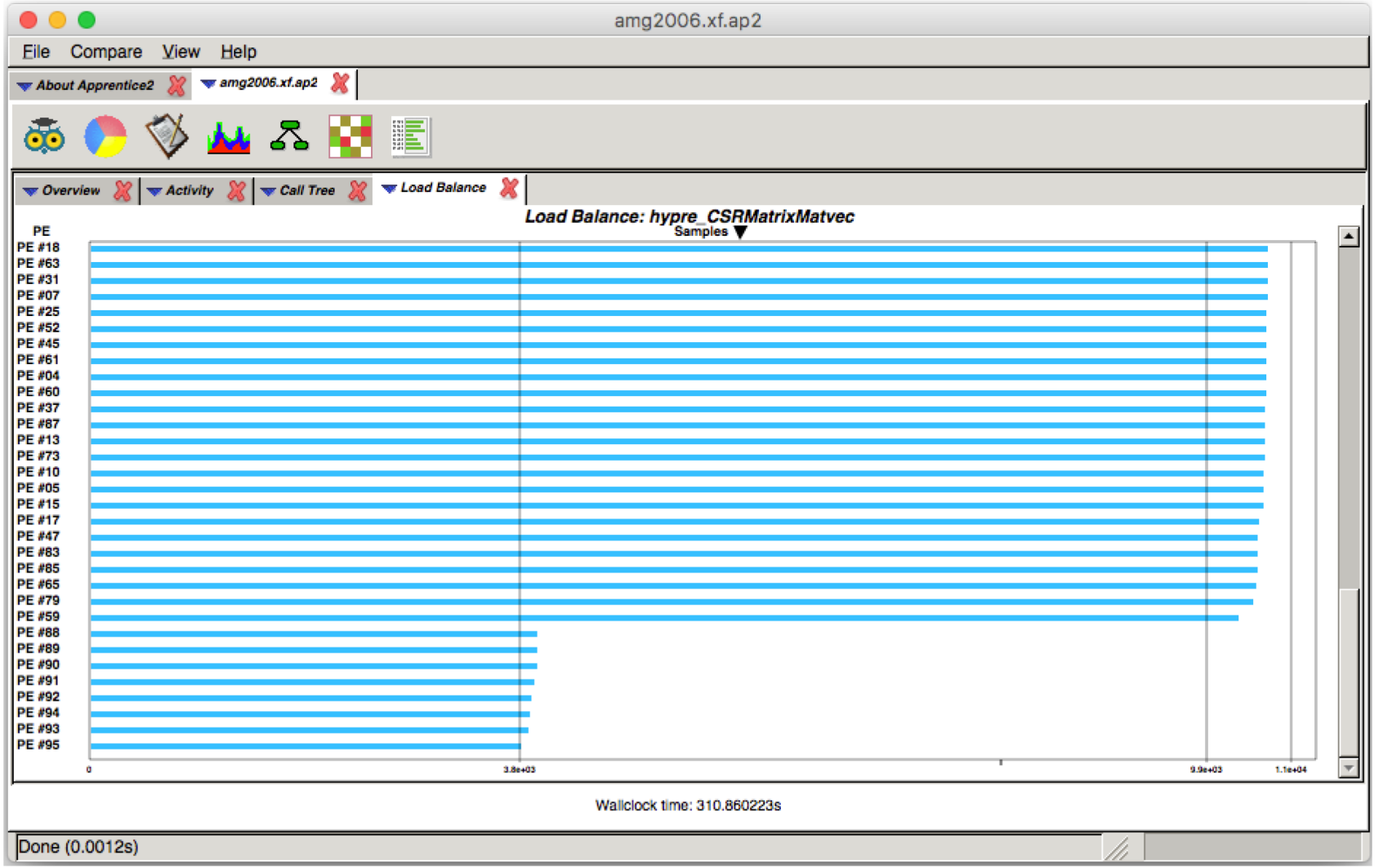


Finding Program Load Imbalance

Table 1: Profile by Function Group and Function

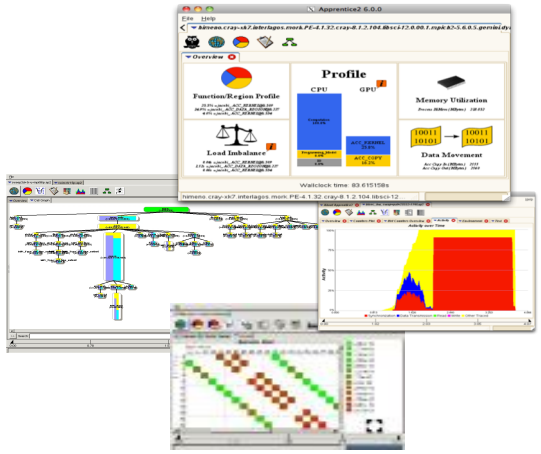
Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE
100.0%	1.957703	--	--	42,970.8	Total
60.0%	1.174021	--	--	3,602.0	USER
30.8%	0.603850	0.176924	23.0%	1,198.0	function3_
19.2%	0.375117	0.128748	26.0%	1,200.0	function2_
9.1%	0.178111	0.081880	32.0%	1,200.0	function1_
36.0%	0.704928	--	--	9,613.0	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

Visualizing Load Imbalance

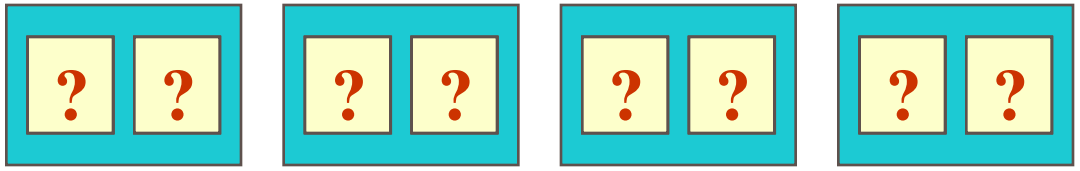


Visualizing Load Imbalance (2)

MPI Rank Reordering



MPI Rank Placement Strategies



- Distributed placement
- SMP style placement
- Folded rank placement
- User provided rank file

When Is Rank Re-ordering Useful?

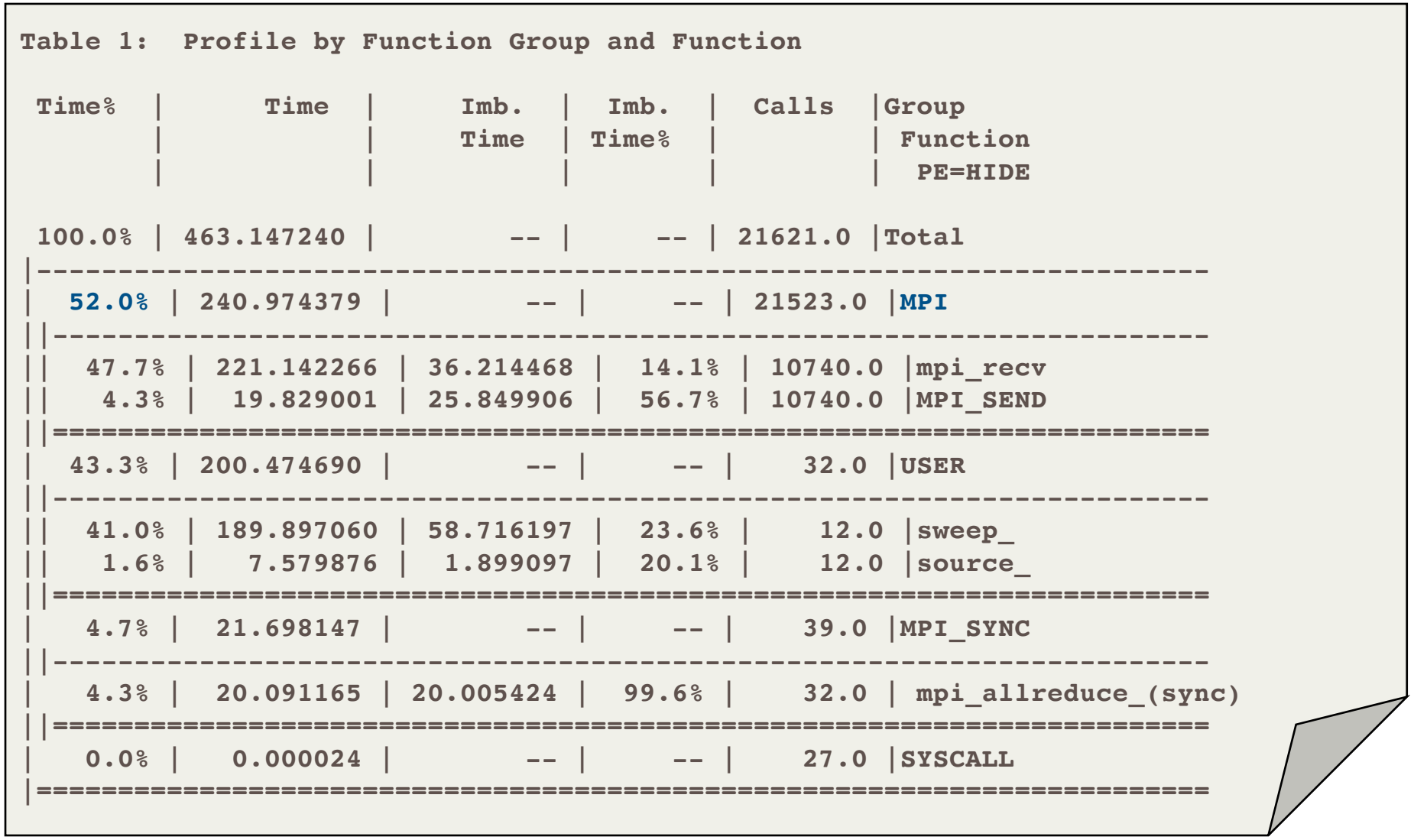
- Maximize on-node communication between MPI ranks
- Physical system topology agnostic
- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication
- Relieve on-node shared resource contention by pairing threads or processes that perform different work on 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
,48,516,112,580,88,548,120,6
12
1,403,65,435,33,411,97,443,9
,467,25,499,105,507,41,475
73,395,81,427,57,459,17,419,
113,491,49,387,89,451,121,48
3
6,436,102,468,70,404,38,412,
14,444,46,476,110,508,78,500
86,396,30,428,62,460,54,492,
118,420,22,452,94,388,126,48
4
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
,23,493,119,389,95,453,127,4
85
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,
109,423,93,455,117,495,125,4
87
2,530,34,562,66,538,98,522,1
0,570,42,554,26,594,50,602
18,514,74,586,58,626,82,546,
106,634,90,578,114,618,122,6
10
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,
108,623,92,551,116,583,124,6
15
3,440,35,432,67,400,99,408,1
1,464,43,496,27,472,51,504
19,392,75,424,59,456,83,384,
107,416,91,488,115,448,123,4
80
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

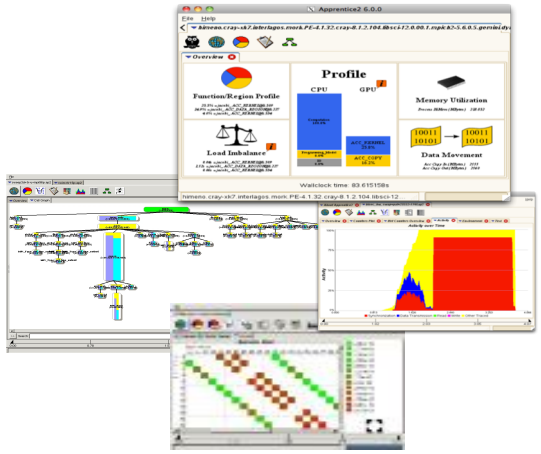
- Save grid_order output to file called MPICH_RANK_ORDER

- `$ 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

Documentation



Documentation

- Release Notes
  - `user@login> module help perftools-base/`*`version_number`*
- User manual “Using the Cray Performance Measurement and Analysis Tools” available at http://pubs.cray.com
- pat_help – interactive help utility on the Cray Performance toolset
- Man pages

Man Pages

- **intro_craypat**(1)
  - Introduces the craypat performance tool
  - Runtime environment variables (enable full trace, etc.)
- **pat_build(1)**
  - Instrument a program for performance analysis
- **pat_help(1)**
  - Interactive online help utility
- **pat_report(1)**
  - Generate performance report in both text and for use with GUI

Summary

- Cray performance tools offer functionality that reduces the time investment associated with porting and tuning applications on new and existing Cray systems

- Cray performance tools come with a simple interface plus a wealth of capability when you need it for analyzing those most critical production codes

QUESTIONS?