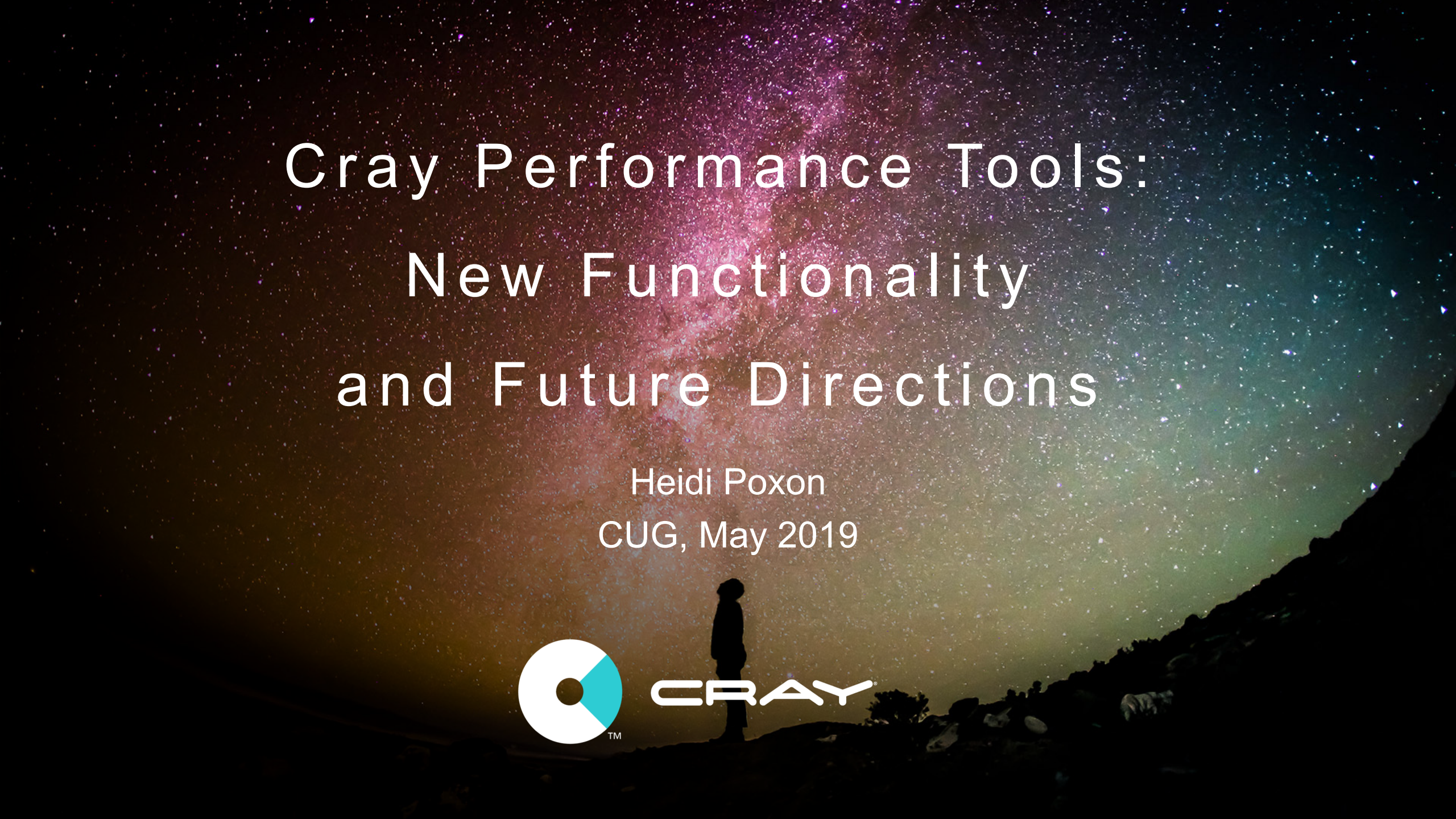
Cray Performance Tools: New Functionality and Future Directions

Heidi Poxon

CUG, May 2019

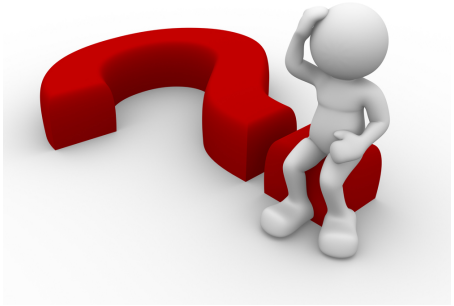


Content

- Highlights since last CUG
  - Ease-of-use
  - Profiling at scale
  - Data interpretation
  - Application sensitivities
- What's next

CRAY

Do Not Assume You Know Your Application Profile



- Which is dominant: computation or data movement and where?
- Is the program sensitive to memory bandwidth or memory latency?
- 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

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

Two Modes of Use

- Lite modes: simple interface for convenience

- 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

Whole Program Analysis

Subtitle here, if needed

When Waiting to Recompile Isn't Practical…

- Example application build scenario
  - 12,382 Fortran files, some source created during pre-processing steps
  - 2,079 C files, 965 of them created with pre-processing utility
  - 54 C++ files
  - 14,151 total source files to compile
  - Creates 26 libraries
  - Code takes 3 – 5 hours to compile depending on compiler used

- *How do I obtain some performance information for this code?*

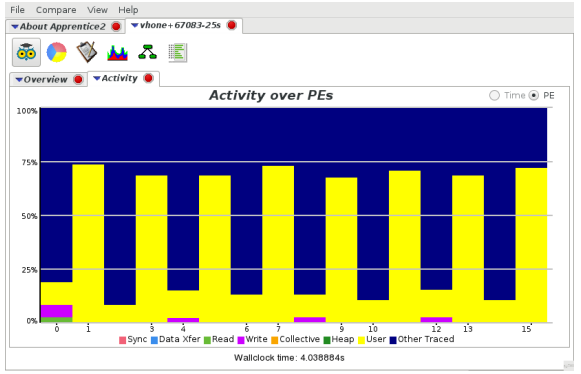
Try pat_run, the '*No-Re-Compile Necessary*' Tool !

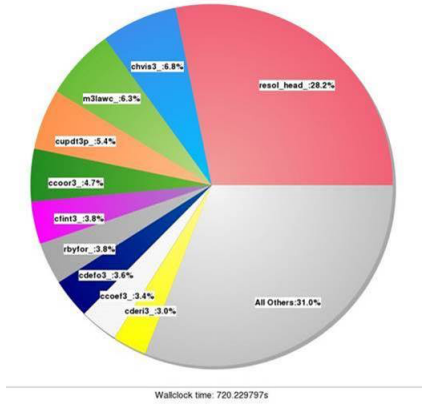
- To use: insert before executable in run command
  - `user@login> srun –n 16 pat_run ./my_program`
  - `user@login> pat_report expdir > my_report`

- pat_run now checks for dynamic vs static linked programs (-d option asserts dynamic and skips validation)

- pat_run can profile MPMD codes:
  - `user@login aprun -n … \`
    `pat_run a.out … pat_run b.out …`

- pat_run requires dynamic linking and Linux 4.X kernel
  - Not available on SLES 11 or Cray CS systems



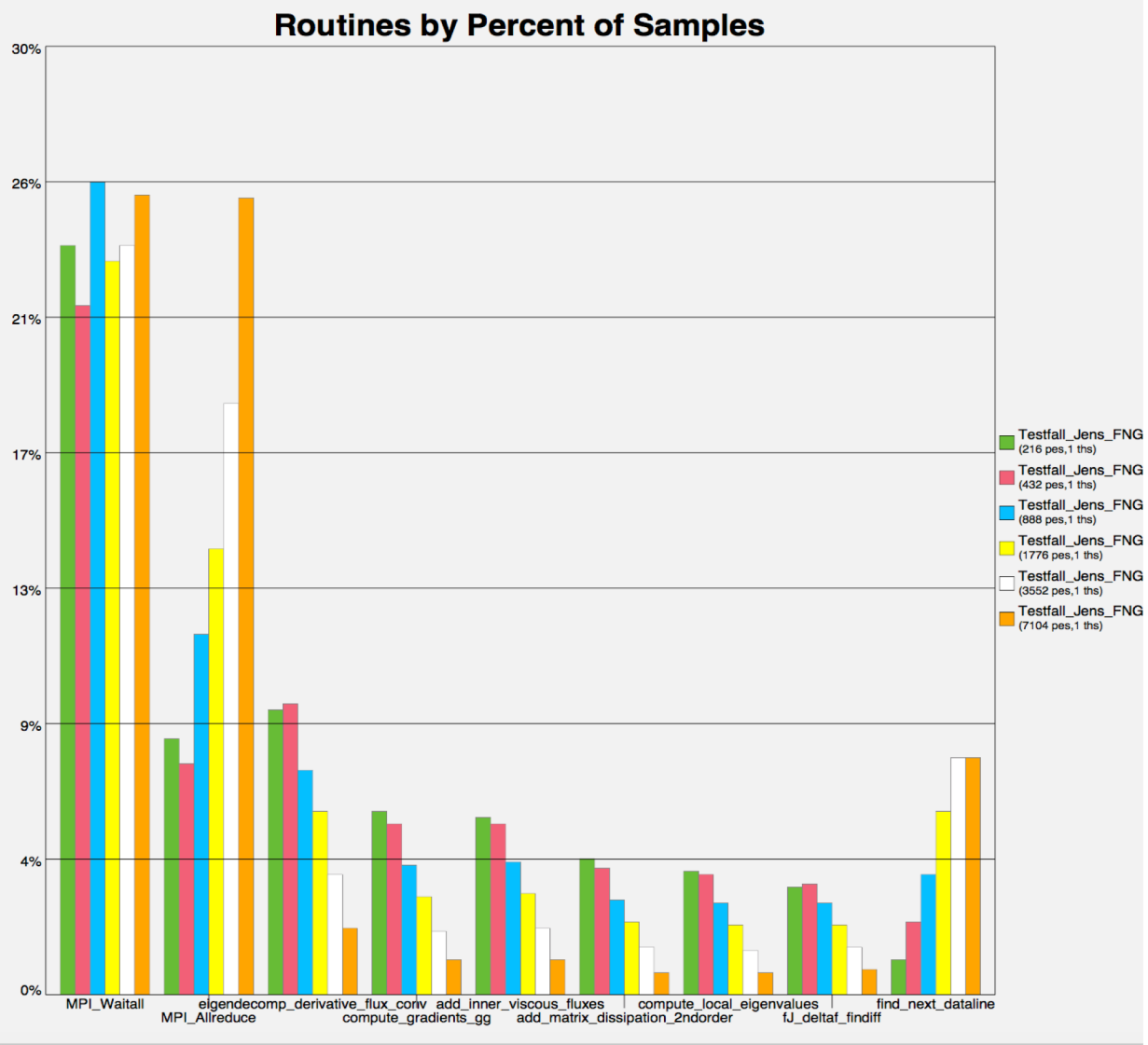


Can’t Wait for a Profile?

- Perftools now allows you to end an experiment early
  - Just force the program to stop (abort, kill job, etc.)

- Collected data is processed as if it were the full run
  - Useful for long running jobs
  - Stepping stone to eventually get profile previews while job is running

Check Program Scaling with pat_view



- pat_view takes multiple experiment directories as input
- Helpful when assessing performance differences between runs
- Good for scaling analysis

Is It Memory-bound?

and Other On-Node Analysis

New Node Sensitivity Guidance

- Running perftools-lite identifies key program bottlenecks

- Running perftools-lite with default HW counters collected with each sample provides finer granularity
  - `user@login> export PAT_RT_SAMPLING_DATA=perfctr@1`
  - Performance counter data presented in function profile

- `perfctr@ratio` will collect the selected performance counters in a specific frequency
  - Ratio = 1 : Data will be collected each time the counter is sampled
  - Ratio = 100 (default) : Data will be collected at every 100th sample

- It is helpful to run within a socket to avoid extra NUMA domains when analyzing application node sensitivities

Memory Bandwidth Sensitivity Guidance

Functions Slowed By Memory Bandwidth Utilization

The performance data for the functions shown below suggest that their performance is limited by memory bandwidth. To confirm this, try running with fewer processes placed on each node.

```
  Samp% |  Memory |   Stall | Function
        | Traffic | PerCent |  Numanode=HIDE
        |       / |         |   PE=HIDE
        | Nominal |         |
        |    Peak |         |
|-----------------------------------------
| 40.9% |   54.1% |   93.8% | daxpy_kernel_8
| 36.1% |   59.4% |   93.8% | dgemv_kernel_4x4
|=========================================
```

Example Traffic From an MPI+OpenMP Run

Table 3: Memory Bandwidth by Numanode (limited entries shown)

Memory Traffic GBytes	Local Memory Traffic GBytes	Remote Memory Traffic GBytes	Thread Time	Memory Traffic GBytes / Sec	Memory Traffic / Nominal Peak	Numanode Node Id=[max3,min3] PE=HIDE Thread=HIDE
184.47	173.59	10.89	11.578777	15.93	20.7%	numanode.0
183.50	173.59	9.91	11.569322	15.86	20.7%	nid.63
182.61	172.40	10.21	11.578777	15.77	20.5%	nid.61
178.55	167.75	10.80	11.563156	15.44	20.1%	nid.71
178.10	168.14	9.96	11.562097	15.40	20.1%	nid.62
178.08	168.07	10.01	11.564512	15.40	20.1%	nid.68
178.01	167.20	10.82	11.572032	15.38	20.0%	nid.70
60.36	14.73	45.62	9.073119	6.65	8.7%	numanode.1
60.36	14.73	45.62	9.072693	6.65	8.7%	nid.63
59.88	14.33	45.55	9.071553	6.60	8.6%	nid.62
59.48	14.19	45.29	9.068044	6.56	8.5%	nid.68
58.78	13.70	45.08	9.069259	6.48	8.4%	nid.70
58.67	13.87	44.81	9.071591	6.47	8.4%	nid.69
58.53	13.86	44.67	9.067146	6.46	8.4%	nid.71

Notice remote memory traffic by OpenMP threads

Low Vectorization Guidance

Functions with Low Vectorization

The performance data for the functions shown below suggest that their performance could be improved by increased vectorization. Use compiler optimization messages to identify loops in those functions that were not vectorized, and try to use directives or restructure the loops to enable them to vectorize.

```
 Samp% |    Vector |   Stall | Function
       |intensity |PerCent |   PE=HIDE
       |           |         |Thread=HIDE
|-------------------------------------------------------
|47.7% |       0.3%|   15.2% | depose_jxjyjz_esirkepov_1_1_1_
|=======================================================
```

Memory Latency Sensitivity Guidance

Functions Slowed By Memory Latency

The performance data for the functions shown below suggest that their performance is limited by memory latency. It could be beneficial to modify prefetching in loops in those functions, by modifying compiler-generated prefetches or inserting directives into the source code.

Samp%	Memory Traffic / Nominal Peak	Stall PerCent	Function Numanode=HIDE PE=HIDE
72.8%	34.8%	33.9%	dim3_sweep$dim3_sweep_module_

Improved pat_report Table Notes

Table 3: Profile by Function Group and Function

Includes data aggregation information

This table shows functions that have the most significant exclusive time, taking for each thread the average time across ranks.
The imbalance percentage is relative to the team observed to participate in execution.
Use `-s th=ALL` to see individual thread values.

For further explanation, see the "General table notes" below,
or use: `pat_report -v -O profile_th_pe ...`

Team Observed to Participate In Execution

Table 3: Profile by Function Group and Function

```
 Samp% |  Samp | Imb. |  Imb. | Team | Group
       |       | Samp | Samp% | Size |  Function=[MAX]
       |       |      |       |      |   Thread=HIDE
       |       |      |       |      |    PE=HIDE

 100.0% | 196.4 |   -- |    -- |   -- | Total
|-------------------------------------------------------------
|  60.3% | 118.4 |   -- |    -- |   -- | USER
||------------------------------------------------------------
||  56.8% | 111.5 | 55.7 | 99.9% |    2 | jacobi_mpiomp_
||   1.4% |   2.8 |  1.5 | 31.8% |   16 | compute_diff_.LOOP@li.256
||============================================================
```

Team size shows number of threads participating or the number of PEs, depending on the context

New Processor Support

- NVIDIA Volta on Cray CS-Storm
  - Available starting with perftools 7.0.2

- AMD Naples on Cray CS systems
  - Available starting with perftools 7.0.2
  - Includes CrayPat, Reveal, and Cray Apprentice2
  - Access to performance counters not yet available
    - Due to older RedHat/CentOS versions running on CS systems

- Intel CascadeLake on Cray XC
  - Available starting with perftools 7.0.6

What's Next?

- Profiling Python codes with CrayPat
- Roofline plot to detect application sensitivities
- Templates for profiling large, long running jobs

Summary of Cray Performance Tools

- Focus on whole program analysis
- Reduce the time investment associated with porting and tuning applications on new and existing Cray systems
- Provide easy-to-use interfaces and a wealth of capability when you need it for analyzing the most critical production codes
- Offer analysis and recommendations that focus on areas that impact performance and scaling, such as
  - Imbalance
  - Communication overhead and inefficiencies
  - Vectorization and memory utilization efficiency

SAFE HARBOR STATEMENT

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts.

These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Thank You

QUESTIONS?