CUG 2008 HELSINKI • MAY 5–8, 2008
CROSSING THE BOUNDARIES

Detecting Application Load Imbalance on Cray Systems

Heidi Poxon
Technical Lead, Performance Tools

Cray Inc.

CRAY
THE SUPERCOMPUTER COMPANY

Outline

-
- Cray Performance Tools Overview
- Motivation for Load Imbalance Analysis
- Metrics Offered by Cray Performance Tools
- Examples

Cray Performance Tools Overview

-
- CrayPat
  - Instrumentation of optimized code
  - No source code modification required
  - Data collection transparent to the user
  - Text-based performance reports
  - Derived metrics
  - Performance analysis
- Cray Apprentice2
  - Performance data visualization tool
  - Call tree view
  - Time line view
  - Source code mappings

Motivation for Load Imbalance Analysis

-
- Increasing system software and architecture complexity
- Systems are scaling to tens of thousands of processors
- 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

CrayPat Load Imbalance Support

-
- Imbalance time and %
- MPI sync time
- OpenMP Performance Metrics
- MPI rank placement suggestions

Imbalance Time

Imbalance time = Maximum time – Average time

- Metric based on execution times
- Identifies computational code regions 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 %

- 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%

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

How to Collect and View Time and % Metrics

- Metrics calculated by default
  - Level depends on Instrumentation chosen
- Available with sampling or event trace
- Statistics available by default in text report
- Options to focus load balance information in report by
  - Whole program
  - Group
  - Function
  - MPI Sent Message Statistics
- Visualize imbalance through Cray Apprentice2

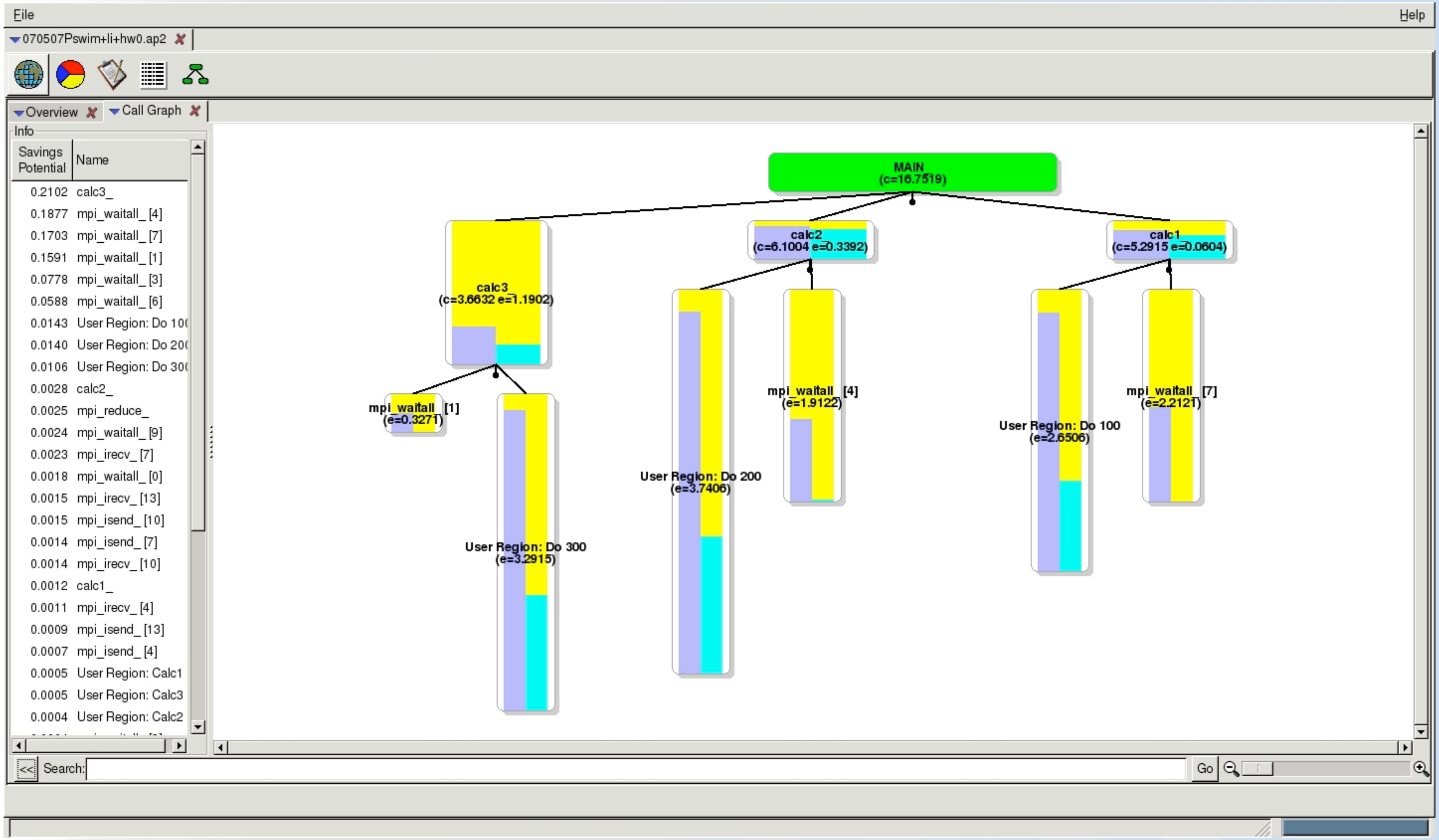
Profile with Load Distribution by Groups

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

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

 100.0% | 0.482144 |       -- |      -- |  2530 |Total
|----------------------------------------------------------
|  83.7% | 0.403314 |       -- |      -- |   303 |USER
||---------------------------------------------------------
||  32.4% | 0.156028 | 0.009882 |    6.8% |     98 |calc3_
||  27.7% | 0.133643 | 0.007400 |    6.0% |    100 |calc2_
||  21.0% | 0.101406 | 0.002552 |    2.8% |    100 |calc1_
||   2.0% | 0.009696 | 0.000287 |    3.3% |      1 |inital_
||=========================================================
|  16.3% | 0.078830 |       -- |      -- |  2227 |MPI
||---------------------------------------------------------
||  12.7% | 0.061266 | 0.078133 |   64.1% |    351 |mpi_waitall_
||   2.2% | 0.010607 | 0.011582 |   59.7% |    936 |mpi_isend_
||   1.4% | 0.006945 | 0.004463 |   44.7% |    936 |mpi_irecv_
|==========================================================
```

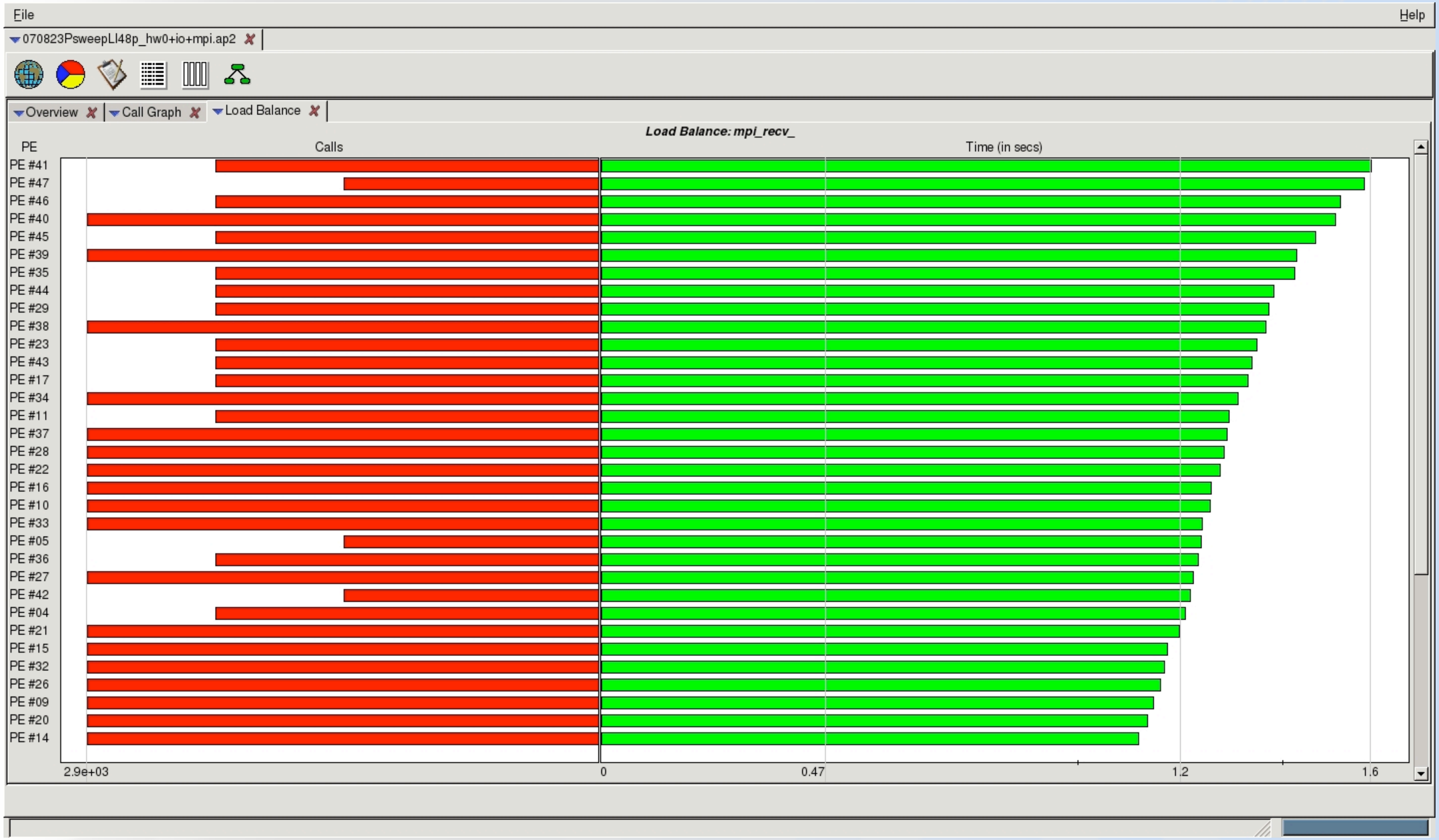
Cray Apprentice2 Load Imbalance Support

-
- Load imbalance can be viewed from:
  - Call Tree Visualization
  - Load Balance Distribution
    - By Time
    - By HW counters

Example: Swim Benchmark



Load Distribution



MPI Sync Time

-
- Determines 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 -O apa …
  - pat_build –g mpi …
- Rank arrival shown separately in report
  - MPI_Reduce(SYNC)
  - MPI_Reduce

OpenMP Performance Metrics

- Per-thread timings
- Overhead incurred at enter/exit of
  - parallel regions
  - worksharing constructs within parallel regions
- Load balance information across threads
- Sampling performance data without API
- Separate metrics for OpenMP runtime and OpenMP API calls

OpenMP Data from pat_report

-
- Default view (no options needed to pat_report)
  - focus on where program is spending its time
  - shows imbalance across all threads
  - assumes all requested resources should be used
  - Highlights non-uniform imbalance across threads
  - Top threads got most of the work
  - Bottom threads got least of the work

Profile Guided Rank Placement Suggestions

- When to use?
  - Point-to-point communication consumes significant fraction of program time and load imbalance detected
- Available if MPI functions are traced
  - pat_build –g mpi …
  - pat_build –O my_program.apa
- Sorted suggestions provided in resulting report
- Custom placement files automatically generated

Profile Guided Rank Placement Suggestions

-
- Rank order suggestions based on:
  - Sent message statistics
    - pat_report –O mpi_sm_rank_order
  - User time
    - pat_report –O mpi_rank_order
  - HW counters
    - pat_report –O mpi_rank_order /
      -s mro_metric=DATA_CACHE_MISSES

Example: -O mpi_sm_rank_order (sweep3d)

```
Notes for table 1:

    To maximize the locality of point to point communication, choose
    and specify a Rank Order with small Max and Avg Sent Msg Total Bytes
    per node for the target number of cores per node.

    To specify a Rank Order with a numerical value, set the environment
    variable MPICH_RANK_REORDER_METHOD to the given value.

    To specify a Rank Order with a letter value 'x', set the environment
    variable MPICH_RANK_REORDER_METHOD to 3, and copy or link the file
    MPICH_RANK_ORDER.x to MPICH_RANK_ORDER.
```

Summary

- Cray tools measure and display imbalance metrics for use in identifying performance bottlenecks
- Metrics available to determine load imbalance in application
  - Process and thread imbalance information
  - Communication versus computation
  - Inter-node versus intra-node activity
  - Degree of imbalance
  - Potential savings if imbalance corrected
- Text and visual formats for viewing code imbalance available

Detecting Application Load Imbalance on Cray Systems

Questions / Comments

Thank You!

Example: -O mpi_sm_rank_order (sweep3d)

```
Table 1:  Sent Message Stats and Suggested MPI Rank Order

                Communication Partner Counts

        Number   Rank
      Partners  Count      Ranks

             2      4      0   7  40  47
             3     20      1   2   3   4  ...
             4     24      9  10  11  12  ...
------------------------------------------------------------
                   Sent Msg Total Bytes per MPI rank

             Max           Avg           Min  Max   Min
      Total Bytes  Total Bytes  Total Bytes  Rank  Rank

         60825600     51840000     29721600  9     7
------------------------------------------------------------
       Dual core:  Sent Msg Total Bytes per node

 Rank         Max           Avg           Min  Max Node  Min Node
Order  Total Bytes  Total Bytes  Total Bytes  Ranks     Ranks

    1     87091200     69120000     42163200  10,11     6,7
    u     87091200     71884800     42163200  18,19     46,47
    d     87091200     72633600     42163200  17,18     46,47
    0    121651200    103680000     71884800  9,33      7,31
    2    121651200    103680000     60134400  26,21     40,7
```