Cray Performance Measurement and Analysis Tools

Heidi Poxon
Manager & Technical Lead, Performance Tools
Cray Inc.

7-9 November, 2011

Swiss National Supercomputing Centre

More on Cray Performance Tools

- Analysis assistance
  - load imbalance
  - automatic grid detection
  - loop work estimates
- Other interesting performance statistics

Load Imbalance Analysis

Performance Measurement and Analysis

- Load imbalance

  - Identifies computational code regions and synchronization calls that could benefit most from load balance optimization (some processes have less work than others, some are waiting longer on barriers, etc)

  - Estimates savings if corresponding section of code were balanced

  - MPI sync time (determines late arrivers to barriers)
  - MPI rank placement suggestions (maximize on-node communication)
  - Imbalance metrics (user functions, MPI functions, OpenMP threads)

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

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
- If desired, PAT_RT_MPI_SYNC=0 deactivates this feature

Imbalance Time

- Metric based on execution time
- It is dependent on the type of activity:
  - User functions

    Imbalance time = Maximum time – Average time
  - Synchronization (Collective communication and barriers)

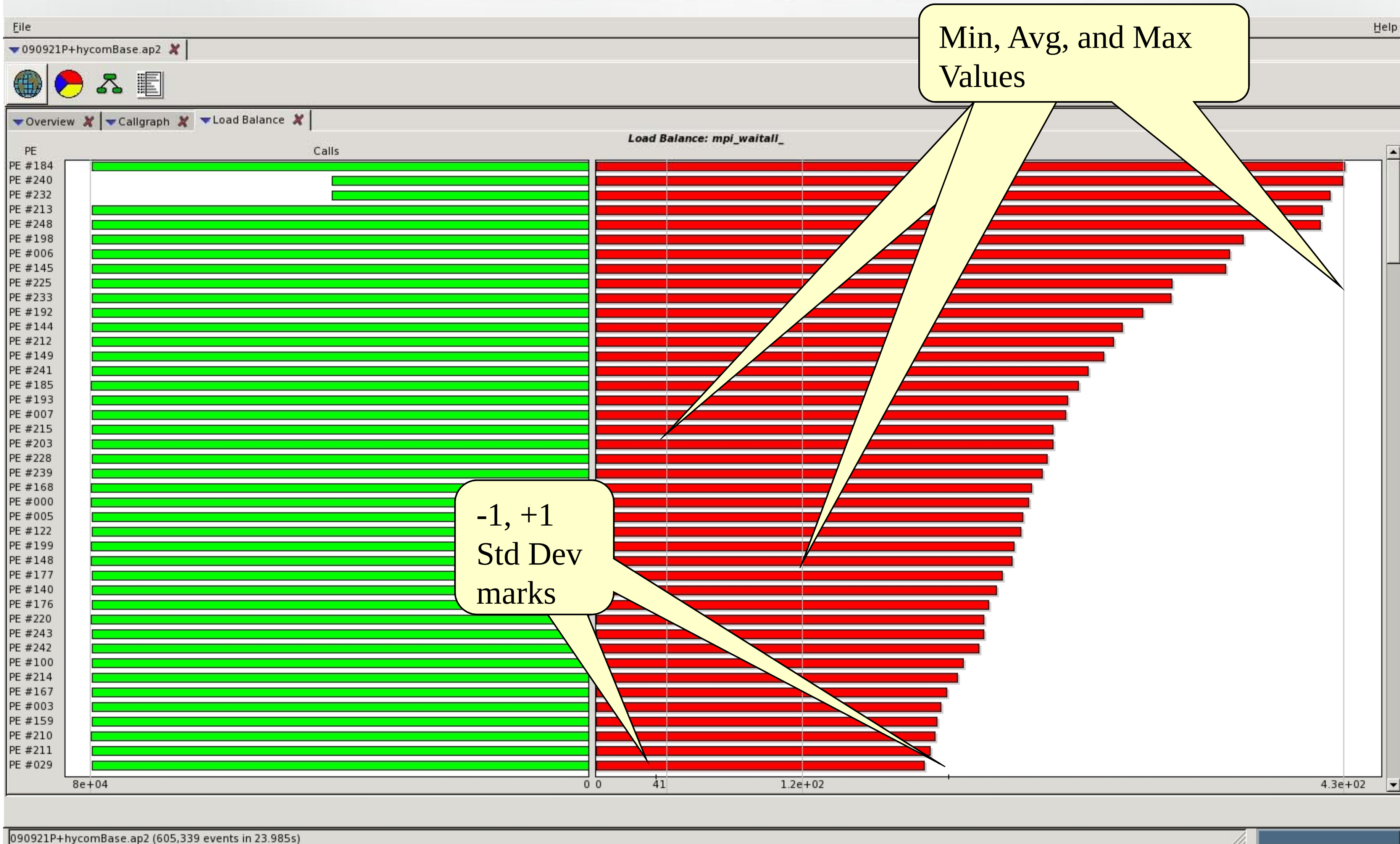
    Imbalance time = Average time – Minimum time
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
- Estimates how much overall program time could be saved if corresponding section of code had a perfect balance
  - Represents upper bound on “potential savings”
  - Assumes other processes are waiting, not doing useful work while slowest member finishes

Imbalance %

$$\text{Imbalance\%} = 100 \text{ X } \frac{\text{Imbalance time}}{\text{Max Time}} \text{ X } \frac{N}{N - 1}$$

- Represents % of resources available for parallelism that is "wasted"
- Corresponds to % of time that rest of team is not engaged in useful work on the given function
- Perfectly balanced code segment has imbalance of 0%
- Serial code segment has imbalance of 100%

Load Distribution



MPI Rank Placement Suggestions

Automatic Communication Grid Detection

- Analyze runtime performance data to identify grids in a program to maximize on-node communication
  - Example: nearest neighbor exchange in 2 dimensions
    - Sweep3d uses a 2-D grid for communication
- Determine whether or not a custom MPI rank order will produce a significant performance benefit
- Grid detection is helpful for programs with significant point-to-point communication
- Doesn’t interfere with MPI collective communication optimizations

Automatic Grid Detection (cont'd)

- Tools produce a custom rank order if it's beneficial based on grid size, grid order and cost metric
- Summarized findings in report
- Available if MPI functions traced (-g mpi)
- Describe how to re-run with custom rank order

Example: Observations and Suggestions

```
MPI Grid Detection:  There appears to be point-to-point MPI
     communication in a 22 X 18 grid pattern. The 48.6% 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.Custom was generated along with this
     report and contains the Custom rank order from the following table.
     This file also contains usage instructions and a table of
     alternative rank orders.

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

        Custom    7.80e+06       78.37%   3
           SMP    5.59e+06       56.21%   1
          Fold    2.59e+05        2.60%   2
    RoundRobin    0.00e+00        0.00%   0
```

MPICH_RANK_ORDER File Example

```
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:     sweep3d.mpi+pat+27054-89t.ap2
# Number PEs:   48
# Max PEs/Node: 4
#
# 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.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
…
```

Example 2 - Hycom

```
================  Observations and suggestions  ========================
MPI grid detection:
    There appears to be point-to-point MPI communication in a 33 X 41
    grid pattern. The 26.1% 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.Custom was generated along with this
    report and contains the Custom rank order from the following table.
    This file also contains usage instructions and a table of
    alternative rank orders.

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

        Custom    1.20e+09      32.21%  3
           SMP    8.70e+08      23.27%  1
          Fold    3.55e+07       0.95%  2
    RoundRobin    1.99e+05       0.01%  0
================  End Observations  ====================================
```

Example 2 - Hycom

- Run on 1353 MPI ranks, 24 ranks per node

- Overall program wallclock:
  - Default MPI rank order: 1450s
  - Custom MPI rank order: 1315s
  - ~10% improvement in execution time!

- Time spent in MPI routines:
  - Default rank order: 377s
  - Custom rank order: 303s

Loop Work Estimates

Loop Work Estimates

- Helps identify loops to optimize (parallelize serial loops):
  - Loop timings approximate how much work exists within a loop
  - Trip counts can be used to help carve up loop on GPU
- Enabled with CCE –h profile_generate option
  - Should be done as separate experiment – compiler optimizations are restricted with this feature
- Loop statistics reported by default in pat_report table
- Next enhancement: integrate loop information in profile
  - Get exclusive times and loops attributed to functions

Collecting Loop Statistics

- Load PrgEnv-cray software
- Load perftools software

- Compile AND link with –h profile_generate

- Instrument binary for tracing
  - pat_build –u my_program or
  - pat_build –w my_program

- Run application
- Create report with loop statistics
  - pat_report my_program.xf > loops_report

Example Report – Loop Work Estimates

```
Table 1:  Profile by Function Group and Function
 Time%   |       Time |      Imb. |  Imb.  |   Calls  |Group
         |            |      Time | Time%  |          | Function
         |            |           |        |          |  PE=HIDE
         |            |           |        |          |   Thread=HIDE

 100.0% | 176.687480 |        -- |     -- | 17108.0 |Total
|-----------------------------------------------------------------------
|  85.3% | 150.789559 |        -- |     -- |      8.0 |USER
||----------------------------------------------------------------------
|  85.0% | 150.215785 | 24.876709 |  14.4% |      2.0 | jacobi_.LOOPS
||======================================================================
|  12.2% |  21.600616 |        -- |     -- | 16071.0 |MPI
||----------------------------------------------------------------------
|  11.9% |  21.104488 | 41.016738 |  67.1% |  3009.0 | mpi_waitall
||======================================================================
|   2.4% |   4.297301 |        -- |     -- |  1007.0 |MPI_SYNC
||----------------------------------------------------------------------
|   2.4% |   4.166092 |  4.135016 |  99.3% |  1004.0 | mpi_allreduce_(sync)
|=======================================================================
```

Example Report – Loop Work Estimates (2)

```
Table 3:  Inclusive Loop Time from -hprofile_generate

 Loop Incl  |     Loop  |  Loop  |  Loop   |Function=/.LOOP[.]
      Time  |      Hit  | Trips  | Trips   | PE=HIDE
     Total  |           |   Min  |   Max   |
|-----------------------------------------------------------------
…
| 175.676881 |        2 |      0 |    1003 |jacobi_.LOOP.07.li.267
|   0.917107 |     1003 |      0 |     260 |jacobi_.LOOP.08.li.276
|   0.907515 |   129888 |      0 |     260 |jacobi_.LOOP.09.li.277
|   0.446784 |     1003 |      0 |     260 |jacobi_.LOOP.10.li.288
|   0.425763 |   129888 |      0 |     516 |jacobi_.LOOP.11.li.289
|   0.395003 |     1003 |      0 |     260 |jacobi_.LOOP.12.li.300
|   0.374206 |   129888 |      0 |     516 |jacobi_.LOOP.13.li.301
| 126.250610 |     1003 |      0 |     256 |jacobi_.LOOP.14.li.312
| 126.223035 |   127882 |      0 |     256 |jacobi_.LOOP.15.li.313
| 124.298650 | 16305019 |      0 |     512 |jacobi_.LOOP.16.li.314
|  20.875086 |     1003 |      0 |     256 |jacobi_.LOOP.17.li.336
|  20.862715 |   127882 |      0 |     256 |jacobi_.LOOP.18.li.337
|  19.428085 | 16305019 |      0 |     512 |jacobi_.LOOP.19.li.338
|=========================================================================
```

Other Interesting Performance Data

Program Instrumentation – Sampling

- Sampling is useful to determine where the program spends most of its time (functions and lines)
- The environment variable PAT_RT_EXPERIMENT allows the specification of the type of experiment prior to execution
  - samp_pc_time (default)
    - Samples the PC at intervals of 10,000 microseconds
    - Measures user CPU and system CPU time
    - Returns total program time and absolute and relative times each program counter was recorded
    - Optionally record the values of hardware counters specified with PAT_RT_HWPC
  - samp_pc_ovfl
    - Samples the PC at a given overflow of a HW counter
    - Does not allow collection of hardware counters
  - samp_cs_time
    - Sample the call stack at a given time interval

-g tracegroup (subset)

- blas Basic Linear Algebra subprograms
- CAF Co-Array Fortran (Cray CCE compiler only)
- HDF5manages extremely large and complex data collections
- heap dynamic heap
- io includes stdio and sysio groups
- lapack Linear Algebra Package
- math ANSI math
- mpi MPI
- omp OpenMP API
- omp-rtl OpenMP runtime library (not supported on Catamount)
- pthreads POSIX threads (not supported on Catamount)
- shmem SHMEM
- sysio I/O system calls
- system system calls
- upc Unified Parallel C (Cray CCE compiler only)

For a full list, please see man pat_build

Specific Tables in pat_report

```
heidi@kaibab:/lus/scratch/heidi> pat_report -O –h

pat_report: Help for -O option:
Available option values are in left column, a prefix can be
specified:
  ct                          -O calltree
  defaults                    <Tables that would appear by default.>
  heap                        -O heap_program,heap_hiwater,heap_leaks
  io                          -O read_stats,write_stats
  lb                          -O load_balance
  load_balance                -O lb_program,lb_group,lb_function
  mpi                         -O mpi_callers
  ---
  D1_D2_observation           Observation about Functions with low
D1+D2 cache hit ratio
  D1_D2_util                  Functions with low D1+D2 cache hit ratio
  D1_observation              Observation about Functions with low D1
cache hit ratio
  D1_util                     Functions with low D1 cache hit ratio
  TLB observation             Observation about Functions with low TLB
refs/miss
  TLB_util                    Functions with low TLB refs/miss
```

Heap Statistics

- -g heap
  - calloc, cfree, malloc, free, malloc_trim, malloc_usable_size, mallopt, memalign, posix_memalign, pvalloc, realloc, valloc

- -g heap
- -g sheap
- -g shmem
  - shfree, shfree_nb, shmalloc, shmalloc_nb, shrealloc

- -g upc (automatic with –O apa)
  - upc_alloc, upc_all_alloc, upc_all_free, uc_all_lock_alloc, upc_all_lock_free, upc_free, upc_global_alloc, upc_global_lock_alloc, upc_lock_free

Heap Statistics

```
Notes for table 5:

  Table option:
    -O heap_hiwater
  Options implied by table option:
    -d am@,ub,ta,ua,tf,nf,ac,ab -b pe=[mmm]

  This table shows only lines with Tracked Heap HiWater MBytes >


Table 5:  Heap Stats during Main Program

 Tracked |  Total | Total | Tracked | Tracked |PE[mmm]
    Heap | Allocs | Frees | Objects |  MBytes |
 HiWater |        |       |     Not |     Not |
  MBytes |        |       |   Freed |   Freed |

   9.794 |    915 |   910 |       4 |   1.011 |Total
|-----------------------------------------------------
|   9.943 |   1170 |  1103 |      68 |   1.046 |pe.0
|   9.909 |    715 |   712 |       3 |   1.010 |pe.22
|   9.446 |   1278 |  1275 |       3 |   1.010 |pe.43
|=====================================================
```

CrayPat API - For Fine Grain Instrumentation

- Fortran

```
include “pat_apif.h”
…
call PAT_region_begin(id, “label”, ierr)
do i = 1,n
…
enddo
call PAT_region_end(id, ierr)
```

- C & C++

```
include <pat_api.h>
…
ierr = PAT_region_begin(id, “label”);
< code segment >
ierr = PAT_region_end(id);
```

PGAS (UPC, CAF) Support

PGAS Support

- Profiles of a PGAS program can be created to show:
  - Top time consuming functions/line numbers in the code
  - Load imbalance information
  - Performance statistics attributed to user source by default
  - Can expose statistics by library as well
    - To see underlying operations, such as wait time on barriers

- Data collection is based on methods used for MPI library
  - PGAS data is collected by default when using Automatic Profiling Analysis (pat_build –O apa)
  - Predefined wrappers for runtime libraries (caf, upc, pgas) enable attribution of samples or time to user source

- UPC and SHMEM heap tracking available
  - `-g heap` will track shared heap in addition to local heap

PGAS Default Report Table 1

```
Table 1:  Profile by Function

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

 100.0% |   48 |   -- |     -- |Total
|-------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||------------------------------------------
||  83.3% |   40 | 1.00 |   3.3% |all2all
||   6.2% |    3 | 0.50 |  22.2% |do_cksum
||   2.1% |    1 | 1.00 |  66.7% |do_all2all
||   2.1% |    1 | 0.50 |  66.7% |mpp_accum_long
||   2.1% |    1 | 0.50 |  66.7% |mpp_alloc
||==========================================
|   4.2% |    2 |   -- |     -- |ETC
||------------------------------------------
||   4.2% |    2 | 0.50 |  33.3% |bzero
|===========================================
```

PGAS Default Report Table 2

```
Table 2:  Profile by Group, Function, and Line

 Samp % | Samp | Imb. |   Imb. |Group
        |      | Samp | Samp % | Function
        |      |      |        |  Source
        |      |      |        |   Line
        |      |      |        |    PE='HIDE'

 100.0% |   48 |   -- |     -- |Total
|--------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||-------------------------------------------
||  83.3% |   40 |   -- |     -- |all2all
3|        |      |      |        | mpp_bench.c
4|        |      |      |        |  line.298
||   6.2% |    3 |   -- |     -- |do_cksum
3|        |      |      |        | mpp_bench.c
||||-----------------------------------------
4|||   2.1% |    1 | 0.25 |  33.3% |line.315
4|||   4.2% |    2 | 0.25 |  16.7% |line.316
||||=========================================
```

PGAS Report Showing Library Functions with Callers

```
Table 1:  Profile by Function and Callers, with Line Numbers
 Samp % | Samp |Group
        |      | Function
        |      |  Caller
        |      |   PE='HIDE'
 100.0% |   47 |Total
|-------------------------
|  93.6% |   44 |ETC
||-------------------------
||  85.1% |   40 |upc_memput
3|        |      | all2all:mpp_bench.c:line.298
4|        |      |  do_all2all:mpp_bench.c:line.348
5|        |      |   main:test all2all.c:line.70
||   4.3% |    2 |bzero
3|        |      | (N/A):(N/A):line.0
||   2.1% |    1 |upc_all_alloc
3|        |      | mpp_alloc:mpp_bench.c:line.143
4|        |      |  main:test_all2all.c:line.25
||   2.1% |    1 |upc_all_reduceUL
3|        |      | mpp_accum_long:mpp_bench.c:line.185
4|        |      |  do_cksum:mpp_bench.c:line.317
5|        |      |   do_all2all:mpp_bench.c:line.341
6|        |      |    main:test_all2all.c:line.70
||========================
```

OpenMP Support

OpenMP Data Collection and Reporting

- Measure overhead incurred entering and leaving
  - Parallel regions
  - Work-sharing constructs within parallel regions
- Show per-thread timings and other data
- Trace entry points automatically inserted by Cray and PGI compilers
  - Provides per-thread information
- Can use sampling to get performance data without API (per process view… no per-thread counters)
  - Run with OMP_NUM_THREADS=1 during sampling
  - Watch for calls to omp_set_num_threads()

OpenMP Data Collection and Reporting (2)

- Load imbalance calculated across all threads in all ranks for mixed MPI/OpenMP programs
  - Can choose to see imbalance to each programming model separately
- Data displayed by default in pat_report (no options needed)
  - Focus on where program is spending its time
  - Assumes all requested resources should be used

Imbalance Options for Data Display (pat_report –O ...)

- profile_pe.th (default view)
  - Imbalance based on the set of all threads in the program
- profile_pe_th
  - Highlights imbalance across MPI ranks
  - Uses max for thread aggregation to avoid showing under-performers
  - Aggregated thread data merged into MPI rank data
- profile_th_pe
  - For each thread, show imbalance over MPI ranks
  - Example: Load imbalance shown where thread 4 in each MPI rank didn't get much work

Profile by Function Group and Function (with –T)

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

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

 100.0% | 12.548996 |       -- |     -- |   7944.7 |Total
|-------------------------------------------------------------------
|  97.8% | 12.277316 |       -- |     -- |   3371.8 |USER
||------------------------------------------------------------------
||  35.6% |  4.473536 | 0.072259 |   1.6% |    498.0 |calc3 .LOOP@li.96
||  29.1% |  3.653288 | 0.070551 |   1.9% |    500.0 |calc2_.LOOP@li.74
||  28.3% |  3.545677 | 0.056303 |   1.6% |    500.0 |calc1_.LOOP@li.69
. . .
||==================================================================
|   1.2% |  0.155028 |       -- |     -- |   1000.5 |MPI_SYNC
||------------------------------------------------------------------
||   1.2% |  0.154899 | 0.674518 |  82.0% |    999.0 |mpi barrier (sync)
||   0.0% |  0.000129 | 0.000489 |  79.8% |      1.5 |mpi_reduce_(sync)
||==================================================================
|   0.7% |  0.082943 |       -- |     -- |   3197.2 |MPI
||------------------------------------------------------------------
||   0.4% |  0.047471 | 0.158820 |  77.6% |    999.0 |mpi_barrier_
||   0.1% |  0.015157 | 0.295055 |  95.9% |    297.1 |mpi_waitall_
. . .
||==================================================================
|   0.3% |  0.033683 |       -- |     -- |    374.5 |OMP
||------------------------------------------------------------------
||   0.1% |  0.013098 | 0.078620 |  86.4% |    125.0 |calc2_.REGION@li.74(ovhd)
||   0.1% |  0.010298 | 0.052760 |  84.3% |    124.5 |calc3_.REGION@li.96(ovhd)
||   0.1% |  0.010287 | 0.068428 |  87.6% |    125.0 |calc1 .REGION@li.69(ovhd)
||==================================================================
|   0.0% |  0.000027 | 0.000128 |  83.0% |      0.8 |PTHREAD
|        |           |          |        |          | pthread_create
|===================================================================
```

OpenMP Parallel DOs <function>.<region>@<line> automatically instrumented

OpenMP overhead is normally small and is filtered out on the default report (< 0.5%). When using "–T" the filter is deactivated

Hardware Counters Information at Loop Level

```
========================================================================
USER / calc3_.LOOP@li.96
------------------------------------------------------------------------
  Time%                                                  37.3%
  Time                                                6.826587 secs
  Imb.Time                                            0.039858 secs
  Imb.Time%                                               0.6%
  Calls                                72.9 /sec         498.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED           64.364M/sec     439531950 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                              10.760M/sec      73477950 fills
  PAPI_L1_DCM                        64.973M/sec     443686857 misses
  PAPI_L1_DCA                       135.699M/sec     926662773 refs
  User time (approx)                  6.829 secs   15706256693 cycles  100.0%Time
  Average Time per Call                               0.013708 sec
  CrayPat Overhead : Time              0.0%
  D1 cache hit,miss ratios            52.1% hits         47.9% misses
  D1 cache utilization (misses)        2.09 refs/miss    0.261 avg hits
  D1 cache utilization (refills)       1.81 refs/refill  0.226 avg uses
  D2 cache hit,miss ratio             85.7% hits         14.3% misses
  D1+D2 cache hit,miss ratio          93.1% hits          6.9% misses
  D1+D2 cache utilization             14.58 refs/miss    1.823 avg hits
  System to D1 refill                10.760M/sec      73477950 lines
  System to D1 bandwidth            656.738MB/sec   4702588826 bytes
  D2 to D1 bandwidth               3928.490MB/sec  28130044826 bytes
========================================================================
```

Caveats

- No support for nested parallel regions
  - To work around this until addressed disable nested regions by setting OMP_NESTED=0
  - Watch for calls to omp_set_nested()

- If compiler merges 2 or more parallel regions, OpenMP trace points are not merged correctly
  - To work around this until addressed, use –h thread1

- We need to add tracing support for barriers (both implicit and explicit)
  - Need support from compilers

Try Adding OpenMP to an MPI Code When…

- When code is network bound
  - Look at collective time, excluding sync time: this goes up as network becomes a problem
  - Look at point-to-point wait times: if these go up, network may be a problem
- When MPI starts leveling off
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue
- Adding OpenMP to memory bound codes may aggravate memory bandwidth issues, but you have more control when optimizing for cache

Questions

??