Cray XC Series Application Programming and Optimization
Student Guide

TR-CPO NERSC

February 12 and 13, 2019 – Day 2

This document is intended for instructional purposes.
Do not use it in place of Cray reference documents

Direct comments about this publication to:

Mail: Cray Inc.
Cray Training
P.O. Box 6000
Chippewa Falls, WI 54729-0080
USA

E-mail: ttd_online@cray.com
Fax: +1 715 726 4991

Performance Tools

CrayPat, Apprentice2, and Reveal

Cray delivers an integrated set of performance tools that provide automatic program instrumentation, without requiring source code or file modifications. Before you can use these tools, ensure that your code compiles cleanly, runs to completion, and produces expected results.

Performance Tools

- **Cray `perftools-base`**
  - Provides access to man pages, utilities such as Reveal, Cray Apprentice2 and grid_order, and instrumentation modules
  - It does not add compiler flags to enable performance data collection
- **Cray `perftools-lite`**
  - An easy-to-use version of the CrayPat Performance Measurement and Analysis Tool
- **Cray `perftools`**
  - CrayPat is a performance analysis tool that collects performance information from a user application
  - Cray Apprentice2 displays graphical reports from the .ap2 file
  - Cray Reveal supports source code navigation using whole-program analysis data provided by the Cray Compiling Environment

2/11/2019 Cray, Inc. Private 2

Trace-based or synchronous experiments count every entry into and out of each function that is called in the application. Build (pat_build) options can reduce the number of functions to include in the experiment.

Further experimentation on a fine-grained portion of the application can occur through source code modifications, where a user uses CrayPat pat_region API in the source code. Normally this is not required.

CrayPat

- **Consists of three major components**

pat_build	Used to instrument the program to be analyzed
pat_report	A report generator
pat_help	An online help system, faq is on the front page
Additional man pages are hwpc, papi_counters, and intro_craypat	

- CrayPat (pat_build) supports two types of experiments: sampling and tracing
  - Sampling experiments capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - Tracing counts an event, such as the number of times an MPI call is executed
  - CrayPat uses PAPI to read the performance counters of the processor

pat_build Sampling

- **If tracing options are not included on the `pat_build` command line, `pat_build` defaults to sampling**
  - Sampling is controlled by the environment variable PAT_RT_EXPERIMENT
    - Supported sampling functions are: samp_pc_time, samp_pc_ovfl, samp_cs_time, or samp_cs_ovfl
    - Caution: Do not collect hardware counter information when you sample by overflow (for example< samp_pc_ovfl)
  - Use sampling to obtain a profile and then trace functions of interest

Using CrayPat

- **To instrument a program:**
  - Load the perftools module
    - `% module load perftools`
  - The executable and object (.o) files are required

```
rns/samp264% ftn -o samp264 samp264.f
WARNING: PerfTools is saving object files from a temporary
directory into directory '/home/users/rns/.craypat/samp264/12204'
swan rns/samp264%
```

2/11/2019 Cray, Inc. Private 5

In the example above, %pat_build program1 examines the program program1 and relinks its object and library files with files from the CrayPat run-time library to produce program1+pat. This operation requires the continued availability of the object files that were used to link program1 (either in their locations at the time program1 was linked or in a directory specified by the PAT_BUILD_LINK_DIR environment variable).

Using pat_build

- Run pat_build to instrument the program

```
rns/samp264% pat_build samp264
rns/samp264% ls -l samp264*
-rwxr-xr-x 1 rns hwpt 12067872 Feb 12 17:41 samp264
-rwxr-xr-x 1 rns hwpt 19306104 Feb 12 17:45 samp264+pat
```

- Execute the instrumented program
  - If your using a workload manager submit the job from the job-script

```
rns/samp264% cat samp264.slm
#! /bin/bash
#SBATCH -n 16
# srun ./samp264
# Job profiling phases
srun ./samp264+pat
# srun ./samp264+apa

rns/samp264% sbatch samp264.slm
Submitted batch job 141769
rns/samp264%

rns/samp264% squeue
 JOBID USER ACCOUNT        NAME  ST REASON START_TIME          TIME TIME_LEFT NODES CPUS
141769  rns  (null) samp264.slm  R None    2019-02-12T21:48:43 1:28     58:32     1   16
rns/samp264%
```

Experiment Output

- **The instrumented program generates a subdirectory**
  - For example the run on the previous page created a directory named `samp264+pat+26031-24s`
    - The directory name contains the following information:
      - name of the instrumented program: `samp264+pat`
      - the process ID: `26031`
      - the physical node—the application started on: `24`
      - and the type of experiment performed: `s` for sample and `t` for trace
    - In the subdirectory will be a subdirectory named `xf-files`
      - In there will be a `.xf` file for each of the nodes
      - The `.xf` files are the experiment output files

By default, for jobs with 255 PEs or less, a single .xf file is created. If the job uses 256 PEs or more, the square root number of PEs .xf files are created.

The user had to instrument their program with pat_build –O apa in order for pat_report to generate the .apa file.

Using `pat_report`

- **Use the `pat_report` command to read the *experiment* file**
  - `pat_report` will generate an `ap2-files` sub-directory, `build-options`.apa file, an `index.ap2` file, and a report to `stdout`
    - The .ap2 is used to generate additional text reports or is used by Apprentice2
      - The .ap2 files are portable; it does not require the *source* or `.xf` files
        - Prior to generating the ap2 files `pat_report` requires the .o, *source*, and `.xf` files be maintained.
        - The ap2 file is portable and can be archived for later use
    - The `build-options`.apa (Automatic Profiling Analysis) file is used (optionally) to assist you in creating a trace based experiment file

```
rns/samp264% ll samp264+pat+26031-24s
total 72
drwxr-xr-x 2 rns hwpt  4096 Feb 12 22:17 ap2-files
-rw-r--r-- 1 rns hwpt  1832 Feb 12 22:17 build-options.apa
-rw-r--r-- 1 rns hwpt 59392 Feb 12 22:17 index.ap2
drwxr-x--- 2 rns hwpt  4096 Feb 12 22:02 xf-files
rns/samp264%
```

2/11/2019 Cray, Inc. Private 8

Automatic Profiling Analysis (APA)

- Use the `build-options.apa` file generated by `pat_report` to build a trace experiment file
  - No need to specify the *executable*
  - You should get an instrumented program *samp264+apa*

```
rns/samp264% pat_build -O samp264+pat+26031-24s/build-options.apa
rns/samp264%
rns/samp264% ls -ltr samp264+*
-rwxr-xr-x 1 rns hwpt 22433392 Feb 12 22:00 samp264+pat
-rwxr-xr-x 1 rns hwpt 22406936 Feb 12 22:35 samp264+apa
```

- Run application to get top time-consuming routines

```
rns/samp264% cat samp264.slm
#! /bin/bash
#SBATCH -n 16
# srun ./samp264
# Job profiling phases
# srun ./samp264+pat
srun ./samp264+apa
```

  - Use `pat_report` to view the `.xf` file
  - The `build-options.apa` file can be modified and used again by you

2/11/2019 Cray, Inc. Private 9

The top time-consuming routines comes from the initial pat_build –O apa, which performs a form of sampling to get an initial profile. Then further information can be obtained for those top time consuming routines (identified in the .apa file) with the program instrumented using the .apa, and rerun.

Use pat_report to process the .xf file, not view the .xf file. View the text report generated to stdout or through Apprentice2.

Automatic Profiling Analysis (APA)

- Use `pat_report` to view the `.xf` file
  - The `build-options.apa` file from the *sample* based experiment can be modified and used again by you

```
rns/samp264% ls -l samp264+apa+27643-24t
total 4
drwxr-x--- 2 rns hwpt 4096 Feb 12 22:49 xf-files
rns/samp264% pat_report samp264+apa+27643-24t

 <<<< pat_report output to the screen >>>

rns/samp264% ls -l samp264+apa+27643-24t
total 88
drwxr-xr-x 2 rns hwpt  4096 Feb 12 22:55 ap2-files
-rw-r--r-- 1 rns hwpt 80896 Feb 12 22:55 index.ap2
drwxr-x--- 2 rns hwpt  4096 Feb 12 22:49 xf-files
rns/samp264%
```


The top time-consuming routines comes from the initial pat_build –O apa, which performs a form of sampling to get an initial profile. Then further information can be obtained for those top time consuming routines (identified in the .apa file) with the program instrumented using the .apa, and rerun.

Use pat_report to process the .xf file, not view the .xf file. View the text report generated to stdout or through Apprentice2.

`pat_build` Trace Options

- **To trace functions and create the instrumented executable, use the following `pat_build` options:**
  - `–g` traces non-user library functions for one of the predefined groups, like `[caf|cuda|gni|…|upc]`
    - Refer to the `pat_build` man page for a complete list
  - `-t` *tracefile* to specify a file containing a lists of functions to trace
  - `-T` *tracefunc* where *tracefunc* is a comma-separated list of function names to trace; *!tracefunc* excludes function
  - `-u` trace user functions
  - `-w` is used to trace MAIN. There are only trace points to collect performance data inserted at the beginning and end of MAIN.
    - This is helpful if the user wants to collect some data that has high collection overhead and wants to minimize additional tracing overhead.
  - `-o` allows you to specify the name of resulting instrumented program or the name can be the final argument. If neither are specified, the program name is appended with `+pat`
  - `-f` is used overwrite existing output file `instr_program`
  - Note: `pat_build` does not enable you to instrument a program that is also using the PAPI interface directly (via `libhwpc`)

Environment Variables

PAT_RT_SUMMARY	0 Turn off summary 1 Enable summary (default)
PAT_RT_PERFCTR	Specify the performance counter group to be collected
PAT_RT_EXPFILE_PER_PROCESS	0 Write experiment data to a single file Requires a file system capable of locking 1 Write a separate file for each process • An application may abort if the number of processes exceeds the number of open files permitted
PAT_RT_EXPFILE_NAME	The experiment file name
PAT_RT_EXPFILE_DIR	The directory that contains the experiment output file • Specify a Lustre directory when you create a single experiment output file

There are a number of environmental variables that define/modify the way CrayPat operates. See the intro_craypat man page for more information.

A Sequence of Commands

```
rns/samp264% module load perftools     # Loaded the CrayPat module
rns/samp264% ftn -o samp264 samp264.f  # compiled the code – simple application
rns/samp264% pat_build samp264         # Created the experiment executable
rns/samp264% vi samp264.slm            # modify the job script to run samp64+pat
rns/samp264% sbatch samp264.slm        # run the job
rns/samp264% cat samp264.slm.o141770   # Made sure the job ran ☺
rns/samp264% pat_report samp264+pat+26031-24s> samp264+pat+26031-24s.report
rns/samp264% view samp264+pat+26031-24s.report
rns/samp264% pat_build -O samp264+pat+26031-24s/build-options.apa
rns/samp264% ls -ltr
total 59184
-rwxr-xr-x 1 rns hwpt     5488 Oct 26  2014 samp264.f
-rwxr-xr-x 1 rns hwpt 15696888 Feb 12 22:00 samp264
-rwxr-xr-x 1 rns hwpt 22433392 Feb 12 22:00 samp264+pat
-rw-r--r-- 1 rns hwpt      127 Feb 12 22:06 samp264.slm.o141770
-rw-r--r-- 1 rns hwpt      147 Feb 12 22:06 samp264.slm.e141770
drwxr-x--- 4 rns hwpt     4096 Feb 12 22:17 samp264+pat+26031-24s
-rw-r--r-- 1 rns hwpt      214 Feb 12 22:39 samp264.slm

rns/samp264% vi samp264.slm            # modify the job script to run samp64+apa
rns/samp264% sbatch samp264.slm        # run the job
rns/samp264% pat_report samp264+apa+27643-24t > samp264+apa+27643-24t.report
rns/samp264% view samp264+apa+27643-24t.report
```

View samp264+pat+26031-24s Output

```
Table 1:  Profile by Function

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

 100.0% | 25,917.9 |   -- |    -- | Total
|-----------------------------------------
|  99.8% | 25,854.3 | 27.7 |  0.1% | USER
||----------------------------------------
||  99.8% | 25,854.3 | 27.7 |  0.1% | ghost_
|=========================================  Profile by
```

This is the report from the first “sample” experiment.
Table 1 shows the highest used functions, ghost_

Table 2 show more detail about the function ghost_ and in this example the high and low process

```
Table 2:  Profile of maximum function times

  Samp% |      Samp | Imb. | Imb.  | Function
        |           | Samp | Samp% |  PE=[max,min]
|---------------------------------------------------
| 100.0% | 25,882.0 | 27.7 |  0.1% | ghost_
||--------------------------------------------------
||  100.0% | 25,882.0 |   -- |    -- | pe.0
||   99.7% | 25,799.0 |   -- |    -- | pe.12
|===================================================
```

2/11/2019 Cray, Inc. Private 14

The table is a portion of the output of program1.rpt1.
The fifth column, labelled ***Calls,*** contains the count for all 4 PEs.
The second column, *Time*, lists the maximum time used by any PE per function.
The third column, ***Imb.*** (Imbalance) ***Time,*** lists the average time required by all PEs per function.
The fourth column, “***Imb. Time %,***, a value of 100% indicates that a single PE executed the function. A value of 0% would indicate that all PEs spent equal time performing the function. (Refer to the man page for information about the math used to calculate the percentage.)

View samp264+pat+26031-24s

```
Table 1:  Profile by Function
  Samp% |      Samp | Imb.  |  Imb.  | Group
        |           | Samp  | Samp%  |  Function
        |           |       |        |   PE=HIDE
 100.0% | 25,917.9 |    -- |     -- | Total
|--------------------------------------------
|  99.8% | 25,854.3 | 27.7 |   0.1% | USER
||-------------------------------------------
||  99.8% | 25,854.3 | 27.7 |   0.1% | ghost_
|============================================

Table 2:  Profile of maximum function times
  Samp% |      Samp | Imb.  |  Imb.  | Function
        |           | Samp  | Samp%  |  PE=[max,min]
|------------------------------------------------
| 100.0% | 25,882.0 | 27.7 |   0.1% | ghost_
||-----------------------------------------------
|| 100.0% | 25,882.0 |    -- |     -- | pe.0
||  99.7% | 25,799.0 |    -- |     -- | pe.12
|================================================

Table 4:  Program HW Performance Counter Data
==============================================================================
  Total
------------------------------------------------------------------------------
  Thread Time                                          259.641887 secs
  CPU_CLK_UNHALTED:THREAD_P                       852,323,364,085
  DTLB_LOAD_MISSES:WALK_DURATION                  223,631,012,363
  INST_RETIRED:ANY_P                               91,684,175,564
  RESOURCE_STALLS:ANY                             792,419,692,546
  UNHALTED_REFERENCE_CYCLES                       774,839,428,386
  OFFCORE_RESPONSE_0:ANY_REQUEST:LLC_MISS_LOCAL     6,912,550,656
  CPU CLK Boost                                              1.10 X
```

This is the report generated after `pat_build -O \ samp264+pat+26031-24s/build-options.apa` was executed and the executable `samp264+apa` was run. The APA file suggested `PERFCT value 1` be used. This is where the performance counter data comes from in Table 4

Hardware Performance Counters

- The APA file suggests which hardware performance counters you should use
  - To use different performance counters, set the PAT_RT_PERFCTR ENVIRONMENTAL variable and rerun the job.

```
samp264/samp264+pat+26031-24s% cat build-options.apa
 [clipped]
#   Collect the default PERFCTR group.
  -Drtenv=PAT_RT_PERFCTR=default
```

```
rns/samp264% cat samp264.slm.org
#! /bin/bash
#SBATCH -n 16
# CrayPat runtime options
export PAT_RT_PERFCTR=2
# export PAT_RT_SUMMARY=0
# Job execution
# srun ./samp264
# Job profiling phases
# srun ./samp264+pat
srun ./samp264+apa
```


An event set is a group of PAPI preset or native events

CrayPat defines 20 groups (sets)

Select a set by using the environment variable PAT_RT_HWPC

Profiling - counting specified events

Used in CrayPat

Overflow - testing events and alerting the application when a count is exceeded

Requires modification of the user application

Looking Closer

- **Load the `perftools` module**
  - Use the CrayPat `pat_region` API to identify the region of interest
    - In Fortran

```
include 'pat_apif.h‘

call PAT_region_begin(1, "Std_Deviation", istat)
…
call PAT_region_end(1,stat);
```

  - Compile your code
    - Use `pat_build` to create an instrumented binary
    - Use the environment variable PAT_RT_PERFCTR to select the hardware counters that you want to collect.   PAT_RT_PERFCTR=0
    - You can also save your favorite counters in a file and pass them to CrayPat
      - Add file name to PAT_RT_PERFCTR_FILE environment variable

In C/C++

```
#include <pat_api.h>
PAT_region_begin(1,“halo_loop");
...
PAT_region_end(1);
```

Looking Closer

```
call PAT_region_begin(1, "Std_Deviation", istat)
!      now find the standard deviation
       do k = 1 , nz
           do j = 1 , ny
               do i = 1 , nx
                   var = var +                                         &
     &                 ((array(i,j,k) - mean)*(array(i,j,k) - mean))
               enddo
           enddo
       enddo
call PAT_region_end(1, istat)
```

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

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

 100.0% | 260.834728 |       -- |    -- | 120,107.1 | Total
|---------------------------------------------------------------------
|  99.4% | 259.170891 |       -- |    -- |     101.0 | USER
||--------------------------------------------------------------------
||  95.0% | 247.879201 | 1.520514 |  0.7% |       1.0 | ghost_
||   4.3% |  11.291691 | 0.040904 |  0.4% |     100.0 | #1.Std_Deviation
|=====================================================================
```

Looking closer

From loop in code

```
Table 3:  Profile by Function Group and Function
… clip …
=============================================================================
  USER / #1.Std_Deviation
-----------------------------------------------------------------------------
  Time%                                                          4.3%
  Time                                                      11.291691 secs
  Imb. Time                                                  0.040904 secs
  Imb. Time%                                                     0.4%
  Calls                                         8.856 /sec      100.0 calls
  CPU_CLK_UNHALTED:THREAD_P                            37,117,179,867
  DTLB_LOAD_MISSES:WALK_DURATION                          180,919,720
  INST_RETIRED:ANY_P                                    7,200,138,317
  RESOURCE_STALLS:ANY                                  32,409,753,945
  UNHALTED_REFERENCE_CYCLES                            33,742,890,690
  OFFCORE_RESPONSE_0:ANY_REQUEST:LLC_MISS_LOCAL           601,357,957
  CPU CLK Boost                                                  1.10 X
  Resource stall cycles / Cycles                                87.3%
  Memory traffic GBytes                         3.408G/sec      38.49 GB
  Local Memory traffic GBytes                   3.408G/sec      38.49 GB
  Memory Traffic / Nominal Peak                                  5.7%
  Retired Inst per Clock                                         0.19
  Average Time per Call                                      0.112917 secs
  CrayPat Overhead : Time                        0.0%
=============================================================================
```

PAPI

- PAPI provides a common interface for the performance counters in various processors, including the Opteron
  - PAPI defines a set of Preset counters that map to a common performance counter in various processors
    - The Preset name matches as closely as possible to the Native event
      - Using the Preset name provides portability between processors when user code is modified to collect performance data
  - A Native event is an actual hardware counter in the processor
    - See the `papi_counters`, `papi_avail`, and `papi_native_avail` man pages
    - `papi_avail`, and `papi_native_avail` are commands that can be executed on the compute node to determine the available counters
      - `srun -n 1 /opt/cray/pe/papi/default/bin`

Rank Order and CrayPAT

- **One can also use the CrayPat performance measurement tools to generate a suggested custom ordering.**
  - Available if MPI functions traced (-g mpi or –O apa)
    - pat_build –O apa my_program
    - see Examples section of pat_build man page
  - pat_report options:
    - mpi_sm_rank_order
      - Uses message data from tracing MPI to generate suggested MPI rank order. Requires the program to be instrumented using the pat_build -g mpi option.
    - mpi_rank_order
      - Uses time in user functions, or alternatively, any other metric specified by using the -s mro_metric options, to generate suggested MPI rank order.

2/11/2019 Cray, Inc. Private 21

Rank Order and CrayPAT

- **`module load perftools`**
- **Rebuild your code**
  - `pat_build –O apa a.out`
  - Run `a.out+pat`
  - `pat_report –Ompi_sm_rank_order a.out+pat+…sdt/ > pat.report`
    - Creates `MPICH_RANK_REORDER_METHOD`.x file
  - Then set environment variable `MPICH_RANK_REORDER_METHOD=3` and link the file `MPICH_RANK_REORDER_METHOD.x` to `MPICH_RANK_ORDER`
  - Rerun your code

Rank Order and CrayPAT Example

```
Table 1: Suggested MPI Rank Order

Eight cores per node: USER Samp per node
 Rank          Max    Max/         Avg     Avg/   Max Node
Order    USER Samp     SMP   USER Samp      SMP   Ranks
    d        17062   97.6%       16907   100.0%   832,328,820,797,…
    2        17213   98.4%       16907   100.0%   53,202,309,458,…
    0        17282   98.8%       16907   100.0%   53,181,309,437,…
    1        17489  100.0%       16907   100.0%   0,1,2,3,4,5,6,7
```

- This suggests that:
  - The custom ordering “d” might be the best
  - Folded-rank next best
  - Round-robin 3rd best
  - Default ordering last
- **The utility `grid_order` can be used to statically generate MPI rank order**

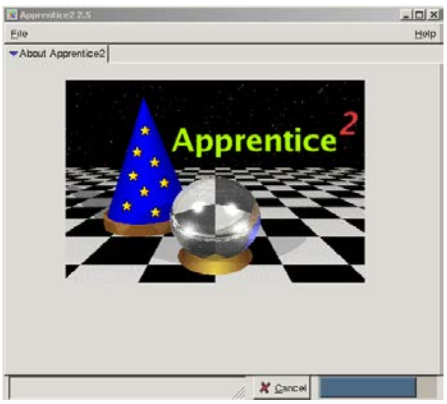
HSN Network Counters

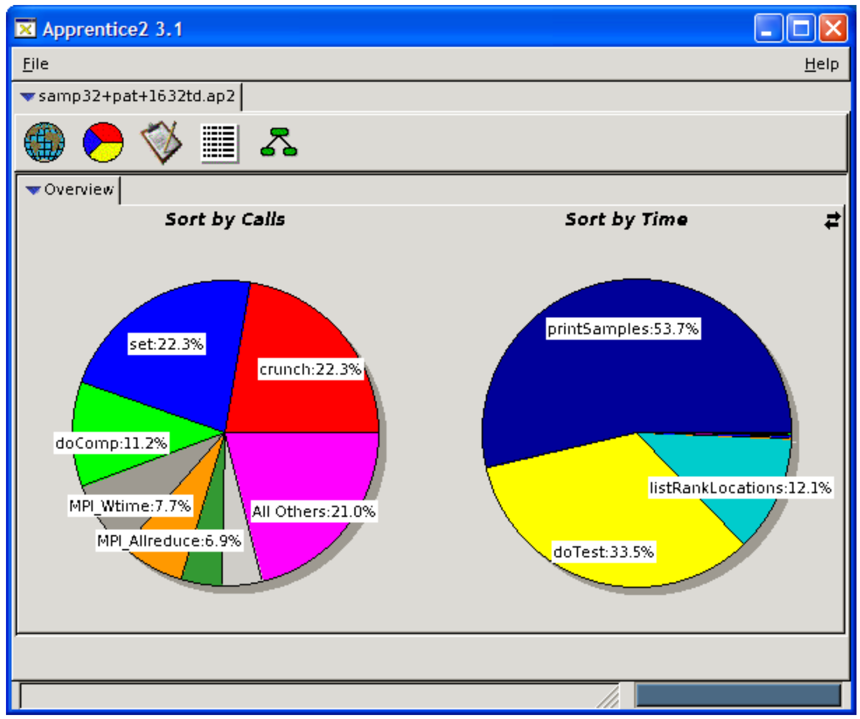
- **HSN Network counters are accessed through CrayPat and environment variables**
  - See the `intro_craypat` and `nwpc` man pages

PAT_RT_NWPC	Specifies individual Gemini performance counter event names.
PAT_RT_NWPC_CONTROL	Specifies parameters that control various aspects of the Gemini networking performance counters.
PAT_RT_NWPC_FILE	Specifies a file or list of files containing individual Gemini performance counter event names.
PAT_RT_NWPC_FILE_GROUP	Specifies a file or list of files containing specifications of Gemini performance counter groups.
PAT_RT_NWPC_FILE_TILE	Specifies a file or list unset of files containing specifications of Gemini performance counters that use the filtering counters to define new events.
PAT_RT_NWPC_TILE_DISPLAY	If set to nonzero value, writes the filtered tile NWPC event specifications to stdout.

Cray Apprentice2

```
% module load perftools
% app2 program1+pat+180tdo-0000.ap2
```

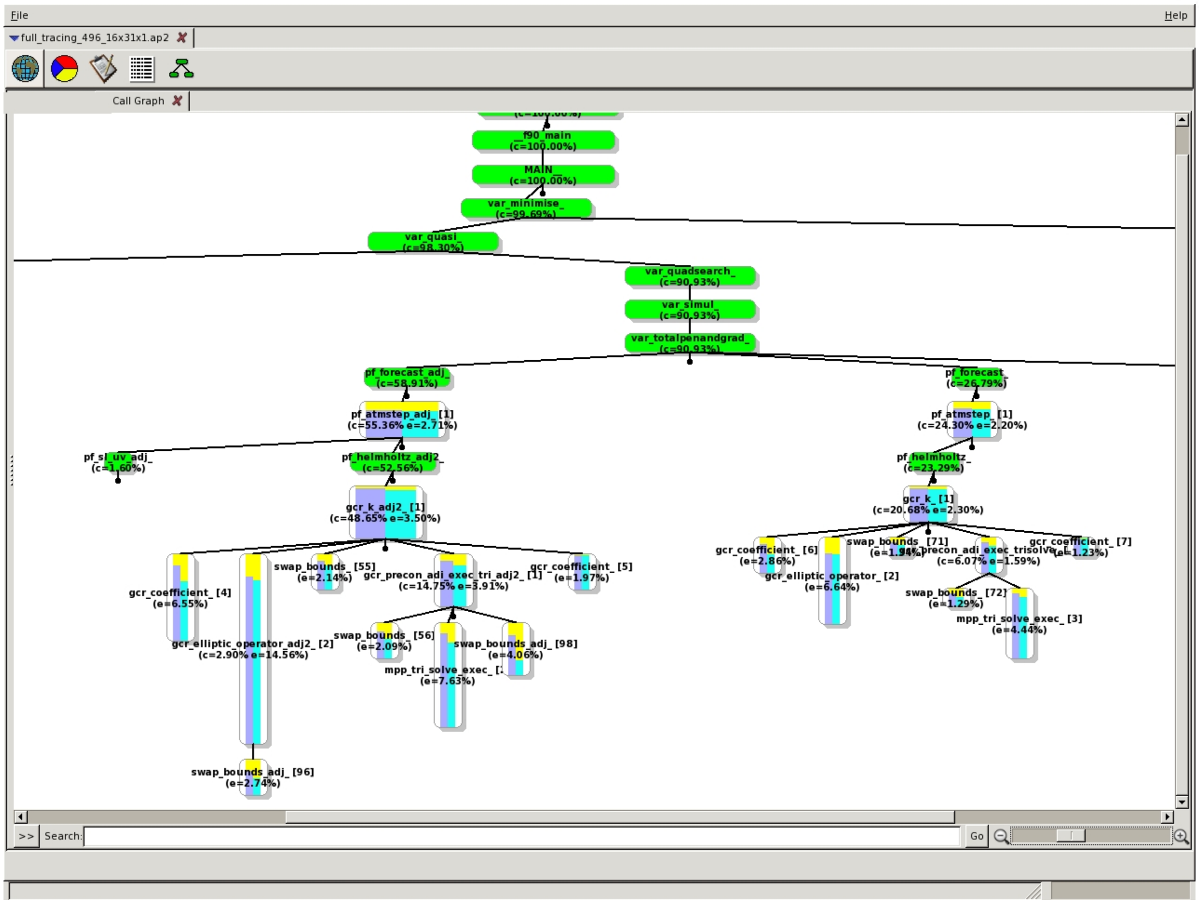




2/11/2019 Cray, Inc. Private 25

The left screen appears during data collection; later, the pie charts appear.

Apprentice2 call tree display



Reveal

- **Performance analysis and code restructuring assistant**
  - Integrated performance analysis and code optimization tool
  - Extends Cray's existing performance measurement, analysis, and visualization technology by combining run-time performance statistics and program source code visualization with Cray Compiling Environment (CCE) compile-time optimization feedback.

```
% module load PrgEnv-cray
% module load perftools

% cc –h pl=himeno.pl –hwp* himeno.c

% ftn –h pl=samp264.pl samp264.f
```

Use with compiler information only (no need to run program):

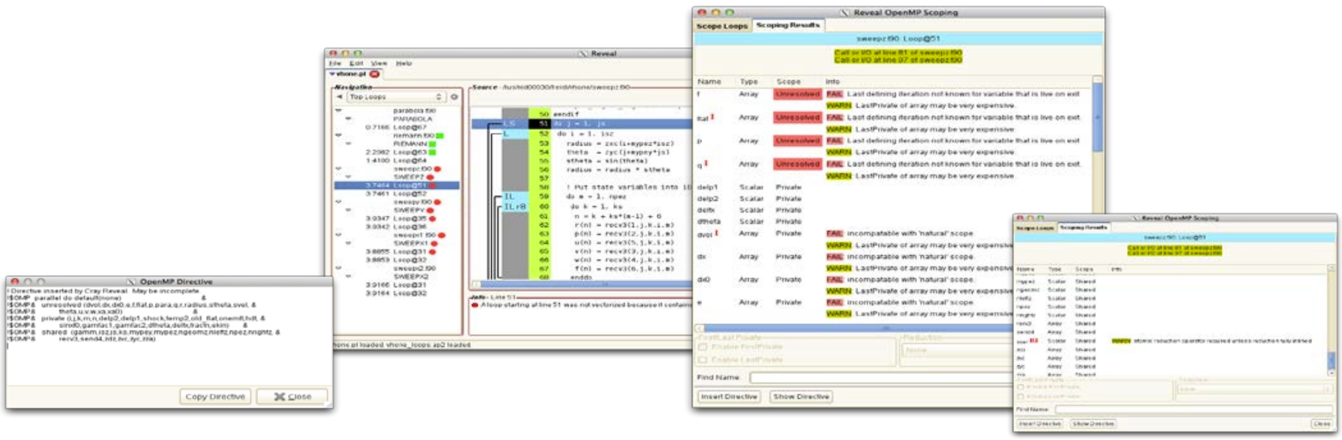
```
% reveal samp264.pl
```

Use with compiler + loop work estimates (include performance data):

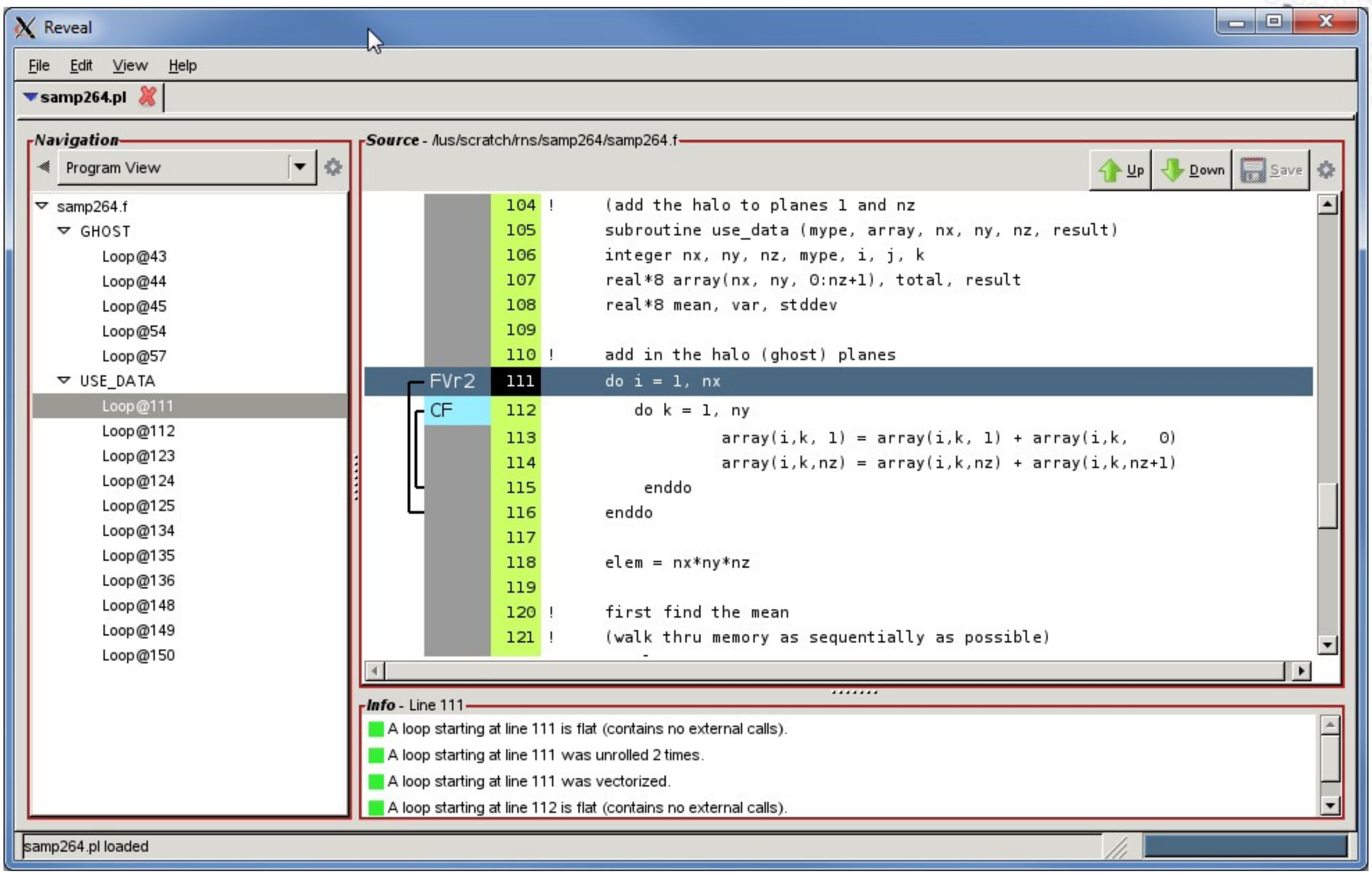
```
% reveal samp264.pl samp264_loops.ap2

* Optionally add whole program analysis for additional inlining.
```

Reveal

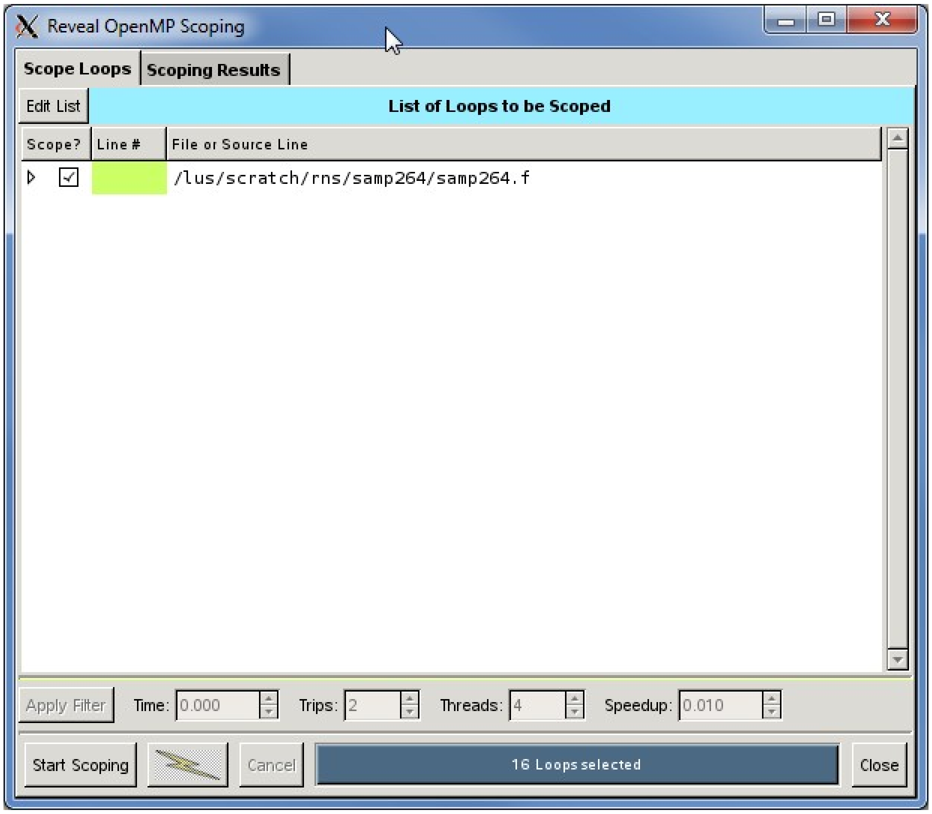
- **Navigate to relevant loops to parallelize**
- **Identify parallelization and scoping issues**
- **Get feedback on issues down the call chain**
  - Shared reductions, etc.
- **Optionally insert parallel directives into source**
- **Validate scoping correctness on existing directives**



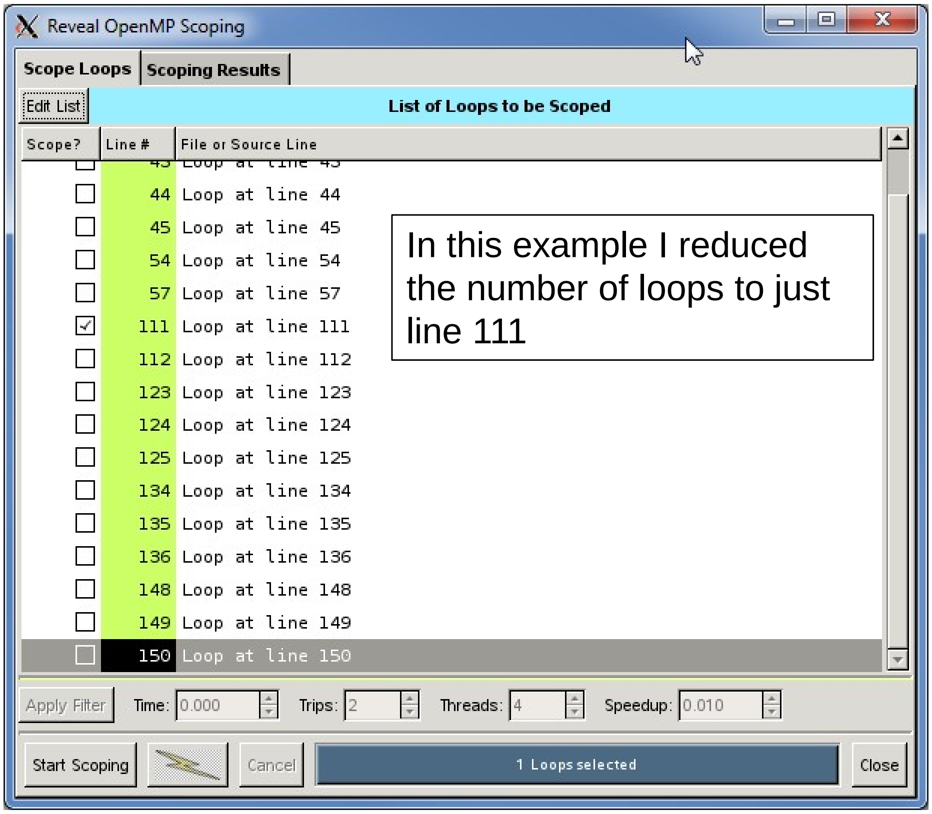
Reveal


Reveal
File Edit View Help
samp264.pl
Navigation
Program View
samp264.f
GHOST
Loop@43
Loop@44
Loop@45
Loop@54
Loop@57
USE_DATA
Loop@111
Loop@112
Loop@123
Loop@124
Loop@125
Loop@134
Loop@135
Loop@136
Loop@148
Loop@149
Loop@150
Source - /lus/scratch/rns/samp264/samp264.f
Up
Down
Save
104 ! (add the halo to planes 1 and nz
105 subroutine use_data (mype, array, nx, ny, nz, result)
106 integer nx, ny, nz, mype, i, j, k
107 real*8 array(nx, ny, 0:nz+1), total, result
108 real*8 mean, var, stddev
109
110 ! add in the halo (ghost) planes
FVr2 111 do i = 1, nx
CF 112 do k = 1, ny
113 array(i,k, 1) = array(i,k, 1) + array(i,k, 0)
114 array(i,k,nz) = array(i,k,nz) + array(i,k,nz+1)
115 enddo
116 enddo
117
118 elem = nx*ny*nz
119
120 ! first find the mean
121 ! (walk thru memory as sequentially as possible)
Info - Line 111
A loop starting at line 111 is flat (contains no external calls).
A loop starting at line 111 was unrolled 2 times.
A loop starting at line 111 was vectorized.
A loop starting at line 112 is flat (contains no external calls).
samp264.pl loaded

Reveal OpenMP Scoping

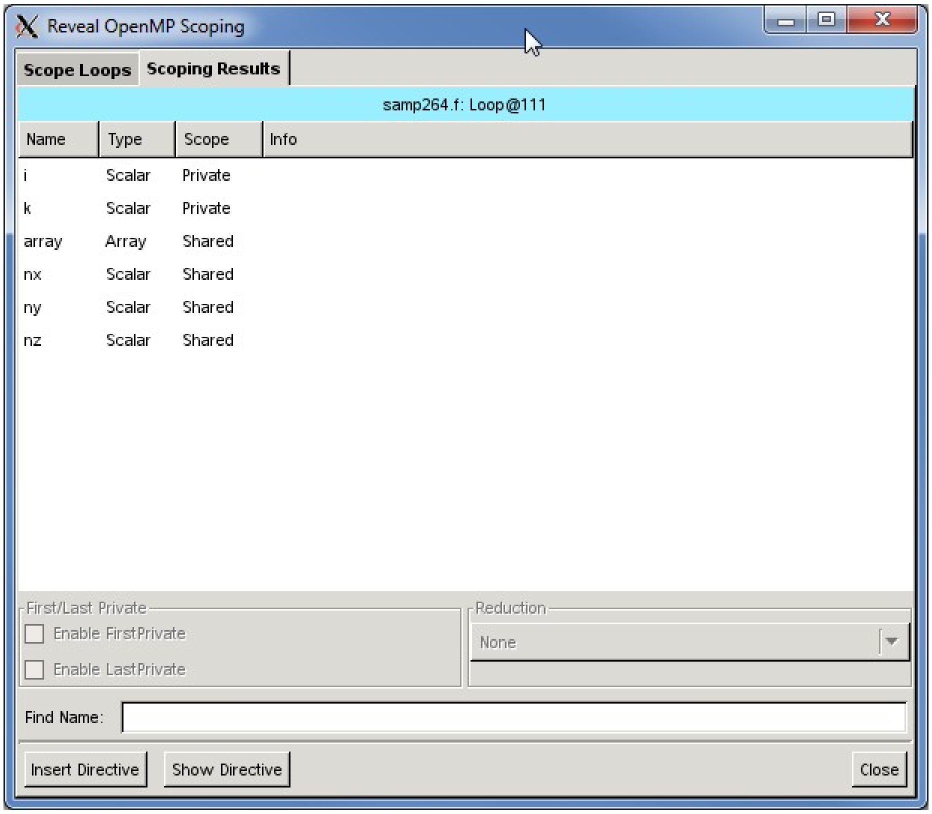


Reveal Loops to Scope



2/11/2019 Cray, Inc. Private 31

Reveal Scoping Results



Reveal Scoping Results