Performance Tools

Customer Documentation and Training

Topics

- CrayPat
- Apprentice2

Performance Tools

- CrayPat is a performance analysis tool that collects performance information from a user application
  - CrayPat 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 Opteron processor
- Cray Apprentice2 generates graphical displays from the .ap2 file

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 on front page**
  - **Additional man pages are hwpc and `papi_counters`, `intro_craypat`**

`pat_build` Sampling

- The lack of *tracing* options causes `pat_build` to default 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`, `samp_cs_ovfl`, `samp_ru_time`, `samp_ru_ovfl`, `samp_heap_time`, `samp_heap_ovfl`
  - Do not collect hardware counter information when you sample by overflow (for example< `samp_pc_ovfl`)
- Use *sampling* to obtain a profile and <u>then</u> *trace* functions of interest

Using CrayPat

- **To instrument a program:**
  - **Load the CrayPat module**

    % module load xt-craypat (perftools (CLE 3.1))

  - **The executable and object (.o) files are required**

```
% ftn -c prog.f90
% cc -c work.c
% ftn -o program1 prog.o work.o
```

  - **Or**

```
nid00008/rns> ftn -o samp264 samp264.f
/opt/cray/xt-asyncpe/3.4.4/bin/ftn: INFO: linux target is being used
WARNING: CrayPat is saving object files from temporary locations
into directory '/home/users/rns/.craypat/samp264/15976'
nid00008/rns>
```

Using CrayPat

- **Run `pat_build` to instrument the program**

```
nid00008/rns> pat_build samp264
nid00008/rns> ls –l samp26*

-rwxr-xr-x 1 rns hwpt  2001593 Sep 24 19:21 samp264
-rwxr-xr-x 1 rns hwpt  3592486 Sep 24 19:22 samp264+pat
```

– **Execute the instrumented program**

```
nid00008/rns> aprun -n 4 samp264+pat
```

Automatic Profiling Analysis (APA)

- The sample based instrumented program will generate a `.xf` file
  - Depending on environmental variable:
    - there is either a single `.xf` file (default) created
    - or a subdirectory with a `.xf` file for each processor used
- Run `pat_report`
  - `pat_report` will generate an `.ap2` and `.apa` file, as well as run a text report to `stdout`
  - The `.ap2` is used to generate additional text reports or by Apprentice2
  - The `.apa` file is used (optionally) to assist you in creating a trace based experiment file

```
nid00008/rns> pat_report samp264+pat+2885-203sdt.xf
nid00008/rns> ls –l samp264+*
-rw------- 1 rns hwpt      444 Sep 24 19:25 samp264.pbs.o1879481
-rw-r----- 1 rns hwpt     7240 Sep 24 19:25 samp264+pat+2885-203sdt.xf
-rw-r--r-- 1 rns hwpt     1568 Sep 24 19:26 samp264+pat+2885-203sdt.apa
-rw-r--r-- 1 rns hwpt    36864 Sep 24 19:26 samp264+pat+2885-203sdt.ap2
```

Automatic Profiling Analysis (APA)

- To use the **.apa** file to build a trace experiment file
  - No need to specify the executable
  - You should get an instrumented program samp264+apa

```
nid00008/rns> pat_build -O samp264+pat+2885-203sdt.apa
INFO: Trace intercept routine created for the 883 byte
function 'use_data_'.
nid00008/rns> ls –ltr
…
-rwxr-xr-x 1 rns hwpt  3599891 Sep 24 19:49 samp264+apa
```

- APA analysis specific support for:
  - Loop Count Statistics and Optimization Guidance
  - OpenMP support
  - PGAS (UPC) support
  - HW counter support

Automatic Profiling Analysis (APA)

- **Run application to get top time consuming routines**

  `% aprun -n 4 samp264+apa`

  - **The `.apa` file can be modified and used again by you**
  - **Use `pat_report` to view the `.xf` file**
  - **The `.apa` file includes `PAT_RT_HWPC=0`**

`pat_build` Trace Options

– **To trace functions and create the instrumented executable, use the following `pat_build` options:**

`–g [heap|stdio|io|…]` for one of the predefined groups

- **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` create new trace intercept routines for those function entry points that are defined in object and archive files

- **Instead of using the -u option, use the following options**
  - **With the PGI compilers, use `-Mprof=func`**
  - **With the GNU compilers, use `-finstrument-functions`**

`-w` creates new trace intercept routines for those function entry points where no trace intercept routine already exists

Other `pat_build` Options

- You can specify the name of resulting instrumented program with the `–o` option or by the final argument. If neither are specified, the `program` name is appended with `+pat`
  - `-A` enables the instrumented program to produce the data file that is accepted as input to Cray Apprentice2 (.ap2)
    - Requires that output be written to a file system that supports locking (such as a Lustre file system)
  - `-f` 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`)

Experiment Output

- **The experiment output file (or data file) is:**
  - **A directory with a file (`.xf`) for each process**
  - **A single `.xf` file**
    - **Requires that output be written to a file system that supports locking (such as a Lustre file system)**
    - **The file named `reduce+pat+3496-12tdt.xf` contains the following information: name of the instrumented program, `reduce+pat`; the process ID `3496`; the physical node—the application started on `12`; and the type of experiment performed**
  - **The `pat_report` command reads the experiment file(s) and produces a text or `.ap2` file**
    - **The `.ap2` file is used as input to Apprentice**
    - **The `.ap2` file can be used by `pat_report` to produce text output**
      - **The `.ap2` file is portable; it does not require the source or `.xf` files**

Environment Variables

- **`PAT_RT_SUMMARY`**
  - **0 turn off summary**
  - **1 enable summary (default)**
- **`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**

Environment Variables

- **`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**
- **`PAT_RT_HWPC`**
  - **Define the HWPC group**

pat_report

- **The default report is 'sample by function'; alternate views that use the `-o` option include:**
  - **`calltree`**
  - **`callers`**
  - **`load_balance`**

A Sequence of Commands

```
rns/crayPatExample> module load xt-craypat    # Loaded the CrayPat module for the XT system
rns/crayPatExample> ftn -o samp264 samp264.f  # compiled the code – simple application
rns/crayPatExample> pat_build samp264         # Created the experiment executable
rns/crayPatExample> vi samp264.pbs            # modify the job script to run samp64+pat
rns/crayPatExample> qsub samp264.pbs          # run the job
rns/crayPatExample> cat samp264.pbs.o1879623  # Made sure the job ran ☺
rns/crayPatExample> pat_report samp264+pat+15346-43sdt.xf > samp264+pat+15346.report
rns/crayPatExample> view samp264+pat+15346.report
rns/crayPatExample> pat_build -O samp264+pat+15346-43sdt.apa
rns/crayPatExample> ls -ltr
-rw-r--r-- 1 rns hwpt    5411 Sep 25 13:34 samp264.f
-rw-r--r-- 1 rns hwpt     306 Sep 25 13:34 samp264.pbs
-rwxr-xr-x 1 rns hwpt 2001625 Sep 25 13:35 samp264
-rwxr-xr-x 1 rns hwpt 3592502 Sep 25 13:35 samp264+pat
-rw------- 1 rns hwpt     459 Sep 25 13:36 samp264.pbs.o1879623
-rw-r----- 1 rns hwpt    7240 Sep 25 13:36 samp264+pat+15346-43sdt.xf
-rw-r--r-- 1 rns hwpt    5248 Sep 25 13:37 samp264+pat+15346.report
-rw-r--r-- 1 rns hwpt    1613 Sep 25 13:37 samp264+pat+15346-43sdt.apa
-rw-r--r-- 1 rns hwpt   36864 Sep 25 13:37 samp264+pat+15346-43sdt.ap2
-rwxr-xr-x 1 rns hwpt 3599971 Sep 25 13:53 samp264+apa

rns/crayPatExample> vi samp264.pbs            # modify the job script to run samp64+apa
rns/crayPatExample> qsub samp264.pbs          # run the job
rns/crayPatExample> pat_report samp264+apa+8557-142tdt.xf > samp264+apa+8557.report
rns/crayPatExample> view samp264+apa+8557.report
```

view samp264+pat+15346.report

```
Table 1:  Profile by Function
 Samp % | Samp | Imb. |   Imb. |Group
        |      | Samp | Samp % | Function
        |      |      |        |  PE='HIDE'
 100.0% |  959 |   -- |     -- |Total
|------------------------------------------
|  99.9% |  958 |   -- |     -- |USER
||-----------------------------------------
||  96.6% |  926 | 0.75 |   0.1% |use_data_
||   3.3% |   32 | 0.00 |   0.0% |MAIN_
|==========================================

Table 2:  Profile by Group, Function, and Line

 Samp % | Samp | Imb. |   Imb. |Group
        |      | Samp | Samp % | Function
        |      |      |        |  Source
        |      |      |        |   Line
        |      |      |        |    PE='HIDE'
 100.0% |  961 |   -- |     -- |Total
|--------------------------------------------
|  99.9% |  960 |   -- |     -- |USER
||-------------------------------------------
||  96.6% |  928 |   -- |     -- |use_data_
3|        |      |      |        | rns/crayPatExample/./samp264.f
||||-----------------------------------------
4|||    5.1% |   49 | 1.50 |   4.0% |line.123
4|||    6.1% |   59 | 1.00 |   2.2% |line.134
4|||   84.9% |  816 | 1.00 |   0.2% |line.148
||||=========================================
||   3.3% |   32 | 0.00 |   0.0% |MAIN_
3|        |      |      |        | rns/crayPatExample/./samp264.f
4|        |      |      |        |  line.43
|===========================================
```

This is the report from the first "sample" experiment.
Table 1 shows the highest used functions, MAIN and use_data

Table 2 show more detail about those functions and provides information on some of the loops in the use_data function. (These are generated when there is enough data (samples) to produce this information, small codes may not report such detail.)

view samp264+apa+8557.report

```
Table 2:  Profile by Function Group and Function
Group / Function / PE='HIDE'
===========================================
Totals for program  (OMITTED BY INSTRUCTOR)
-------------------------------------------
===========================================
USER                (OMITTED BY INSTRUCTOR)
-------------------------------------------
========================================================================
USER / use_data_
------------------------------------------------------------------------
  Time%                                              96.3%
  Time                                           14.805327 secs
  Imb.Time                                        0.007424 secs
  Imb.Time%                                           0.1%
  Calls                             2.0 /sec          30.0 calls
  PAPI_L1_DCM                    12.355M/sec     182917104 misses
  PAPI_TLB_DM                     5.849M/sec      86592140 misses
  PAPI_L1_DCA                   208.397M/sec    3085410313 refs
  PAPI_FP_OPS                   145.973M/sec    2161200000 ops
  User time (approx)             14.805 secs   34052604860 cycles  100.0%Time
  Average Time per Call                           0.493511 sec
  CrayPat Overhead : Time          0.0%
  HW FP Ops / User time         145.973M/sec    2161200000 ops   1.6%peak(DP)
  HW FP Ops / WCT               145.973M/sec
  Computational intensity          0.06 ops/cycle    0.70 ops/ref
  MFLOPS (aggregate)             583.89M/sec
  TLB utilization                 35.63 refs/miss   0.070 avg uses
  D1 cache hit,miss ratios        94.1% hits          5.9% misses
  D1 cache utilization (misses)  16.87 refs/miss    2.108 avg hits
========================================================================
USER / main         (OMITTED BY INSTRUCTOR)
------------------------------------------------------------------------
```

This is the report generated after `pat_build -O samp264+pat+15346-43sdt.apa` was executed and the executable `samp264+apa` was run. The APA file suggested `HWPC value 1 be used.` This is where the performance counter data comes form in Table 2

Hardware Performance Counters

– **The APA file makes suggestions about what hardware performance counters should be used**
  - **To use different performance counter set the PAT_RT_HWPC ENVIRONMENTAL variable and rerun the job.**

```
rns/crayPatExample> cat samp264+pat+15346-43sdt.apa
 [clipped]
# ----------------------------------------------------------------------
#       HWPC group to collect by default.
  -Drtenv=PAT_RT_HWPC=1  # Summary with TLB metrics.
# ----------------------------------------------------------------------

rns/crayPatExample> cat samp264.pbs
#!/bin/ksh
#PBS -j oe
#PBS -l mppwidth=4
#PBS -l walltime=00:30:00
export PAT_RT_HWPC=0
cd $PBS_O_WORKDIR
#aprun -n 4 ./samp264
#aprun -n 4 ./samp264+pat
aprun -n 4 ./samp264+apa
```

Looking Closer

– **Load the correct modules**
– **Since you are probably interested in hardware counters for only a narrow range of code, use the CrayPat API to identify the region of interest.**
  - **In C/C++**

```
#include <pat_api.h>

PAT_region_begin(1,"loop");
…
PAT_region_end(1);
```

  - **In Fortran**

```
#include <pat_apif.h>

call PAT_region_begin(1,"loop“,stat);
…
call PAT_region_end(1,stat);
```

Looking Closer

- Compile your code.
- Use `pat_build` to relink and create an instrumented binary.
- Use the environment variable PAT_RT_HWPC to select the hardware counters that you want to collect.

  `PAT_RT_HWPC=0`
- You can also save your favorite counters in a file and pass them to CrayPat
  - Add file name to `PAT_RT_HWPC_FILE` environment variable

Looking Closer

```
!     first find the mean
!     (walk thru memory as sequentially as possible)
      call PAT_region_begin(1, "halo_loop", istat)
      total = 0.0
      do k = 1, nz
          do j = 1, ny
              do i = 1, nx
                  total = total + array(i, j, k)
              enddo
          enddo
      enddo
      call PAT_region_end(1, istat)
```

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

 Time % |       Time |Imb. Time |   Imb. | Calls |Group
        |            |          | Time % |       | Function
        |            |          |        |       |  PE='HIDE'
 100.0% | 16.066011 |        -- |     -- |  65.0 |Total
|-----------------------------------------------------------
| 100.0% | 16.065887 |        -- |     -- |  63.0 |USER
||----------------------------------------------------------
||  87.9% | 14.121553 | 0.012971 |   0.1% |  30.0 |use_data_
||   8.6% |  1.385652 | 0.001772 |   0.2% |  30.0 |#1.halo_loop
||   3.5% |  0.558584 | 0.011731 |   2.7% |   1.0 |MAIN_
|===========================================================
```

Looking closer

```
USER / #1.halo_loop
------------------------------------------------------------------------
  Time%                                                   8.6%
  Time                                                1.385652 secs
  Imb.Time                                            0.001772 secs
  Imb.Time%                                               0.2%
  Calls                                 21.6 /sec         30.0 calls
  PAPI_L1_DCM                          0.791M/sec      1095556 misses
  PAPI_TOT_INS                      1969.692M/sec   2729544748 instr
  PAPI_L1_DCA                        923.223M/sec   1279376922 refs
  PAPI_FP_OPS                        130.757M/sec    181200000 ops
  User time (approx)                   1.386 secs   3187276531 cycles
100.0%Time
  Average Time per Call                               0.046188 sec
  CrayPat Overhead : Time               0.0%
  HW FP Ops / User time              130.757M/sec    181200000 ops
1.4%peak(DP)
  HW FP Ops / WCT                    130.757M/sec
  HW FP Ops / Inst                                        6.6%
  Computational intensity               0.06 ops/cycle    0.14 ops/ref
  Instr per cycle                                         0.86 inst/cycle
  MIPS                               7878.77M/sec
  MFLOPS (aggregate)                  523.03M/sec
  Instructions per LD & ST             46.9% refs         2.13 inst/ref
  D1 cache hit,miss ratios             99.9% hits         0.1% misses
  D1 cache utilization (misses)      1167.79 refs/miss  145.973 avg hits
========================================================================
```

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**

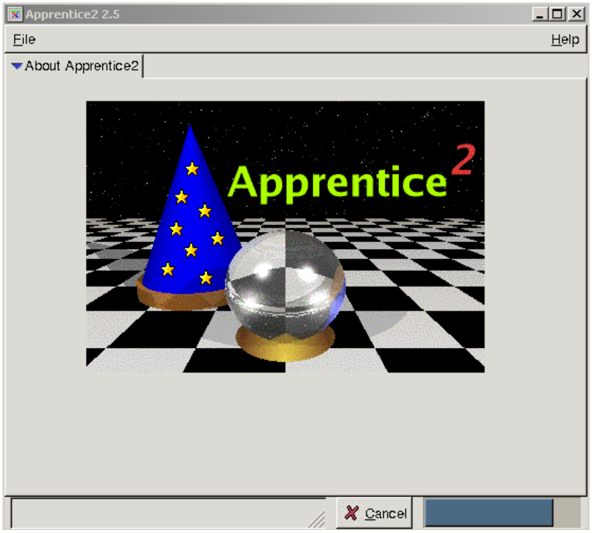
```
aprun -n 1 /opt/xt-tools/papi/default/bin/papi_avail
```

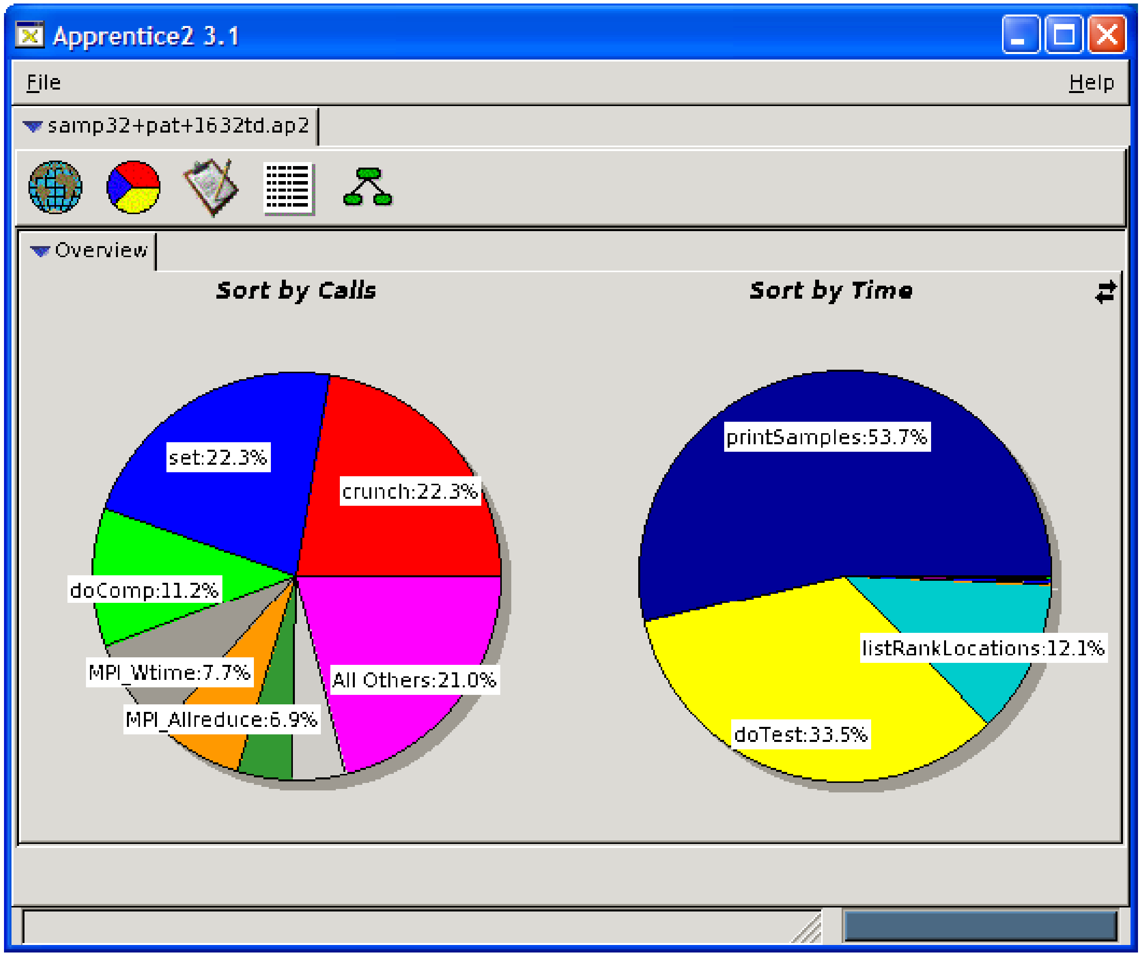
PAPI Terminology

- **An event set is a group of native events, preset events, or a combination of both**
  - **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**

Cray Apprentice2

- `% module load apprentice2`
- `% app2 program1+pat+180tdo-0000.ap2`





Apprentice2 call tree display