Interpreting perftools Performance Data

Heidi Poxon
Sr. Principal Engineer
Cray Inc.

April 2019
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.
Some Useful Experiments

• Identify slowest areas and notable bottlenecks of a program
  • Use **perftools-lite**
  • Good for examining performance characteristics of a program and for scaling analysis

• Focus on MPI communication
  • Use **perftools-lite** first to determine if MPI time is dominant or if there is a load imbalance between ranks
  • Use **perftools (pat_build –g mpi)** to collect more detailed MPI-specific information including MPI_SYNC time to detect late arrivers to collectives
  • Good for identifying source of imbalance and scaling analysis at targeted final job size

• Focus on loop optimization
  • Use **perftools-lite-loops**
  • Use **perftools-lite-hbm** for memory traffic analysis
  • Good for vectorizing, parallelizing and cache optimization
Cray Compiler Listings

- CCE provides loopmark, cross-references, compile options, and optimization messages in easy-to-read text files

- Just add the following flag to the application’s Makefile:

  `-h list=a` ...

- Tip: For additional information on restructuring and optimization changes made by CCE, try `-h list=d` for decompiled code

- Use with perftools to understand top time consuming loop optimization information
Get Additional Information Without Re-running

• Generate full report
  • user@login> pat_report my_data_directory+12s/ > rpt

• Generate report with call tree (or by callers)
  • user@login> pat_report –O calltree+src

• Generate report without pruning
  • user@login> pat_report –P

• Show each MPI rank or each OpenMP thread in report
  • user@login> pat_report –s pe=ALL
  • user@login> pat_report –s th=ALL
Don’t See an Expected Function?

• To make the profile easier to interpret, samples are attributed to a caller that is either a user defined function, or a library function called directly by a user defined function.

• To disable this adjustment, and show functions actually sampled, use the `pat_report -P` option to disable pruning.

• You should be able to see the caller/callee relationship with `pat_report -P -O callers`.
Don’t See an Expected Function? (continued)

Why don’t I see a particular function in a report?

• Cray tools filter out data that may distract you
  • Use ‘pat_report -T’ to see functions that didn’t take much time

• Still don’t see it?
  • Check the compiler listing to see if the function was inlined
What is ETC Group in a Report?

• When a function is called that cannot be attributed to a user-defined parent function, it gets placed in ETC

• Try ‘pat_report -P’

• Note: pat_report depends on the accuracy of the DWARF issued by the compiler
How Do I See per-Rank or per-Thread Data?

• $ pat_report -s pe=ALL

• $ pat_report -s th=ALL
How Was Data Aggregated?

• Check the Notes before each table in the text report

Notes for table 1:

This table shows functions that have significant exclusive sample hits, averaged across ranks.

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

Table 1: Profile by Function
When to Collect Counters

• Use to understand the “why” of a bottleneck

• Default set of counters are collected for whole program
  • Used to present memory and vector summary metrics

• User can choose to collect per function or per region of code with PAT_region API
Performance Counters Overview

• Cray supports raw counters, derived metrics and thresholds for:
  • Processor (core and uncore)
  • Network
  • Accelerator
  • Power

• Predefined groups
  • Groups together counters for experiments

• See *hwpc, nwpc, accpc, and rapl* man pages
CrayPat 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 HW, network or power counter events
  • Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)
How to Get List of Events for a Processor

• Run the following utility on a compute node:
  • papi_nativeavail
  • papi_avail

• Use pat_help to see counter groups and derived metrics
  • user@login> pat_help counters processor_type deriv

• To collect performance counters
  • Set PAT_RT_PERFCTR environment variable to list of events or group prior to execution
Focus on Loop Optimization – Find Top Loops

- $ module load perftools-lite-loops

- Build program with CCE
  - Should see messages from CrayPat during build saying that it created an instrumented executable

- Remember to add `-hlist=a` to build with CCE listing

- Add `-hpl=/path_to_program_library/my_program.pl` if you want to use Reveal

- Run program

- Performance data sent to STDOUT and to directory with unique name
## Example Loop Statistics

<table>
<thead>
<tr>
<th>Inclusive and Exclusive Time in Loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop</td>
</tr>
<tr>
<td>Incl Time</td>
</tr>
<tr>
<td>Time%</td>
</tr>
<tr>
<td>99.4%</td>
</tr>
<tr>
<td>98.7%</td>
</tr>
<tr>
<td>26.5%</td>
</tr>
<tr>
<td>24.6%</td>
</tr>
<tr>
<td>24.2%</td>
</tr>
<tr>
<td>22.5%</td>
</tr>
<tr>
<td>22.5%</td>
</tr>
<tr>
<td>22.5%</td>
</tr>
<tr>
<td>18.9%</td>
</tr>
<tr>
<td>17.1%</td>
</tr>
<tr>
<td>15.7%</td>
</tr>
<tr>
<td>5.0%</td>
</tr>
</tbody>
</table>
How to Identify Important Loops

• High inclusive time

• Create call tree with loops:
  • `user@login> pat_report -O calltree`
What About Memory Bandwidth?

- Phased in over perftools 7.0.0, 7.0.1, and 7.0.2 for Intel Xeon processors

- New default counter group with perftools-lite and perftools experiments

- New table for memory bandwidth by NUMA node in default lite and full reports

- Separate functionality from perftools-lite-hbm experiment which uses CCE, CrayPat, and Reveal to track memory traffic and associate with allocation sites
Example: Memory Bandwidth per NUMA

8 MPI ranks, 4 on each of 2 nodes

Table 1: Memory Bandwidth by Numanode

<table>
<thead>
<tr>
<th>Memory GBytes</th>
<th>Traffic GBytes</th>
<th>Local Memory</th>
<th>Remote Memory</th>
<th>Thread Time</th>
<th>Memory Traffic GBytes / Sec</th>
<th>Nominal Peak</th>
<th>Numanode Node Id</th>
<th>PE=HIDE</th>
<th>Thread=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>39,429</td>
<td>39,429</td>
<td>0</td>
<td>990.218871</td>
<td>39.82</td>
<td>33.4%</td>
<td></td>
<td>Max of Numanode values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
<td>----------------------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>39,429</td>
<td>39,429</td>
<td>0</td>
<td>990.217439</td>
<td>39.82</td>
<td>33.4%</td>
<td></td>
<td>numanode.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39,429</td>
<td>39,429</td>
<td>0</td>
<td>990.217439</td>
<td>39.82</td>
<td>33.4%</td>
<td></td>
<td>nid.200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38,389</td>
<td>38,389</td>
<td>0</td>
<td>990.224163</td>
<td>38.77</td>
<td>32.5%</td>
<td></td>
<td>nid.205</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38,857</td>
<td>38,857</td>
<td>0</td>
<td>990.218903</td>
<td>39.24</td>
<td>32.9%</td>
<td></td>
<td>numanode.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38,857</td>
<td>38,857</td>
<td>0</td>
<td>990.211194</td>
<td>39.24</td>
<td>32.9%</td>
<td></td>
<td>nid.200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38,528</td>
<td>38,528</td>
<td>0</td>
<td>990.226690</td>
<td>38.91</td>
<td>32.6%</td>
<td></td>
<td>nid.205</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Examine NUMA Traffic

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

<table>
<thead>
<tr>
<th>Memory Traffic GBytes</th>
<th>Local GBytes</th>
<th>Remote GBytes</th>
<th>Thread Traffic Time</th>
<th>Memory Traffic GBytes</th>
<th>Memory Traffic / Sec</th>
<th>Node Id</th>
<th>PE=HIDE</th>
<th>Numanode</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.95</td>
<td>171.48</td>
<td>1.48</td>
<td>19.755654</td>
<td>8.75</td>
<td>11.4%</td>
<td>numanode.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.77</td>
<td>171.48</td>
<td>1.30</td>
<td>19.414237</td>
<td>8.90</td>
<td>11.6%</td>
<td>nid.68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.09</td>
<td>170.61</td>
<td>1.48</td>
<td>19.071340</td>
<td>9.02</td>
<td>11.7%</td>
<td>nid.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>171.20</td>
<td>169.93</td>
<td>1.27</td>
<td>17.631761</td>
<td>9.71</td>
<td>12.6%</td>
<td>nid.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162.51</td>
<td>161.07</td>
<td>1.43</td>
<td>19.675857</td>
<td>8.26</td>
<td>10.8%</td>
<td>nid.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162.28</td>
<td>160.82</td>
<td>1.46</td>
<td>19.730793</td>
<td>8.22</td>
<td>10.7%</td>
<td>nid.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161.75</td>
<td>160.29</td>
<td>1.46</td>
<td>19.755654</td>
<td>8.19</td>
<td>10.7%</td>
<td>nid.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>168.69</td>
<td>166.81</td>
<td>1.89</td>
<td>19.781479</td>
<td>8.53</td>
<td>11.1%</td>
<td>numanode.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>168.69</td>
<td>166.81</td>
<td>1.89</td>
<td>19.454144</td>
<td>8.67</td>
<td>11.3%</td>
<td>nid.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>167.74</td>
<td>166.03</td>
<td>1.71</td>
<td>19.476164</td>
<td>8.61</td>
<td>11.2%</td>
<td>nid.63</td>
<td></td>
<td></td>
</tr>
<tr>
<td>166.66</td>
<td>164.88</td>
<td>1.78</td>
<td>19.225409</td>
<td>8.67</td>
<td>11.3%</td>
<td>nid.61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161.68</td>
<td>160.07</td>
<td>1.61</td>
<td>19.781479</td>
<td>8.17</td>
<td>10.6%</td>
<td>nid.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161.60</td>
<td>159.99</td>
<td>1.62</td>
<td>19.642791</td>
<td>8.23</td>
<td>10.7%</td>
<td>nid.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>157.32</td>
<td>156.01</td>
<td>1.31</td>
<td>18.036118</td>
<td>8.72</td>
<td>11.4%</td>
<td>nid.72</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Memory Bandwidth by Numanode (limited entries shown)

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

**perftools-lite:**

- Optionally run `pat_report` on the data directory from login node
  - `export PAT_RT_REPORT_CMD=pat_report,-Q0`
  - Reduces job execution time, but disables parallel `pat_report` execution

**perftools-lite or perftools:**

- For a quick preview of performance data, use subset of data to generate a report
  - `user@login> pat_report -Q1` ➡️ report from 1st ap2 file
  - `user@login> pat_report -Q3` ➡️ report from 1st, middle, and last file
Controlling Instrumentation

• Record Subset of PEs during execution
  • It works again! (we found that it was broken last year)
  • Example: export PAT_RT_EXPFILE_PES=0,4,5,10

• Don’t instrument select binaries when using perftools-lite
  • Good for applications that generate test or intermediate binaries with CMake and GNU Autotools
  • Use CRAYPAT_LITE_WHITELIST for binaries you DO want instrumented
Utility that allows you to profile un-instrumented, dynamically linked binaries with CrayPat!

- Delivers Cray performance tools profiling information for codes that cannot easily be rebuilt
- Makes profiling possible for a wider set of HPC applications
- Available starting with perftools 7.0.1
- Initially targets Cray XC systems running CLE 6 or later
Using pat_run

• Insert before executable in run command
  • user@login> srun -n 16 pat_run ./my_program
  • user@login> pat_report expdir > my_report

• Use existing perftools capability
  • Optionally collect a different group of performance counters
    • user@login> export PAT_RT_PERFCTR=1
    • user@login> aprun -n 16 pat_run ./my_program

• Perform other experiments, for example trace MPI routines
  • user@login> pat_run -g mpi ./my_program

• Create additional views of the data with pat_report options, such as
  • user@login> pat_report -P -O callers+src
Example Advanced Capability

• Use pat_region API to mark regions in your code to profile

• Enable or disable specific function tracing based on name, size, …

• Choose additional predefined trace groups (blas, hdf5, …)

• Adjust sampling rate (collect counters per sample, sample 1000 times a second, …)

• Collect different performance counters (cache hierarchy, TLB misses, …)

• Customize data sorting, aggregation, or collection
Creating a Timeline

• Can produce huge amounts of performance data
  • Adjust job size for shorter and smaller runs

• Can be used with sampling and tracing (tracing is most common)
  • perftools-lite-events module
  • pat_build -g mpi
  • pat_build -u -g mpi

• To enable, set PAT_RT_SUMMARY=0 environment variable at runtime
View Program Timeline (36GB CP2K Full Trace)

CPU call stack:
- Bar represents CPU function or region: Hover over bar to get function name, start and end time.

Shows wait time

Hover to see what different filters do

Program histogram showing wait time

Program wallclock time line
Memory High Water Over Time (Apprentice2)

Produced with:
pat_build ./my_program
PAT_RT_SAMPLING_DATA=memory
PAT_RT_SUMMARY=0
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 complimented with 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