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 **then** *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'
    ```
Using CrayPat

- Run `pat_build` to instrument the program

```bash
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

```bash
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` file 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`)
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_EXPRFILE_NAME**
  - The experiment file name

- **PAT_RT_EXPRFILE_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
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> 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------ 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
Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb.</th>
<th>Func.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>959</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>99.9%</td>
<td>958</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>96.6%</td>
<td>926</td>
<td>0.75</td>
<td>0.1%</td>
<td>use_data</td>
</tr>
<tr>
<td>3.3%</td>
<td>32</td>
<td>0.00</td>
<td>0.0%</td>
<td>MAIN</td>
</tr>
</tbody>
</table>

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

Table 2: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb.</th>
<th>Func.</th>
<th>Source</th>
<th>Line</th>
<th>PE='HIDE'</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>961</td>
<td>--</td>
<td>--</td>
<td>Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.9%</td>
<td>960</td>
<td>--</td>
<td>--</td>
<td>USER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96.6%</td>
<td>928</td>
<td>--</td>
<td>--</td>
<td>use_data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3%</td>
<td>32</td>
<td>0.00</td>
<td>0.0%</td>
<td>MAIN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.)
Table 2: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Group / Function / PE='HIDE'</th>
<th>Totals for program (OMITTED BY INSTRUCTOR)</th>
</tr>
</thead>
</table>

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 from in Table 2.

<table>
<thead>
<tr>
<th>USER / use_data_</th>
<th>Time%</th>
<th>96.3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>14.805327 secs</td>
<td></td>
</tr>
<tr>
<td>Imb.Time</td>
<td>0.007424 secs</td>
<td></td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>0.1%</td>
<td></td>
</tr>
<tr>
<td>Calls</td>
<td>2.0 /sec</td>
<td></td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>12.355M/sec</td>
<td></td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>5.849M/sec</td>
<td></td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>208.397M/sec</td>
<td></td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>145.973M/sec</td>
<td></td>
</tr>
<tr>
<td>User time (approx)</td>
<td>14.805 secs</td>
<td></td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.493511 sec</td>
<td></td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>145.973M/sec</td>
<td></td>
</tr>
<tr>
<td>HW FP Ops / WCT</td>
<td>145.973M/sec</td>
<td></td>
</tr>
<tr>
<td>Computational intensity</td>
<td>0.06 ops/cycle</td>
<td></td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
<td>583.89M/sec</td>
<td></td>
</tr>
<tr>
<td>TLB utilization</td>
<td>35.63 refs/miss</td>
<td></td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>94.1% hits</td>
<td></td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>16.87 refs/miss</td>
<td></td>
</tr>
</tbody>
</table>

This is where the performance counter data comes from 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
```
– 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++

```c
#include <pat_api.h>

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

- In Fortran

```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

<table>
<thead>
<tr>
<th>Time %</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>16.066011</td>
<td>--</td>
<td>--</td>
<td>65.0</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>16.065887</td>
<td>--</td>
<td>--</td>
<td>63.0</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87.9%</td>
<td>14.121553</td>
<td>0.012971</td>
<td>0.1%</td>
<td>30.0</td>
<td>use_data_</td>
</tr>
<tr>
<td>8.6%</td>
<td>1.385652</td>
<td>0.001772</td>
<td>0.2%</td>
<td>30.0</td>
<td>#1.halo_loop</td>
</tr>
<tr>
<td>3.5%</td>
<td>0.558584</td>
<td>0.011731</td>
<td>2.7%</td>
<td>1.0</td>
<td>MAIN_</td>
</tr>
</tbody>
</table>

10/18/2010 Cray Private
Looking closer

**USER / #1.halo_loop**

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

10/18/2010 Cray Private
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
• % module load apprentice2
• % app2 program1+pat+180tdo-0000.ap2
Apprentice2 call tree display