Performance Profiling on KNL with Cray perftools-lite

Heidi Poxon
Sr. Principal Engineer
Cray Programming Environment

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 publically 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.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and 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.

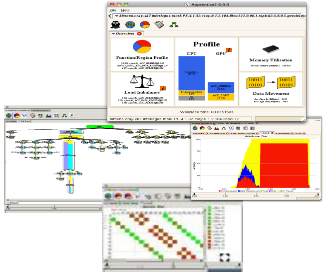
Other names and brands may be claimed as the property of others. Other product and service names mentioned herein are the trademarks of their respective owners.

Agenda

- Overview of Cray Performance Tools
- Identifying slowest areas of a program
- Tips for analyzing program performance

Overview of Cray Performance Tools

Load modules to access software



Choose experiment to target your goal

Visualize application bottlenecks

Goals

- 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 your profiling experience by using simple (lite mode) and/or advanced interfaces for a wealth of capability that targets analyzing large HPC jobs

Functional Highlights

- **Whole program performance analysis with**
  - Novice and advanced user interfaces
  - Support for MPI, SHMEM, OpenMP, PGAS, OpenACC, CUDA
  - Load imbalance detection
  - HW counter metrics (hit rates, computational intensity, etc.)
  - Observations on inefficiencies
  - Data correlation to user source
  - Minimal program perturbation

- **Sampling, tracing with runtime summarization (RTS), full trace (timeline) modes available**

- **Supports CCE, Intel and GCC compilers on Cray XC systems**
- **Supports CCE on Cray CS systems**

Components

- **CrayPat and CrayPat-lite**
  - Identifies top time consuming routines, work load imbalance, MPI rank placement strategies, etc.
- **PAPI**
  - Performance counters (used by CrayPat or directly by user)
- **Cray Apprentice2**
  - Visualize load imbalance, excessive communication, network contention, excessive serialization
- **Reveal**
  - View CCE optimization messages, key loops in program, high bandwidth memory traffic, add OpenMP to program

Cray Performance Tools Status

- **perftools-base/7.0.0 released in Dec 2017**
- **What's new?**
  - Perftools-lite performance improvements (execution, scaling)
  - Performance data experiment directory
  - Memory and vector sensitivity metrics

How to Access Cray Performance Tools

perftools-base module

- Provides access to performance tools instrumentation modules, documentation, Reveal and Apprentice2
- Doesn't impact program build
- If not loaded by default on a system, you can load in your .profile or .login and leave it loaded
- Once loaded, do a '`module avail perftools`' to see available instrumentation modules

Program Instrumentation Modules

- Instrumentation modules prepare an application for performance data collection

```
> module avail perftools

---------- /opt/cray/pe/perftools/7.0.0/modulefiles ----------
perftools
perftools-lite
perftools-lite-events
perftools-lite-gpu
perftools-lite-hbm
perftools-lite-loops
perftools-nwpc
```

Use first to get basic program performance profile

Interfaces Available

- **CrayPat-lite: simple interface for convenience**

- **CrayPat: advanced interface for in-depth performance investigation and tuning assistance as well as data collection control**

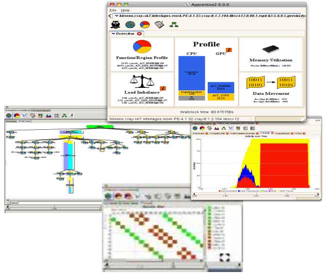
- **Both offer:**
  - Whole program analysis across many nodes
  - Indication of causes of problems
  - Ability to easily switch between the two interfaces

Simple vs Advanced Interface

- Has fewer steps
- Is easier to use if you rarely profile applications (don’t have to remember how to use the tools)
- Provides a condensed text report
- Performs performance data processing and report generation at end of job on compute nodes
- Allows you to mix with advanced interface
  - Run pat_report to get full report with simple interface

Identifying Slowest Areas of a Program

Load perftools-lite



Build and run program

Interpret results

Example: Using perftools-lite

- `$ module load perftools-lite`
- **Build program**
  - Should see message at end of build from CrayPat saying that it created an instrumented executable
  - Add `–hlist=a` to build with CCE listing for optimization feedback
- `$ aprun/srun –n … ./my_program`
- **Performance data sent to `STDOUT` and to directory with unique name**
  - Refer to CCE listing with sampling by line data in Table 2

Example: Cray loopmark Messages

cc/ftn/CC –hlist=a …

```
b-------<  do i3=2,n3-1
b b-----<     do i2=2,n2-1
b b Vr--<        do i1=1,n1
b b Vr            u1(i1) = u(i1,i2-1,i3) + u(i1,i2+1,i3)
b b Vr     *          + u(i1,i2,i3-1) + u(i1,i2,i3+1)
b b Vr            u2(i1) = u(i1,i2-1,i3-1) + u(i1,i2+1,i3-1)
b b Vr     *          + u(i1,i2-1,i3+1) + u(i1,i2+1,i3+1)
b b Vr-->        enddo
b b Vr--<        do i1=2,n1-1
b b Vr            r(i1,i2,i3) = v(i1,i2,i3)
b b Vr     *       - a(0) * u(i1,i2,i3)
b b Vr     *       - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
b b Vr     *       - a(3) * ( u2(i1-1) + u2(i1+1) )
b b Vr-->        enddo
b b----->     enddo
b------->  enddo
```

Outer loops were blocked (b)

Inner-loops were vectorized and unrolled (Vr)

Example: Cray loopmark Messages (cont)

```
ftn-6289 ftn: VECTOR File = resid.f, Line = 29
  A loop starting at line 29 was not vectorized because a recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 29
  A loop starting at line 29 was blocked with block size 4.
ftn-6289 ftn: VECTOR File = resid.f, Line = 30
  A loop starting at line 30 was not vectorized because a recurrence was found on "U1" between lines 32 and 38.
ftn-6049 ftn: SCALAR File = resid.f, Line = 30
  A loop starting at line 30 was blocked with block size 4.
ftn-6005 ftn: SCALAR File = resid.f, Line = 31
  A loop starting at line 31 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 31
  A loop starting at line 31 was vectorized.
ftn-6005 ftn: SCALAR File = resid.f, Line = 37
  A loop starting at line 37 was unrolled 4 times.
ftn-6204 ftn: VECTOR File = resid.f, Line = 37
  A loop starting at line 37 was vectorized.
```

Example of Explain Utility

users/ldr> explain ftn-6289

VECTOR: A loop starting at line %s was not vectorized because a recurrence was found on "var" between lines num and num.

Scalar code was generated for the loop because it contains a linear recurrence. The following loop would cause this message to be issued:

```
DO I = 2,100
   B(I) = A(I-1)
   A(I) = B(I)
ENDDO
```

Example: perftools-lite Job Summary

```
#################################################################
#                                                               #
#            CrayPat-lite Performance Statistics                #
#                                                               #
#################################################################
CrayPat/X:  Version 7.0.0.45 Revision 11f412d  11/08/17 09:36:36
Experiment:                  lite  lite/sample_profile
Number of PEs (MPI ranks):     96
Numbers of PEs per Node:       16  PEs on each of  6  Nodes
Numbers of Threads per PE:      1
Number of Cores per Socket:    68
Execution start time:  Tue Nov 14 11:44:06 2017
System name and speed:  nid00037  1401 MHz (approx)
Intel Knights Landing CPU  Family:  6  Model: 87  Stepping:  1
MCDRAM: 7.2 GHz, 16 GiB available as quad, cache (100% cache)

Avg Process Time:   612.10 secs
High Memory:      16,053.7 MBytes     167.2 MBytes per PE
I/O Read Rate:    1.764988 MBytes/sec
I/O Write Rate:   4.349897 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
```

COMPUTE | STORE | ANALYZE

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
```

Get Additional Information Without Re-running

- Run pat_report after collecting data with lite mode
  - `pat_report my_programXXXs/ > full_rpt`

  - `pat_report –O callers` or `pat_report –O callers+src`
  - `pat_report –O calltree` or `pat_report –O calltree+src`
  - **Check out load balance table**

- Learn about related tables in “Table Notes”
  - We try to suggest reports that dive deeper on a related topic
  - Provide data aggregation method

Example: Load Balance by Max Time

```
Table 2:  Profile of maximum function times (limited entries shown)
  Samp% |      Samp |     Imb. |  Imb. | Function
        |           |     Samp | Samp% |  PE=[max,min]
|------------------------------------------------------------------------
| 100.0% | 51,891.0 | 2,055.7 |   4.0% | LAMMPS_NS::PairLJCut::compute
||-----------------------------------------------------------------------
|| 100.0% | 51,891.0 |      -- |     -- | pe.32
||  93.0% | 48,263.0 |      -- |     -- | pe.93
||=======================================================================
|  11.3% |  5,871.0 |   193.1 |   3.3% | LAMMPS_NS::Neighbor::half_bin_newton
||-----------------------------------------------------------------------
||  11.3% |  5,871.0 |      -- |     -- | pe.66
||  10.7% |  5,535.0 |      -- |     -- | pe.94
||=======================================================================
|   8.6% |  4,480.0 | 2,418.6 |  54.6% | MPI_Send
||-----------------------------------------------------------------------
||   8.6% |  4,480.0 |      -- |     -- | pe.45
||   0.9% |    443.0 |      -- |     -- | pe.32
```

Recognizing OpenMP in a Report

- **CrayPat can collect the most detail from OpenMP using the Cray compiler**

- **OpenMP regions and loops are identified in report with the following syntax:**
  - function.REGION@li.49
  - function.LOOP@li.53

- **OpenMP statistics are collected by default (no need to enable anything in the tools)**
  - Most information is available with Cray compiler

How Do I See per-Rank or per-Thread Data?

- `$ pat_report –s pe=ALL`
- `$ 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? (cont’d)

- **Why don’t I see a particular function in the 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 the 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

Documentation Available

- Release Notes
  - `> module help perftools-base/`*`version_number`*

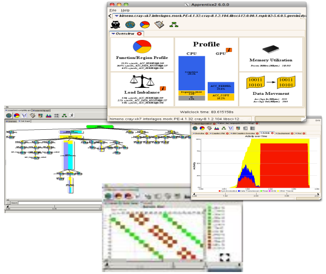
- User manual "Using the Cray Performance Measurement and Analysis Tools" available at http://pubs.cray.com

- pat_help – interactive help utility on the Cray Performance toolset

- Man pages

Tips for Analyzing Program Performance

Load perftools-lite



Build and run program

Interpret results

Where Do I Start?

- Determine problem size / job size that you ultimately want to run
- Get high level program profile at scale to locate key bottlenecks
- Work from high-level (inter-node) to low-level (intra-node) bottlenecks

Helpful 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 loop optimization, including adding OpenMP with Reveal**
  - Use `perftools-lite-loops`
  - Use `perftools-lite-hbm` for memory bandwidth sensitivity study
- **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
  - Good for scaling analysis at targeted final job size

Focus on Loop Optimization – Find Top Loops

- **`$ module load PrgEnv-cray perftools-lite-loops`**
  - Needs Cray compiler
- **Build program (build from scratch – we add compiler flags)**
  - Should see message at end of build from CrayPat 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
- **`$ aprun –n … ./my_program`**
- **Performance data sent to `STDOUT` and to experiment data directory with unique name**

Focus on MPI Communication

- `$ module load perftools`
- **Build program**
  - Remember to add `–hlist=a` to build with CCE listing
  - Can relink or use “a.out+orig” if created with perftools-lite
- **Instrument program**
  - `$ pat_build –g mpi ./my_program`
- **Run application**
  - `$ aprun/srun -n …  my_program+pat`
- **Create report**
  - `$ pat_report my_programXXX/ > my_report`

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

COMPUTE | STORE | ANALYZE

Questions?

Thank You!