Cray Performance Measurement and Analysis Tools

Heidi Poxon
Manager & Technical Lead, Performance Tools
Cray Inc.
Day 1 – Introduction to the Cray Performance Tools

- Cray performance tools overview
- Steps to using the tools
- Performance measurement on the Cray XE system
- Using HW performance counters
- Profiling applications
- Visualization of performance data through pat_report
- Visualization of performance data through Cray Apprentice2
- Building, launching and running on raven
Tools Overview
Design Goals

- **Assist** the user with application performance analysis and optimization
  - Help user identify important and meaningful information from potentially massive data sets
  - Help user identify problem areas instead of just reporting data
  - Bring optimization knowledge to a wider set of users

- **Focus on** ease of use and intuitive user interfaces
  - Automatic program instrumentation
  - Automatic analysis

- **Target scalability** issues in all areas of tool development
  - Data management
    - Storage, movement, presentation
Strengths

Provide a complete solution from instrumentation to measurement to analysis to visualization of data

- Performance measurement and analysis on large systems
  - Automatic Profiling Analysis
  - Load Imbalance
  - HW counter derived metrics
  - Predefined trace groups provide performance statistics for libraries called by program (blas, lapack, pgas runtime, netcdf, hdf5, etc.)
  - Observations of inefficient performance
  - Data collection and presentation filtering
  - Data correlates to user source (line number info, etc.)
  - Support MPI, SHMEM, OpenMP, UPC, CAF
  - Access to network counters
  - Minimal program perturbation
Strengths (2)

- **Usability on large systems**
  - Client / server
  - Scalable data format
  - Intuitive visualization of performance data

- **Supports “recipe” for porting MPI programs to many-core or hybrid systems**

- **Integrates with other Cray PE software for more tightly coupled development environment**
The Cray Performance Analysis Framework

- Supports traditional post-mortem performance analysis
  - Automatic identification of performance problems
    - Indication of causes of problems
    - Suggestions of modifications for performance improvement

- **pat_build**: provides automatic instrumentation
- **CrayPat run-time library** collects measurements (transparent to the user)
- **pat_report** performs analysis and generates text reports
- **pat_help**: online help utility
- **Cray Apprentice2**: graphical visualization tool
Separately licensed product from Cray
- Uses FLEXlm licensing (checked during program instrumentation)

Accessed on your system via software modulefiles
- `% module load perftools`

Current release: perftools/5.2.3

Default and past versions available on a system (up to site)
- `% module avail perf`

----------------------------- /opt/cray/modulefiles -----------------------------
perftools/5.1.2   perftools/5.2.2   perftools/5.3.0.8241
perftools/5.1.3   perftools/5.2.3(default) perftools/5.3.0.8250
New .ap2 Format + Client/Server Model

- Reduced pat_report processing and report generation times
- Reduced app2 data load times
- Graphical presentation handled locally (not passed through ssh connection)
- Better tool responsiveness
- Minimizes data loaded into memory at any given time
- Reduced server footprint on Cray XT/XE service node
- Larger data files handled (1.5TB .xf -> 800GB .ap2)
Scalable Data Format Reduced Processing Times

- **CPMD**
  - MPI, instrumented with `pat_build -u, HWPC=1`
  - 960 cores

<table>
<thead>
<tr>
<th></th>
<th>Perftools 5.1.3</th>
<th>Perftools 5.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>.xf -&gt; .ap2</td>
<td>88.5 seconds</td>
<td>22.9 seconds</td>
</tr>
<tr>
<td>ap2 -&gt; report</td>
<td>1512.27 seconds</td>
<td>49.6 seconds</td>
</tr>
</tbody>
</table>

- **VASP**
  - MPI, instrumented with `pat_build -gmpi -u, HWPC=3`
  - 768 cores

<table>
<thead>
<tr>
<th></th>
<th>Perftools 5.1.3</th>
<th>Perftools 5.2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>.xf -&gt; .ap2</td>
<td>45.2 seconds</td>
<td>15.9 seconds</td>
</tr>
<tr>
<td>ap2 -&gt; report</td>
<td>796.9 seconds</td>
<td>28.0 seconds</td>
</tr>
</tbody>
</table>
Log into Cray XT/XE login node

```bash
% ssh -Y kaibab
```

Launch Cray Apprentice2 on Cray XT/XE login node

```bash
% app2 /lus/scratch/mydir/my_program.ap2
```

- User interface displayed on desktop via ssh X11 forwarding
- Entire .ap2 file loaded into memory on login node (can be Gbytes of data)
Launch Cray Apprentice2 on desktop, point to data

```
% app2 kaibab:/lus/scratch/mydir/my_program.ap2
```

- User interface displayed on desktop via X Windows-based software
- Minimal subset of data from .ap2 file loaded into memory on login node at any given time
- Only data requested sent from server to client
Multiple Dimensions of Scalability

- **Millions of lines of code**
  - **Automatic profiling analysis**
    - Identifies top time consuming routines
    - Automatically creates instrumentation template customized to your application

- **Lots of processes/threads**
  - **Load imbalance analysis**
    - Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
    - Estimates savings if corresponding section of code were balanced

- **Long running applications**
  - Detection of **outliers**
Steps to Using the Tools
Collecting Performance Data

- **When** performance measurement is triggered
  - **External agent** (asynchronous)
    - Sampling
      - Timer interrupt
      - Hardware counters overflow
  - **Internal agent** (synchronous)
    - Code instrumentation
      - Event based
      - Automatic or manual instrumentation

- **How** performance data is recorded
  - **Profile** ::= Summation of events over time
    - run time summarization (functions, call sites, loops, …)
  - **Trace file** ::= Sequence of events over time
Application Instrumentation with pat_build

- **pat_build** is a stand-alone utility that automatically instruments the application for performance collection.

- Requires **no** source code or makefile **modification**
  - Automatic instrumentation at group (function) level
    - Groups: mpi, io, heap, math SW, ...

- Performs link-time instrumentation
  - Requires **object files**
  - Instruments optimized code
  - Generates stand-alone instrumented program
  - Preserves original binary
supports two categories of experiments
- asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows
  - event-based experiments (tracing) which count some events such as the number of times a specific system call is executed

while tracing provides most useful information, it can be very heavy if the application runs on a large number of cores for a long period of time

sampling can be useful as a starting point, to provide a first overview of the work distribution
Large programs
- Scaling issues more dominant
- Use automatic profiling analysis to quickly identify top time consuming routines
- Use loop statistics to quickly identify top time consuming loops

Small (test) or short running programs
- Scaling issues not significant
- Can skip first sampling experiment and directly generate profile
- For example: `% pat_build -u -g mpi my_program`
Sampling is useful to determine where the program spends most of its time (functions and lines).

The environment variable `PAT_RT_EXPERIMENT` allows the specification of the type of experiment prior to execution:

- **samp_pc_time** (default)
  - Samples the PC at intervals of 10,000 microseconds
  - Measures user CPU and system CPU time
  - Returns total program time and absolute and relative times each program counter was recorded
  - Optionally record the values of hardware counters specified with `PAT_RT_HWPC`.

- **samp_pc_ovfl**
  - Samples the PC at a given overflow of a HW counter
  - Does not allow collection of hardware counters.

- **samp_cs_time**
  - Sample the call stack at a given time interval.
Automatic profiling analysis (APA)

- Provides simple procedure to instrument and collect performance data for novice users
- Identifies top time consuming routines
- Automatically creates instrumentation template customized to application for future in-depth measurement and analysis
Steps to Collecting Performance Data

- Access performance tools software
  
  \texttt{% module load perftools}

- Build application keeping .o files (CCE: -h keepfiles)

  \texttt{% make clean}
  \texttt{% make}

- Instrument application for automatic profiling analysis
  - You should get an instrumented program a.out+pat

  \texttt{% pat_build -O apa a.out}

- Run application to get top time consuming routines
  - You should get a performance file ("<sdatafile>.xf") or multiple files in a directory <sdatadir>

  \texttt{% aprun ... a.out+pat} (or \texttt{qsub <pat script>})
Steps to Collecting Performance Data (2)

- Generate report and .apa instrumentation file
  
  `pat_report -o my_sampling_report [<sdatafile>.xf | <sdatadir>]`

- Inspect .apa file and sampling report

- Verify if additional instrumentation is needed
APA File Example

You can edit this file, if desired, and use it to reinstrument the program for tracing like this:

```
pat_build -O standard.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.512.quad.cores.seal.090405.1154 mpi.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.apa
```

These suggested trace options are based on data from:

```
/home/users/malice/pat/Runs/Runs.seal.pat5001.2009Apr04/.pat.quad/homme/standard.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2-Oapa.512.quad.cores.seal.090405.1154 mpi.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.cdb
```

HWPC group to collect by default.

```
-Drtenv=PAT_RT_HWPC=1 # Summary with TLB metrics.
```

Libraries to trace.

```
-g mpi
```

User-defined functions to trace, sorted by % of samples.

```
The way these functions are filtered can be controlled with pat_report options (values used for this file are shown):

- s apa_max_count=200 No more than 200 functions are listed.
- s apa_min_size=800 Commented out if text size < 800 bytes.
- s apa_min_pcnt=1 Commented out if it had < 1% of samples.
- s apa_max_cum_pcnt=90 Commented out after cumulative 90%.

Local functions are listed for completeness, but cannot be traced.

```
-w # Enable tracing of user-defined functions.
```

31.29% 38517 bytes

```
-T prim_advance_mod_preq_advance_exp
```

15.07% 14158 bytes

```
-T prim_si_mod_prim_diffusion
```

9.76% 5474 bytes

```
-T derivative_mod_gradient_str_nonstag
```

...

2.95% 3067 bytes

```
-T forcing_mod_apply_forcing
```

2.93% 118585 bytes

```
-T column_model_mod_applycolumnmodel
```

Functions below this point account for less than 10% of samples.

```
0.66% 4575 bytes
```

```
-T bndry_mod_bndry_exchangev_thsave_time
```

```
0.10% 46797 bytes
```

```
-T baroclinic_inst_mod_binst_init_state
```

```
0.04% 62214 bytes
```

```
-T prim_state_mod_prim_printstate
```

```
```

0.00% 118 bytes

```
-T time_mod_timelevel_update
```

New instrumented program.

```
-pregx.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x+apa
```

Original program.

```
./AUTO/cray/css_pe_tools/malice/craypat/build/pat/2009Apr03/2.1.56HD/amd64/homme/pgi-pat-5.0.0.2/homme/2005Dec08/build.Linux/pregx.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x # Original program.
```
Generating Profile from APA

- Instrument application for further analysis (a.out+apa)

  `% pat_build -O <apafilename>.apa`

- Run application

  `% aprun ... a.out+apa (or qsub <apa script>)`

- Generate text report and visualization file (.ap2)

  `% pat_report -o my_text_report.txt [<datafile>.xf | <datadir>]`

- View report in text and/or with Cray Apprentice²

  `% app2 <datafile>.ap2`
Where to Run Instrumented Application

- **MUST run on Lustre** ( /work/… , /lus/…, /scratch/…, etc.)

- Number of files used to store raw data
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – $n$ processes
  - Ability to customize with `PAT_RT_EXPFILE_MAX`
## Files Generated and the Naming Convention

<table>
<thead>
<tr>
<th>File Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.out+pat</td>
<td>Program <em>instrumented</em> for data collection</td>
</tr>
<tr>
<td>a.out...s.xf</td>
<td><em>Raw data</em> for <em>sampling</em> experiment, available after application execution</td>
</tr>
<tr>
<td>a.out...t.xf</td>
<td><em>Raw data</em> for <em>trace</em> (summarized or full) experiment, available after application execution</td>
</tr>
<tr>
<td>a.out...st.ap2</td>
<td><em>Processed</em> data, generated by <em>pat_report</em>, contains application symbol information</td>
</tr>
<tr>
<td>a.out...s.apa</td>
<td>Automatic profiling <em>pnanalysis</em> <em>template</em>, generated by <em>pat_report</em> (based on <em>pat_build</em> –O apa experiment)</td>
</tr>
<tr>
<td>a.out+apa</td>
<td>Program instrumented using .apa file</td>
</tr>
<tr>
<td>MPICH_RANK_ORDER.Custom</td>
<td>Rank <em>reorder</em> file generated by <em>pat_report</em> from automatic grid detection and reorder suggestions</td>
</tr>
</tbody>
</table>
Why Should I generate an “.ap2” file?

- The “.ap2” file is a self contained compressed performance file
  - Normally it is about 5 times smaller than the “.xf” file
  - Contains the information needed from the application binary
    - Can be reused, even if the application binary is no longer available or if it was rebuilt
- It is the only input format accepted by Cray Apprentice²
CrayPat Runtime Options

- Runtime controlled through PAT_RT Xxx environment variables

- See intro_craypat(1) man page

- Examples of control
  - Enable full trace
  - Change number of data files created
  - Enable collection of HW counters
  - Enable collection of network counters
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)
Example Runtime Environment Variables

- Optional timeline view of program available
  - export `PAT_RT_SUMMARY=0`
  - View trace file with Cray Apprentice\(^2\)

- Number of files used to store raw data:
  - 1 file created for program with 1 – 256 processes
  - \(\sqrt{n}\) files created for program with 257 – \(n\) processes
  - Ability to customize with `PAT_RT_EXPFILE_MAX`

- Request hardware performance counter information:
  - export `PAT_RT_HWPC=<HWPC Group>`
  - Can specify events or predefined groups
View Data with pat_report
pat_report

- Performs data conversion
  - Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Formats data for input into Cray Apprentice²
CrayPat/X:  Version 5.2.3.8078 Revision 8078 (xf 8063) 08/25/11 ...

Number of PEs (MPI ranks): 16
Numbers of PEs per Node: 16
Numbers of Threads per PE: 1
Number of Cores per Socket: 12
Execution start time: Thu Aug 25 14:16:51 2011
System type and speed: x86_64 2000 MHz
Current path to data file:
  /lus/scratch/heidi/ted_swim/mpi-openmp/run/swim+pat+27472-34t.ap2
Notes for table 1:
...
## Sampling Output (Table 1)

### Notes for table 1:
...

### Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb. Samp</th>
<th>Imb. Samp %</th>
<th>Group Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>775</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>94.2%</td>
<td>730</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
</tbody>
</table>

| 43.4% | 336  | 8.75 | 2.6% | PE='HIDE' |
| 16.1% | 125  | 6.28 | 4.9% | half |
| 8.0%  | 62   | 6.25 | 9.5% | full |
| 6.8%  | 53   | 1.88 | 3.5% | artv |
| 4.9%  | 38   | 1.34 | 3.6% | bnd |
| 3.6%  | 28   | 2.00 | 6.9% | currenf |
| 2.2%  | 17   | 1.50 | 8.6% | bndsf |
| 1.7%  | 13   | 1.97 | 13.5% | model |
| 1.4%  | 11   | 1.53 | 12.2% | cfl |
| 1.3%  | 10   | 0.75 | 7.0% | currenh |
| 1.0%  | 8    | 5.28 | 41.9% | bndbo |
| 1.0%  | 8    | 8.28 | 53.4% | bndto |
| 5.4%  | 42   | -- | -- | MPI |

| 1.9%  | 15   | 4.62 | 23.9% | mpi_sendrecv |
| 1.8%  | 14   | 16.53 | 55.0% | mpi_bcast |
| 1.7%  | 13   | 5.66 | 30.7% | mpi_barrier |
# pat_report: Flat Profile

<table>
<thead>
<tr>
<th>Time %</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>104.593634</td>
<td>--</td>
<td>--</td>
<td>22649</td>
<td>Total</td>
</tr>
<tr>
<td>71.0%</td>
<td>74.230520</td>
<td>--</td>
<td>--</td>
<td>10473</td>
<td>MPI</td>
</tr>
<tr>
<td>69.7%</td>
<td>72.905208</td>
<td>0.508369</td>
<td>0.7%</td>
<td>125</td>
<td>mpi_allreduce_</td>
</tr>
<tr>
<td>1.0%</td>
<td>1.050931</td>
<td>0.030042</td>
<td>2.8%</td>
<td>94</td>
<td>mpi_alltoall_</td>
</tr>
<tr>
<td>25.3%</td>
<td>26.514029</td>
<td>--</td>
<td>--</td>
<td>73</td>
<td>USER</td>
</tr>
<tr>
<td>16.7%</td>
<td>17.461110</td>
<td>0.329532</td>
<td>1.9%</td>
<td>23</td>
<td>selfgravity_</td>
</tr>
<tr>
<td>7.7%</td>
<td>8.078474</td>
<td>0.114913</td>
<td>1.4%</td>
<td>48</td>
<td>ffte4_</td>
</tr>
<tr>
<td>2.5%</td>
<td>2.659429</td>
<td>--</td>
<td>--</td>
<td>435</td>
<td>MPI_SYNC</td>
</tr>
<tr>
<td>2.1%</td>
<td>2.207467</td>
<td>0.768347</td>
<td>26.2%</td>
<td>172</td>
<td>mpi_barrier_(sync)</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.188998</td>
<td>--</td>
<td>--</td>
<td>11608</td>
<td>HEAP</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.166707</td>
<td>0.142473</td>
<td>11.1%</td>
<td>5235</td>
<td>free</td>
</tr>
</tbody>
</table>
Table 4: MPI Message Stats by Caller

<table>
<thead>
<tr>
<th>MPI Msg Bytes</th>
<th>MPI Msg Count</th>
<th>MsgSz &lt;16B</th>
<th>4KB&lt;=</th>
<th>Function</th>
<th>PE[mmm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>15138076.0</td>
<td>4099.4</td>
<td>411.6</td>
<td>3687.8</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>15138028.0</td>
<td>4093.4</td>
<td>405.6</td>
<td>3687.8</td>
<td>MPI_ISEND</td>
<td></td>
</tr>
<tr>
<td>8080500.0</td>
<td>2062.5</td>
<td>93.8</td>
<td>1968.8</td>
<td>calc2</td>
<td>MAIN_</td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
<td></td>
</tr>
<tr>
<td>8208000.0</td>
<td>2000.0</td>
<td>--</td>
<td>2000.0</td>
<td>pe.9</td>
<td></td>
</tr>
<tr>
<td>6160000.0</td>
<td>2000.0</td>
<td>500.0</td>
<td>1500.0</td>
<td>pe.15</td>
<td></td>
</tr>
<tr>
<td>6285250.0</td>
<td>1656.2</td>
<td>125.0</td>
<td>1531.2</td>
<td>calc1</td>
<td>MAIN_</td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
<td></td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.3</td>
<td></td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.5</td>
<td></td>
</tr>
</tbody>
</table>
Using HW Performance Counters
AMD Opteron Hardware Performance Counters

- **Four** 48-bit performance counters.
  - Each counter can monitor a single event
    - Count specific processor events
      - the processor increments the counter when it detects an occurrence of the event
      - (e.g., cache misses)
    - Duration of events
      - the processor counts the number of processor clocks it takes to complete an event
      - (e.g., the number of clocks it takes to return data from memory after a cache miss)

- **Time Stamp Counters (TSC)**
  - Cycles (user time)
PAPI Predefined Events

- Common set of events deemed relevant and useful for application performance tuning
  - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
  - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node

- PAPI also provides access to native events
  - The “papi_native_avail” utility lists all AMD native events available on the system – execute on compute node

- PAPI uses perf_events Linux subsystem

- Information on PAPI and AMD native events
  - pat_help counters
  - man intro_papi
Hardware Counters Selection

- HW counter collection enabled with PAT_RT_HWPC environment variable

- PAT_RT_HWPC <set number> | <event list>
  - A set number can be used to select a group of predefined hardware counters events (recommended)
    - CrayPat provides 22 groups on the Cray XT/XE systems
    - See `pat_help(1)` or the `hwpc(5)` man page for a list of groups
  - Alternatively a list of hardware performance counter event names can be used
  - Hardware counter events are not collected by default
HW Counter Information Available in Reports

- Raw data
- Derived metrics
- Desirable thresholds
## Example: HW counter data and Derived Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_TLB_DM</td>
<td>Data translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>Level 1 data cache accesses</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>Floating point operations</td>
</tr>
<tr>
<td>DC_MISS</td>
<td>Data Cache Miss</td>
</tr>
<tr>
<td>User_Cycles</td>
<td>Virtual Cycles</td>
</tr>
</tbody>
</table>

---

### USER

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>98.3%</td>
</tr>
<tr>
<td>Time</td>
<td>4.434402 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>0.001M/sec, 4500.0 calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>14.820M/sec, 65712197 misses</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.902M/sec, 3998928 misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>333.331M/sec, 1477996162 refs</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>445.571M/sec, 1975672594 ops</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>4.434 secs, 11971868993 cycles, 100.0%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000985 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.1%</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>445.571M/sec, 1975672594 ops, 4.1%peak(DP)</td>
</tr>
<tr>
<td>HW FP Ops / WCT</td>
<td>445.533M/sec</td>
</tr>
<tr>
<td>Computational intensity</td>
<td>0.17 ops/cycle, 1.34 ops/ref</td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
<td>1782.28M/sec</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>369.60 refs/miss, 0.722 avg uses</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>95.6% hits, 4.4% misses</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>22.49 refs/miss, 2.811 avg hits</td>
</tr>
</tbody>
</table>

**Note:**
- PAT_RT_HWPC=1
- Flat profile data
- Raw counts
- Derived metrics
Predefined Interlagos HW Counter Groups

See pat_help -> counters -> amd_fam15h -> groups

0: Summary with instructions metrics
1: Summary with TLB metrics
2: L1 and L2 Metrics
3: Bandwidth information
4: <Unused>
5: Floating operations dispatched
6: Cycles stalled, resources idle
7: Cycles stalled, resources full
8: Instructions and branches
9: Instruction cache
10: Cache Hierarchy (unsupported for IL)
11: Floating point operations dispatched
12: Dual pipe floating point operations dispatched
13: Floating point operations SP
14: Floating point operations DP
L3 (socket and core level) (unsupported)
19: Prefetchs
20: FP, D1, TLB, MIPS  <<-new for Interlagos
21: FP, D1, TLB, Stalls
22: D1, TLB, MemBW
New HW counter groups for Interlagos (6 counters)

- **Group 20: FP, D1, TLB, MIPS**
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_TOT_INS

- **Group 21: FP, D1, TLB, Stalls**
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_RES_STL

October 12-13 2011
Hardware performance counter events:

- **PAPI_L1_DCM**  Level 1 data cache misses
- **CYCLES_RTC**  User Cycles (approx, from rtc)
- **PAPI_L1_DCA**  Level 1 data cache accesses
- **PAPI_TLB_DM**  Data translation lookaside buffer misses
- **PAPI_FP_OPS**  Floating point operations

Estimated minimum overhead per call of a traced function, which was subtracted from the data shown in this report (for raw data, use the option:  `-s overhead=include`):

- **PAPI_L1_DCM**  8.040  misses
- **PAPI_TLB_DM**  0.005  misses
- **PAPI_L1_DCA**  474.080  refs
- **PAPI_FP_OPS**  0.000  ops
- **CYCLES_RTC**  1863.680  cycles
- **Time**  0.693  microseconds
### USER

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>98.3%</td>
</tr>
<tr>
<td>Time</td>
<td>4.436808 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>0.001M/sec 4500.0 calls</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS:</td>
<td></td>
</tr>
<tr>
<td>L2_MODIFIED:L2_OWNED:</td>
<td>9.821M/sec 43567825 fills</td>
</tr>
<tr>
<td>L2_EXCLUSIVE:L2_SHARED</td>
<td></td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_SYSTEM:</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>24.743M/sec 109771658 fills</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>14.824M/sec 65765949 misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>332.960M/sec 1477145402 refs</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>4.436 secs 11978286133 cycles 100.0%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000986 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.1%</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>95.5% hits 4.5% misses</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>22.46 refs/miss 2.808 avg hits</td>
</tr>
<tr>
<td>D1 cache utilization (refills)</td>
<td>9.63 refs/refill 1.204 avg uses</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>28.4% hits 71.6% misses</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
<td>96.8% hits 3.2% misses</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>31.38 refs/miss 3.922 avg hits</td>
</tr>
<tr>
<td>System to D1 refill</td>
<td>24.743M/sec 109771658 lines</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
<td>1510.217MB/sec 7025386144 bytes</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>599.398MB/sec 2788340816 bytes</td>
</tr>
</tbody>
</table>

---

**PAT_RT_HWPC=2 (L1 and L2 Metrics)**

---
Example: Observations and Suggestions

**D1 + D2 cache utilization:** 39.8% of total execution time was spent in 4 functions with combined D1 and D2 cache hit ratios below the desirable minimum of 97.0%. Cache utilization might be improved by modifying the alignment or stride of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>D1_D2_cache_hit_ratio</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>56.8%</td>
<td>12.0%</td>
<td>calc3_</td>
</tr>
<tr>
<td>77.9%</td>
<td>6.4%</td>
<td>calc2_</td>
</tr>
<tr>
<td>95.7%</td>
<td>1.4%</td>
<td>calc1_</td>
</tr>
<tr>
<td>96.3%</td>
<td>20.0%</td>
<td>calc3_.LOOP@li.80</td>
</tr>
</tbody>
</table>

**TLB utilization:** 19.6% of total execution time was spent in 3 functions with fewer than the desirable minimum of 512 data references per TLB miss. TLB utilization might be improved by modifying the alignment or stride of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>LS_per_TLB_DM</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.56</td>
<td>12.0%</td>
<td>calc3_</td>
</tr>
<tr>
<td>5.32</td>
<td>6.3%</td>
<td>calc2_</td>
</tr>
</tbody>
</table>
Profile Visualization with Cray Apprentice2
• Call graph profile
• Communication statistics
• Time-line view
  • Communication
  • I/O
• Activity view
• Pair-wise communication statistics
• Text reports
• Source code mapping

Cray Apprentice\(^2\) helps identify:
• Load imbalance
• Excessive communication
• Network contention
• Excessive serialization
• I/O Problems
Accessing Remote File from app2 Client
Statistics Overview

Switch Overview display
Function Profile

Cray Inc.

October 12-13 2011
<table>
<thead>
<tr>
<th>Time</th>
<th>Percent</th>
<th>MB/s</th>
<th>Callsites</th>
<th>Inclusion %</th>
<th>Potential</th>
<th>Function</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>124.175511</td>
<td>63.29</td>
<td>576</td>
<td>1</td>
<td>6.83</td>
<td>0.15</td>
<td>sweep_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>40.211774</td>
<td>20.50</td>
<td>116600</td>
<td>1</td>
<td>23.40</td>
<td>0.25</td>
<td>mpi_recv_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>16.319627</td>
<td>8.32</td>
<td>48</td>
<td>1</td>
<td>48.26</td>
<td>0.30</td>
<td>env</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>6.173296</td>
<td>3.15</td>
<td>1536</td>
<td>3</td>
<td>50.00</td>
<td>0.12</td>
<td>mpi_bcast_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>2.705306</td>
<td>1.41</td>
<td>116600</td>
<td>1</td>
<td>17.30</td>
<td>0.01</td>
<td>mpi_send_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>2.250029</td>
<td>1.15</td>
<td>576</td>
<td>1</td>
<td>2.82</td>
<td>0.00</td>
<td>source_</td>
<td>/usr/local/apps/sweep3d/source.f</td>
</tr>
<tr>
<td>1.004220</td>
<td>1.01</td>
<td>144</td>
<td>1</td>
<td>2.59</td>
<td>0.00</td>
<td>mpi_bcast_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.867398</td>
<td>0.44</td>
<td>192</td>
<td>2</td>
<td>2.47</td>
<td>0.00</td>
<td>mpi_bcast_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.419251</td>
<td>0.21</td>
<td>576</td>
<td>1</td>
<td>2.90</td>
<td>0.00</td>
<td>fun_err_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.382130</td>
<td>0.18</td>
<td>116600</td>
<td>2</td>
<td>10.38</td>
<td>0.00</td>
<td>mpi_send_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.257772</td>
<td>0.12</td>
<td>309</td>
<td>1</td>
<td>95.76</td>
<td>0.07</td>
<td>fortran</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.186227</td>
<td>0.09</td>
<td>116600</td>
<td>2</td>
<td>17.30</td>
<td>0.00</td>
<td>mpi_lhs_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.057302</td>
<td>0.03</td>
<td>48</td>
<td>1</td>
<td>4.56</td>
<td>0.00</td>
<td>init_a</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.039407</td>
<td>0.03</td>
<td>48</td>
<td>1</td>
<td>4.39</td>
<td>0.00</td>
<td>init_b</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.014317</td>
<td>0.02</td>
<td>48</td>
<td>1</td>
<td>23.84</td>
<td>0.00</td>
<td>init_c</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.023940</td>
<td>0.01</td>
<td>6</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>init_d</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.016802</td>
<td>0.01</td>
<td>68</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>init_e</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.033104</td>
<td>0.00</td>
<td>4892</td>
<td>2</td>
<td>20.14</td>
<td>0.00</td>
<td>init_f</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.002867</td>
<td>0.00</td>
<td>576</td>
<td>1</td>
<td>18.79</td>
<td>0.00</td>
<td>global_real_ax_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.002162</td>
<td>0.00</td>
<td>48</td>
<td>1</td>
<td>68.95</td>
<td>0.00</td>
<td>main_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.001286</td>
<td>0.00</td>
<td>576</td>
<td>1</td>
<td>38.10</td>
<td>0.00</td>
<td>global_int_sum_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.001333</td>
<td>0.00</td>
<td>48</td>
<td>1</td>
<td>10.23</td>
<td>0.00</td>
<td>inner_auto_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.000822</td>
<td>0.00</td>
<td>48</td>
<td>1</td>
<td>97.74</td>
<td>0.00</td>
<td>task_init_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.003728</td>
<td>0.00</td>
<td>384</td>
<td>2</td>
<td>27.87</td>
<td>0.00</td>
<td>global_real_sum_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.00062</td>
<td>0.00</td>
<td>2</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>fopen_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
<tr>
<td>0.000468</td>
<td>0.00</td>
<td>48</td>
<td>1</td>
<td>7.54</td>
<td>0.00</td>
<td>init_mce_</td>
<td>/usr/local/apps/sweep3d/sweep.f</td>
</tr>
</tbody>
</table>
Load Balance View (Aggregated from Overview)

Min, Avg, and Max Values

-1, +1 Std Dev marks
Load balance overview:
Height ↔ Max time
Middle bar ↔ Average time
Lower bar ↔ Min time

Yellow represents imbalance time

Width ↔ inclusive time
Height ↔ exclusive time

Filtered nodes or sub tree

DUH Button: Provides hints for performance tuning

Zoom

Function List
Call Tree View – Function List

Function List off

Right mouse click: Node menu e.g., hide/unhide children

Right mouse click: View menu: e.g., Filter

Sort options % Time, Time, Imbalance % Imbalance time

Function List off

October 12-13 2011 Cray Inc.
Apprentice\textsuperscript{2} Call Tree View of Sampled Data
Discrete Unit of Help (DUH Button)
Load Balance View (from Call Tree)

Min, Avg, and Max Values

-1, +1 Std Dev marks
Complimentary performance data available in one place

Drop down menu provides quick access to most common reports

Ability to easily generate different views of performance data

Provides mechanism for more in depth explanation of data presented
Example of pat_report Tables in Cray Apprentice2

New text table icon

Right click for table generation options
Memory Hierarchy Thresholds

- **TLB utilization: < 90.0%**
  - Measures how well the memory hierarchy is being utilized with regards to TLB
  - This metric depends on the computation being single precision or double precision
    - A page has 4 Kbytes. So, one page fits 512 double precision words or 1024 single precision words
  - TLB utilization < 1 indicates that not all entries on the page are being utilized between two TLB misses

- **Cache utilization: < 1 (D1 or D1+D2)**
  - A cache line has 64 bytes (8 double precision words or 16 single precision words)
  - Cache utilization < 1 indicates that not all entries on the cache line are being utilized between two cache misses

- **D1 cache hit (or miss) ratios: < 90% ( > 10%)**

- **D1 + D2 cache hit (or miss) ratios: < 92% ( > 8%)**
  - D1 and D2 caches on the Opteron are complementary
  - This metric provides a view of the Total Cache hit (miss) ratio
Running on Raven
Cray XE6

Runs CLE 4.0.UP00

20 Compute nodes
- 2 MC12 chips (24 cores per node)
- 1.9 GHz
- 32 GB memory

Lustre parallel filesystem

Usage information available at:
- https://partners.cray.com/
Purpose of Raven

- Allow users to familiarize themselves with Cray XE6 system
- Aid in port applications to Cray XE6
- Provide platform for the optimization and tuning of Cray XE6 applications

- System is relatively small
- Shared by many users
- Job execution time limits in place
- Limit large jobs (15-20 compute nodes) to 10 minutes or less
- Small jobs (1 or 2 nodes) can run longer (more than an hour is ok)
- xtnodestat shows current system layout
Software Access on Raven

- To see list of modulefiles loaded by default:
  - % module list

- To see list of available modulefiles:
  - % module avail

- To swap a modulefile
  - % module swap PrgEnv-pgi PrgEnv-cray

- To load the performance tools
  - % module load perftools
Cray XE6 systems use compiler driver scripts:
- Automatically reference MPI, PGAS, SHMEM
- Should be used with all programming environments (Cray, PGI, GNU, Intel)
  - `% ftn -h profile_generate -o test1 test1.f`
  - `cc -h profile_generate -o test2 test2.c`

Compilers require you to target a processor
- `xtpe-mc12` modulefile is loaded by default
  - Cray and PGI compilers target MC by default
Running on Raven

- Name for Lustre filesystem is /lus/scratch
  - Create directory here using your login

- Batch scheduler is PBS

- Job queue names available:
  - small walltime=6hrs, nodelimit=4 nodes
  - medium walltime=6hrs, nodelimit=8 nodes
  - large walltime=6hrs, nodelimit=16 nodes
  - long walltime=24hrs, nodelimit=4 nodes

```bash
% qsub -l mppwidth=96 -l mppnppn=24 -q large testrun.pbs
```

- Job size
- # of processes per node (1 – 24)
Sample PBS Job Script

#PBS -l mppwidth=24
#PBS -l mppnppn=4
#PBS -l walltime=00:10:00
#PBS -j oe

cd $PBS_O_WORKDIR
aprun -n 24 -N 4 ./mpi_app
Questions

??