Introduction to Cray Performance Analysis Tools

February 2013
Don Bahls
User Consultant
dmbahls@alaska.edu
Overview

- What is CrayPat?
- CrayPat tools
  - `pat_build`
    - Trace groups
    - Tracing user subroutines
    - Hardware performance counters
  - `pat_report`
  - `app2`
  - `reveal` (Cray Compiler only)

References
Purpose of this Training

- This lecture is about a set of performance analysis tools not on code optimization. If you have questions on code optimization you can talk to the ARSC Help Desk.

- Some general optimization recommendations:
  - Try optimizing serial case first if possible
  - Start with most time consuming routines
  - For parallel code look for routines with significant load imbalance.
What is CrayPat?

- CrayPat is a set of tools that can be used to gather performance information from an executable including:
  - Timing information for subroutines
  - Hardware counter information.
  - MPI, IO, heap, and other performance data.
Why CrayPat?

- Instrumentation can be done without making changes to the original source code.
- For finer grained instrumentation, CrayPat includes a C and Fortran API. (This will not be covered in this lecture). See chapter 2 of S-2376-60.
CrayPat Caveats

- Use the “cc”, “CC”, and “ftn” compiler wrappers and not gcc, gfortran, craycc or crayftn, etc.
- Use the PrgEnv-pgi or PrgEnv-cray programming environments and load perftools before compiling!
- Make sure your Makefile keeps object files around (needed for CrayPat).
  - e.g. cc mycode.c -o mycode (won’t work)
CrayPat Tools

- **pat_build**: generates an instrumented executable for performance analysis
- **pat_report**: generates a report based on output from a CrayPat generated executable.
pat_build

- **pat_build** lets you generate an instrumented executable.

- Common **pat_build** options include:
  - `-u` trace routines in your source code.
  - `-g group` trace routines in the specified group(s).
  - `-D directive` change the behavior of **pat_build**.
Some Available pat_build Groups

<table>
<thead>
<tr>
<th>armci</th>
<th>fftw</th>
<th>io</th>
<th>netcdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>blacs</td>
<td>ga</td>
<td>lapack</td>
<td>oacc</td>
</tr>
<tr>
<td>blas</td>
<td>gni</td>
<td>lustre</td>
<td>petsc</td>
</tr>
<tr>
<td>caf</td>
<td>hdf5</td>
<td>math</td>
<td>stdio</td>
</tr>
<tr>
<td>chapel</td>
<td>heap</td>
<td>mpi</td>
<td>upc</td>
</tr>
</tbody>
</table>

# trace mpi routines
% pat_build –g mpi ./a.out

# trace mpi, io, heap related routines
% pat_build –g mpi,io,heap ./a.out
Example use of pat_build to profile

```bash
% module load perftools
% make arch=fish
% cd bin/fish
% pat_build -g mpi -u ./xhpl
# generates xhpl+pat

% cat run_HPL
# run the executable as you normally would
aprun -n 8 ./xhpl+pat
% qsub run_HPL
```
Output from pat_build

% pat_build -g mpi -u ./xhpl
INFO: Trace intercept routine created for the 12351 byte function 'HPL_pdinfo'.
INFO: Trace intercept routine created for the 2693 byte function 'HPL_pdtst'.
INFO: Trace intercept routine created for the 199 byte function 'HPL_fprintf'.
INFO: Trace intercept routine created for the 239 byte function 'HPL_dgemv'.
INFO: Trace intercept routine created for the 541 byte function 'HPL_grid_init'.
...
INFO: Trace intercept routine created for the 107 byte function 'HPL_lmul'.
INFO: Trace intercept routine created for the 248 byte function 'HPL_perm'.
INFO: Trace intercept routine created for the 534 byte function 'HPL_logsort'.
WARNING: Tracing small, frequently called functions can add excessive overhead.
INFO: To set a minimum size, say 800 bytes, for traced functions, use:
-D trace-text-size=800
Instrumented Executable Output

- When run, an instrumented executable will produce a *xf file with performance data for the job.
- The pat_report command can be generate a text report and a *.ap2 file.
Output from pat_report

```
fish1 % pat_report xhpl+pat+3472-2t.xf
CrayPat/X:  Version 5.3.2 Revision 9087 (xf 8679)  04/17/12 12:31:06

Number of PEs (MPI ranks):   16
Numbers of PEs per Node:     16
Numbers of Threads per PE:   1
Number of Cores per Socket:  16
Execution start time:  Wed Jan 30 08:06:46 2013
System type and speed:  x86_64 2100 MHz
```
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>100.0%</th>
<th>455.991641</th>
<th>--</th>
<th>--</th>
<th>320433026.6</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>45.2%</td>
<td>205.906449</td>
<td>--</td>
<td>--</td>
<td>317509961.5</td>
<td>USER</td>
</tr>
<tr>
<td>18.8%</td>
<td>85.598093</td>
<td>7.109473</td>
<td>8.2%</td>
<td>78243777.1</td>
<td>HPL_setran</td>
</tr>
<tr>
<td>9.4%</td>
<td>42.724728</td>
<td>5.397195</td>
<td>12.0%</td>
<td>78128125.0</td>
<td>HPL_rand</td>
</tr>
<tr>
<td>8.5%</td>
<td>38.571844</td>
<td>3.988142</td>
<td>10.0%</td>
<td>2.0</td>
<td>HPL_pdmatgen</td>
</tr>
<tr>
<td>4.3%</td>
<td>19.415649</td>
<td>2.917556</td>
<td>13.9%</td>
<td>78376024.1</td>
<td>HPL_lmul</td>
</tr>
<tr>
<td>1.9%</td>
<td>8.642060</td>
<td>0.910804</td>
<td>10.2%</td>
<td>78309904.1</td>
<td>HPL_ladd</td>
</tr>
<tr>
<td>39.9%</td>
<td>181.976505</td>
<td>--</td>
<td>--</td>
<td>1657781.5</td>
<td>BLAS</td>
</tr>
<tr>
<td>36.4%</td>
<td>166.145470</td>
<td>18.609812</td>
<td>10.7%</td>
<td>811228.2</td>
<td>dgemm_</td>
</tr>
<tr>
<td>3.3%</td>
<td>15.247237</td>
<td>1.106441</td>
<td>7.2%</td>
<td>811228.2</td>
<td>dtrsm_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
...
Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb. Time</th>
<th>Imb. Time%</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
</tbody>
</table>

...  

| 14.9% | 68.108622 | -- | -- | 1265183.6 | MPI |

| 9.6%  | 43.685509 | 14.580971 | 26.7% | 115.1 | MPI_Recv |
| 4.9%  | 22.176232 | 9.627945 | 32.3% | 12729.1 | MPI_Send |

Imbalance Percent
Imbalance Time (Max-Avg)
Average Time
Percent Time spent in MPI_Recv
<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>MPI</th>
<th>MPI Msg</th>
<th>Avg MPI</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>575.428553</td>
<td>12729.1</td>
<td>128546315.8</td>
<td>100985.15</td>
<td>Total</td>
</tr>
</tbody>
</table>

...  

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11.9%</td>
<td>68.580201</td>
<td>12729.1</td>
<td>128546315.8</td>
<td>100985.15</td>
<td>MPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.6%</td>
<td>90.017115</td>
<td>12509.0</td>
<td>1225363692.0</td>
<td>97958.57</td>
<td>pe.6</td>
</tr>
<tr>
<td>12.5%</td>
<td>72.176595</td>
<td>12515.0</td>
<td>1248247808.0</td>
<td>99740.14</td>
<td>pe.1</td>
</tr>
<tr>
<td>5.5%</td>
<td>31.712696</td>
<td>13412.0</td>
<td>1359053712.0</td>
<td>101331.17</td>
<td>pe.0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</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>PE</td>
</tr>
<tr>
<td>100.0%</td>
<td>455.991641</td>
<td>--</td>
<td>--</td>
<td>320433026.6</td>
<td>Total</td>
</tr>
</tbody>
</table>

```
fish% pat_report -s pe=ALL xhpl+pat+3472-2t.xf
...

Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</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>PE</td>
</tr>
<tr>
<td>14.9%</td>
<td>68.108622</td>
<td>--</td>
<td>--</td>
<td>1265183.6</td>
<td>MPI</td>
</tr>
</tbody>
</table>

| 9.6% | 43.685509 | 14.580971 | 26.7% | 115.1 | MPI_Recv |

| 12.8% | 58.266480 | --   | --   | 119.0 | pe.7   |
| 12.4% | 56.459500 | --   | --   | 114.0 | pe.6   |

| 6.7% | 30.399617 | --   | --   | 115.0 | pe.12  |
| 5.4% | 24.465880 | --   | --   | 138.0 | pe.8   |
| 4.6% | 20.968014 | --   | --   | 136.0 | pe.0   |
```
### Extended PE Info Cont.

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Time (s)</th>
<th>Wall Time (s)</th>
<th>Memory (MB)</th>
<th>Total (s)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>5.190040</td>
<td>2159.4</td>
<td>12549656.0</td>
<td>5811.71</td>
<td>Total</td>
</tr>
<tr>
<td>63.0%</td>
<td>3.268430</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>12.2%</td>
<td>5.074357</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>pe.1</td>
</tr>
<tr>
<td>12.2%</td>
<td>5.058965</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>pe.0</td>
</tr>
<tr>
<td>12.2%</td>
<td>5.050925</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>pe.2</td>
</tr>
<tr>
<td>37.0%</td>
<td>1.921609</td>
<td>2159.4</td>
<td>12549656.0</td>
<td>5811.71</td>
<td>MPI</td>
</tr>
<tr>
<td>10.7%</td>
<td>4.440580</td>
<td>578.0</td>
<td>6008772.0</td>
<td>10395.80</td>
<td>pe.7</td>
</tr>
<tr>
<td>10.1%</td>
<td>4.189313</td>
<td>560.0</td>
<td>8125716.0</td>
<td>14510.21</td>
<td>pe.3</td>
</tr>
<tr>
<td>5.2%</td>
<td>2.145788</td>
<td>2707.0</td>
<td>16026420.0</td>
<td>5920.36</td>
<td>pe.6</td>
</tr>
</tbody>
</table>
Other pat_report options.

The –O option can produce a number of different reports (here are a few).

<table>
<thead>
<tr>
<th>accelerator</th>
<th>heap</th>
<th>mpi_callers</th>
</tr>
</thead>
<tbody>
<tr>
<td>accpc</td>
<td>heap_program</td>
<td>mpi_rank_order</td>
</tr>
<tr>
<td>acc_fu</td>
<td>heap_hiwater</td>
<td>program_time</td>
</tr>
<tr>
<td>acc_time</td>
<td>heap_leaks</td>
<td>samp_profile+src</td>
</tr>
<tr>
<td>profile</td>
<td>load_balance</td>
<td>thread_times</td>
</tr>
</tbody>
</table>
Apprentice2

- When pat_report is run, an Apprentice2 file is generated.
- Apprentice2 is a GUI based application for visualizing performance analysis data.
- Available via the “perftools” module:
- Version 6.0.0+ does some assessment of results

```bash
% module load perftools/6.0.1
% app2 xhpl+pat+3383-13tdt.ap2
```
Performance Overview (CrayPat 6+)

**Function/Region Profile**
- 36.4% = dgemm_
- 18.8% = HPL_setran
- 9.6% = MPI_Recv

**Load Imbalance**
- 18.61s = dgemm_
- 7.11s = HPL_setran
- 14.58s = MPI_Recv

**Profile CPU**
- Computation: 85.10%
- Programming Model: 14.90%
  - IO: 0.00%

**Memory Utilization**
- Process HiMem (MBytes): 447.280

**Data Movement**
- MPI Msg.Bytes: 12854463158
Call Counts and Call Timing

Sort by Calls

- HPL_ladd: 24.4%
- HPL_lmul: 24.5%
- HPL_setran: 24.4%
- HPL_rand: 24.4%

Sort by Time

- HPL_setran: 19.9%
- dgemm: 28.9%
- HPL_ladd: 6.6%
- HPL_lmul: 8.5%
- MPI_Recv: 7.6%
- tgen: 6.7%
- All Others: 9.3%

Overview, Environment, Profile Report
Routine Level Detail (dgemm_)
Call Graph Overview

- Horizontal size - relative time spent in node’s children.
- Vertical time spent in the function itself.
- Nodes that contain only callers are green.
- Yellow bar - max time
- Pale Purple - min time
- Purple bar - average time
- Hovering over node show more info
Hardware Performance Counters

- The AMD Opteron processor has a number of built-in hardware counters.
- The PAPI interface provides user access to these counters.
- CrayPat provides access to hardware counters via PAPI.
Hardware performance counters groups can be selected via the PAT_RT_HWPC environment variable.

```bash
# gather L1 and L2 metrics (group 2)
export PAT_RT_HWPC=2
aprun -n 8 ./xhpl+pat
```

These provide access of groups of counters that Cray has found useful.

See “man hwpc” for performance counter group options.
<table>
<thead>
<tr>
<th>HWPC Group</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Summary with instruction metrics</td>
</tr>
<tr>
<td>1</td>
<td>Summary with TLB metrics</td>
</tr>
<tr>
<td>2</td>
<td>L1 and L2 metrics</td>
</tr>
<tr>
<td>3</td>
<td>Bandwidth information</td>
</tr>
<tr>
<td>4</td>
<td>Hypertransport info (not supported Interlagos)</td>
</tr>
<tr>
<td>5</td>
<td>Floating point mix</td>
</tr>
<tr>
<td>6</td>
<td>Cycles stalled, resources idle</td>
</tr>
<tr>
<td>7</td>
<td>Cycles stalled, resources full</td>
</tr>
<tr>
<td>8</td>
<td>Instructions and branches</td>
</tr>
<tr>
<td>9</td>
<td>Instruction cache</td>
</tr>
<tr>
<td>10</td>
<td>Cache hierarchy (not supported Interlagos)</td>
</tr>
<tr>
<td>HWPC Group</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>11</td>
<td>Floating point operations mix (2)</td>
</tr>
</tbody>
</table>
| 12         | Floating point operations mix (vectorization)  
Or Dual-pipe floating point operations dispatched (Interlagos) |
| 13         | Floating point operations mix (SP) |
| 14         | Floating point operations mix (DP) |
| 15         | L3 (socket-level) (not supported Interlagos) |
| 16         | L3 (core-level reads) (not supported Interlagos) |
| 17         | L3 (core-level misses) (not supported Interlagos) |
| 18         | L3 (core-level fills caused by L2 evictions)  
(not supported Interlagos) |
| 19         | Prefetches |
## Totals for program

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Time</td>
<td>8.504229 secs</td>
</tr>
<tr>
<td>Imb.Time</td>
<td>-- secs</td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>--</td>
</tr>
<tr>
<td>Calls</td>
<td>0.348M/sec 3246329.4 calls</td>
</tr>
<tr>
<td>REQUESTS_TO_L2:DATA</td>
<td>33.896M/sec 315997615 req</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS:</td>
<td></td>
</tr>
<tr>
<td>L2_MODIFIED:L2_OWNED:</td>
<td></td>
</tr>
<tr>
<td>L2_EXCLUSIVE:L2_SHARED</td>
<td>30.850M/sec 287598348 fills</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_SYSTEM:</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>0.150M/sec 1400089 fills</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>924.789M/sec 8621346605 refs</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>9.322 secs 21441750000 cycles 100.0%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000003 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>54.8%</td>
</tr>
<tr>
<td>D1 cache hit,miss ratio (R)</td>
<td>96.6% hits 3.4% misses</td>
</tr>
<tr>
<td>D1 cache utilization</td>
<td>29.83 refs/refill 3.729 avg uses</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>99.6% hits 0.4% misses</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
<td>100.0% hits 0.0% misses</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>6157.71 refs/miss 769.714 avg uses</td>
</tr>
<tr>
<td>System to D1 refill</td>
<td>0.150M/sec 1400089 lines</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
<td>9.166MB/sec 89605680 bytes</td>
</tr>
<tr>
<td>L2 to Dcache bandwidth</td>
<td>1882.929MB/sec 18406294280 bytes</td>
</tr>
</tbody>
</table>
You can also select individual counters via the PAT_RT_HWPC environment variable.

```bash
# gather PAPI L1 and L2 data cache misses
% export PAT_RT_HWPC=PAPI_L1_DCM,PAPI_L2_DCM
aprun -n 8 ./xhpl+pat
```

Run the papi_avail command to get available counters. (module load papi)

NOTE: there are limitations on counters.
Totals for program

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Time</td>
<td>344.482603 secs</td>
</tr>
<tr>
<td>Imb.Time</td>
<td>-- secs</td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>--</td>
</tr>
<tr>
<td>Calls</td>
<td>2.905M/sec 293545262.4 calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>39.396M/sec 3980920255 misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>4.047M/sec 408931670 misses</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>101.050 secs 232413993750 cycles 29.3%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000001 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>103.0%</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>89.7% hits 10.3% misses</td>
</tr>
</tbody>
</table>

PAT_RT_HWPC=PAPI_L1_DCM,PAPI_L2_DCM

USER

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>88.0%</td>
</tr>
<tr>
<td>Time</td>
<td>303.161934 secs</td>
</tr>
<tr>
<td>Imb.Time</td>
<td>-- secs</td>
</tr>
<tr>
<td>Imb.Time%</td>
<td>--</td>
</tr>
<tr>
<td>Calls</td>
<td>4.793M/sec 292504062.8 calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>25.204M/sec 1538041148 misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>3.099M/sec 189101751 misses</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>61.024 secs 140354215312 cycles 20.1%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000001 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>116.6%</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>87.7% hits 12.3% misses</td>
</tr>
</tbody>
</table>
Hardware Counters in app2
Observations

- Be sure to time your application without CrayPat, so you can ensure the instrumented executable performance is comparable to the original version.
- You can use `pat_build -D trace-text-size=800` to limit the size of profiled routines if CrayPat introduces performance issues. Here “800” limits the size of profiled routines to 800 bytes.
More Observations

- We have seen some cases where CrayPat generated executables crash when the original code didn’t.
- If that happens let us know and we look into it and open a case with Cray.
- It’s normally possible to disable the call(s) causing the crash.
Reveal

- Reveal is a new tool included in CrayPat 6+.
- It allows compiler feedback information to be viewed along with performance information.
- Reveal must be used with the Cray Compiler suite.
Basic workflow

- module swap PrgEnv-pgi PrgEnv-cray
- Make sure cce version is 8 or newer! (module list cce)
- module load perftools/6.0.1
- Add –h profile_generate to compile flags
- This shuts off most optimizations and gets loop count information.
- cc –c –h profile_generate mycode.c
- cc –o mycode –h profile_generate mycode.o
Basic Workflow continued

- `pat_build -w mycode`
- `aprun -n 1 ./mycode+pat`
- `pat_report mycode+pat.XXX.xf`

- Recompile code using the "-hpl=mylib" flag
  - `cc -c -hpl=mylib mycode.c`
  - `cc -o mycode -hpl=mylib mycode.o`
Lastly

- Startup Reveal with the program library and apprentice2 file.
- `reveal mylib mycode.XXXX.ap2`
Comments on Reveal

- Helps track down the loops taking time.
- A new product, so there will likely be improvements.
- Seems nicer the loopmark listings alone.
- See S-2376-60 Chapter 6 for more details.
References

- CrayPat man pages
  - man intro_craypat
  - man pat_build
  - man pat_report
  - man hwpc
- Apprentice2 man pages
  - man app2
- PAPI man pages
  - man papi_avail
- Using Cray® Performance Measurement and Analysis Tools (S-2376-60)