Cray Performance Measurement and Analysis Tools

Cray XE6 Performance Workshop
20-22 November
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
- New features in Cray Apprentice2
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
The Cray Performance Analysis Framework (2)

- **CrayPat**
  - Instrumentation of optimized code
  - No source code modification required
  - Data collection transparent to the user
  - Text-based performance reports
  - Derived metrics
  - Performance analysis

- **Cray Apprentice2**
  - Performance data visualization tool
  - Call tree view
  - Source code mappings
Steps to Using the Tools
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
Application Instrumentation with pat_build (2)

- **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**
Program Instrumentation Tips

● **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`
Where to Run Instrumented Application

- **MUST** run on Lustre ( /mnt/snx3/... , /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
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 Apprentice2

- **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
● 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 Apprentice

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
<table>
<thead>
<tr>
<th>File Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.out+pat</td>
<td>Program instrumented for data collection</td>
</tr>
<tr>
<td>a.out...s.xf</td>
<td>Raw data for sampling experiment, available after application execution</td>
</tr>
<tr>
<td>a.out...t.xf</td>
<td>Raw data for trace (summarized or full) experiment, available after application execution</td>
</tr>
<tr>
<td>a.out...st.ap2</td>
<td>Processed data, generated by pat_report, contains application symbol information</td>
</tr>
<tr>
<td>a.out...s.apa</td>
<td>Automatic profiling panalysis template, generated by pat_report (based on pat_build –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 reorder file generated by pat_report from automatic grid detection an reorder suggestions</td>
</tr>
</tbody>
</table>
Program Instrumentation - Automatic Profiling Analysis

- 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**
  
  ```bash
  module load perftools
  ```

- **Build application keeping .o files (CCE: -h keepfiles)**
  
  ```bash
  make clean
  make
  ```

- **Instrument application for automatic profiling analysis**
  - You should get an instrumented program `a.out+pat`
  
  ```bash
  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>`
  
  ```bash
  aprun ... a.out+pat  (or 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.minecraft.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.apa
#
# These suggested trace options are based on data from:
#
# /home/users/malice/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.minecraft.pat_rt_exp=default.pat_rt_hwpc=none.14999.xf.xf.cd
#
# 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_pct=1 Commented out if it had < 1% of samples.
# -s apa_max_cum_pct=90 Commented out after cumulative 90%.
#
# Local functions are listed for completeness, but cannot be traced.
-w # Enable tracing of user-defined functions.
# Note: -w should NOT be specified as an additional option.

# 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% 4679 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.

/./AUTO/cray/css.pe_tools/malice/craypat/build/pat2009Apr03/2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x+apa
# Original program.
Generating Profile from APA

- Instrument application for further analysis (a.out+apa)
  
  \[
  \texttt{% pat\_build\ -O <apafilename>.apa}
  \]

- Run application
  
  \[
  \texttt{% aprun \ldots a.out+apa (or qsub <apa\ script>)}
  \]

- Generate text report and visualization file (.ap2)
  
  \[
  \texttt{% pat\_report\ -o my\_text\_report.txt [<datafile>.xf | <datadir>]}
  \]

- View report in text and/or with Cray Apprentice²
  
  \[
  \texttt{% app2 <datafile>.ap2}
  \]
HW Performance Counters
Hardware Performance Counters - MC

- **AMD Family 10H Opteron Hardware Performance Counters**
  - Each core has **4** 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)
Hardware Performance Counters - IL

● **AMD Family 15H Opteron Hardware Performance Counters**
  ● Each node has 4 48-bit NorthBridge counters
  
  ● Each core has 6 48-bit performance counters
    ● Not all events can be counted on all counters
    ● Supports multi-events
      ● events have a maximum count per clock that exceeds one event per clock
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 (points to PAPI documentation: http://icl.cs.utk.edu/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 23 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
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)
Predefined Interlagos HW Counter Groups (cont’d)

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
Example of MC -> IL Counter Name Changes

Check spelling via papi_native_avail

PAPI_DP_OPS

● **AMD Family 10H:**
  - RETIRED_SSE_OPERATIONS:DOUBLE_ADD_SUB_OPS:DOUBLE_MUL_OPS:DOUBLE_DIV_OPS

● **AMD Family 15H:**
Example: HW counter data and Derived Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</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>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>USER</td>
<td></td>
</tr>
<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>

PAT_RT_HWPC=1
Flat profile data
Raw counts
Derived metrics
Profile Visualization with pat_report and Cray Apprentice2
Examples of Recent Scaling Efforts
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>
Old Client/Server (Cray Performance Tools 5.0.0)

- Log into Cray XT/XE login node
  \% ssh -Y kaibab

- Launch Cray Apprentice2 on Cray XT/XE login node
  \% 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)
New Client/Server (Cray Performance Tools 5.2.0)

- 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
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:
...
Notes for table 1:

Table option:
-0 profile

Options implied by table option:
-d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE

Other options:
-7

Options for related tables:
-0 profile_pe.th
-0 profile+src
-0 callers
-0 calltree
-0 profile_th_pe
-0 load_balance
-0 callers+src
-0 calltree+src

The Total value for Time, Calls is the sum for the Group values.
The Group value for Time, Calls is the sum for the Function values.
The Function value for Time, Calls is the \textit{avg} for the PE values.
(To specify different aggregations, see: \texttt{pat_help report options s1})

This table shows only lines with Time\% > 0.

Percentages at each level are of the Total for the program.
(For percentages relative to next level up, specify:
- \texttt{s percent=r[elative]})
pat_report: Additional Information

Instrumented with:
   pat_build -gmpi -u himenoBMTxpr.x

Program invocation:
   ../bin/himenoBMTxpr+pat.x

Exit Status:  0 for 256 PEs

CPU    Family: 15h  Model: 01h  Stepping: 2
Core Performance Boost:  Configured for   0 PEs
                          Capable    for 256 PEs

Memory pagesize:  4096

Accelerator Model: Nvidia X2090 Memory:  6.00 GB Frequency: 1.15 GHz

Programming environment:  CRAY

Runtime environment variables:
   OMP_NUM_THREADS=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 PE='HIDE'</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>
<tr>
<td>43.4%</td>
<td>336</td>
<td>8.75</td>
<td>2.6%</td>
<td>mlwxyz</td>
</tr>
<tr>
<td>16.1%</td>
<td>125</td>
<td>6.28</td>
<td>4.9%</td>
<td>half</td>
</tr>
<tr>
<td>8.0%</td>
<td>62</td>
<td>6.25</td>
<td>9.5%</td>
<td>full</td>
</tr>
<tr>
<td>6.8%</td>
<td>53</td>
<td>1.88</td>
<td>3.5%</td>
<td>artv</td>
</tr>
<tr>
<td>4.9%</td>
<td>38</td>
<td>1.34</td>
<td>3.6%</td>
<td>bnd_</td>
</tr>
<tr>
<td>3.6%</td>
<td>28</td>
<td>2.00</td>
<td>6.9%</td>
<td>currenf_</td>
</tr>
<tr>
<td>2.2%</td>
<td>17</td>
<td>1.50</td>
<td>8.6%</td>
<td>bndsf_</td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>1.97</td>
<td>13.5%</td>
<td>model_</td>
</tr>
<tr>
<td>1.4%</td>
<td>11</td>
<td>1.53</td>
<td>12.2%</td>
<td>cfl_</td>
</tr>
<tr>
<td>1.3%</td>
<td>10</td>
<td>0.75</td>
<td>7.0%</td>
<td>currenh</td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>5.28</td>
<td>41.9%</td>
<td>bndbo_</td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>8.28</td>
<td>53.4%</td>
<td>bndto_</td>
</tr>
<tr>
<td>5.4%</td>
<td>42</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>1.9%</td>
<td>15</td>
<td>4.62</td>
<td>23.9%</td>
<td>mpi_sendrecv</td>
</tr>
<tr>
<td>1.8%</td>
<td>14</td>
<td>16.53</td>
<td>55.0%</td>
<td>mpi_bcast</td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>5.66</td>
<td>30.7%</td>
<td>mpi_barrier</td>
</tr>
</tbody>
</table>
## Table 1: Profile by Function Group and Function

<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>104.59</td>
<td>0.00</td>
<td>0.00</td>
<td>22649</td>
<td>Total</td>
</tr>
<tr>
<td>71.0%</td>
<td>74.23</td>
<td>0.00</td>
<td>0.00</td>
<td>10473</td>
<td>MPI</td>
</tr>
<tr>
<td>69.7%</td>
<td>72.90</td>
<td>0.50</td>
<td>0.7%</td>
<td>125</td>
<td>mpi_allreduce</td>
</tr>
<tr>
<td>1.0%</td>
<td>1.05</td>
<td>0.03</td>
<td>2.8%</td>
<td>94</td>
<td>mpi_alltoall</td>
</tr>
<tr>
<td>25.3%</td>
<td>26.51</td>
<td>0.33</td>
<td>1.9%</td>
<td>73</td>
<td>USER</td>
</tr>
<tr>
<td>16.7%</td>
<td>17.46</td>
<td>0.33</td>
<td>1.4%</td>
<td>48</td>
<td>fft4_</td>
</tr>
<tr>
<td>7.7%</td>
<td>8.08</td>
<td>0.11</td>
<td>1.4%</td>
<td>435</td>
<td>MPI_SYNC</td>
</tr>
<tr>
<td>2.1%</td>
<td>2.21</td>
<td>0.77</td>
<td>26.2%</td>
<td>172</td>
<td>mpi_barrier(sync)</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.19</td>
<td>0.14</td>
<td>11.1%</td>
<td>11608</td>
<td>HEAP</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.17</td>
<td>0.14</td>
<td>5235</td>
<td>free</td>
<td></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>Caller</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_3</td>
<td></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_3</td>
<td></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>

...
MPI Rank Placement Suggestions
Automatic Communication Grid Detection

- Analyze runtime performance data to identify grids in a program to maximize on-node communication
  - Example: nearest neighbor exchange in 2 dimensions
    - Sweep3d uses a 2-D grid for communication

- Determine whether or not a custom MPI rank order will produce a significant performance benefit

- Grid detection is helpful for programs with significant point-to-point communication

- Doesn’t interfere with MPI collective communication optimizations
Automatic Grid Detection (cont’d)

- Tools produce a custom rank order if it’s beneficial based on grid size, grid order and cost metric

- Summarized findings in report

- Available if MPI functions traced (-g mpi)

- Describe how to re-run with custom rank order
Example: Observations and Suggestions

**MPI Grid Detection:** There appears to be point-to-point MPI communication in a 22 X 18 grid pattern. The 48.6% of the total execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named `MPICH_RANK_ORDER.Custom` was generated along with this report and contains the Custom rank order from the following table. This file also contains usage instructions and a table of alternative rank orders.

<table>
<thead>
<tr>
<th>Rank Order</th>
<th>On-Node Bytes/PE</th>
<th>On-Node Bytes/PE% of Total Bytes/PE</th>
<th>MPICH_RANK_REORDER_METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>7.80e+06</td>
<td>78.37%</td>
<td>3</td>
</tr>
<tr>
<td>SMP</td>
<td>5.59e+06</td>
<td>56.21%</td>
<td>1</td>
</tr>
<tr>
<td>Fold</td>
<td>2.59e+05</td>
<td>2.60%</td>
<td>2</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.00e+00</td>
<td>0.00%</td>
<td>0</td>
</tr>
</tbody>
</table>
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program: /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File: sweep3d.mpi+pat+27054-89t.ap2
# Number PEs: 48
# Max PEs/Node: 4
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.

...
Example 2 - Hycom

================= Observations and suggestions ==================

MPI grid detection:

There appears to be point-to-point MPI communication in a 33 X 41 grid pattern. The 26.1% of the total execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named MPICH_RANK_ORDER.Custom was generated along with this report and contains the Custom rank order from the following table. This file also contains usage instructions and a table of alternative rank orders.

<table>
<thead>
<tr>
<th>Rank Order</th>
<th>Bytes/PE</th>
<th>Bytes/PE% of Total Bytes/PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>1.20e+09</td>
<td>32.21%</td>
</tr>
<tr>
<td>SMP</td>
<td>8.70e+08</td>
<td>23.27%</td>
</tr>
<tr>
<td>Fold</td>
<td>3.55e+07</td>
<td>0.95%</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>1.99e+05</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

================= End Observations ==================
Example 2 - Hycom

- Run on 1353 MPI ranks, 24 ranks per node

- **Overall program wallclock:**
  - Default MPI rank order: 1450s
  - Custom MPI rank order: 1315s
  - ~10% improvement in execution time!

- **Time spent in MPI routines:**
  - Default rank order: 377s
  - Custom rank order: 303s
Loop Work Estimates
Loop Work Estimates

● Helps identify loops to optimize (parallelize serial loops):
  ● Loop timings approximate how much work exists within a loop
  ● Trip counts can be used to help carve up loop on GPU

● Enabled with CCE –h profile_generate option
  ● Should be done as separate experiment – compiler optimizations are restricted with this feature

● Loop statistics reported by default in pat_report table

● Next enhancement: integrate loop information in profile
  ● Get exclusive times and loops attributed to functions
Collecting Loop Statistics

- Load PrgEnv-cray software
- Load perftools software

- Compile **AND** link with –h profile_generate

- Instrument binary for tracing
  - pat_build –u my_program or
  - pat_build –w my_program

- Run application
- Create report with loop statistics
  - pat_report my_program.xf > loops_report
# Example Report – Loop Work Estimates

## 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>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time%</td>
<td></td>
<td></td>
<td></td>
<td>PE=HIDE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Thread=HIDE</td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>176.687480</td>
<td>--</td>
<td>--</td>
<td>17108.0</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>85.3%</td>
<td>150.789559</td>
<td>--</td>
<td>--</td>
<td>8.0</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td>85.0%</td>
<td>150.215785</td>
<td>24.876709</td>
<td>14.4%</td>
<td>2.0</td>
<td>jacobi_.LOOPS</td>
<td></td>
</tr>
<tr>
<td>12.2%</td>
<td>21.600616</td>
<td>--</td>
<td>--</td>
<td>16071.0</td>
<td>MPI</td>
<td></td>
</tr>
<tr>
<td>11.9%</td>
<td>21.104488</td>
<td>41.016738</td>
<td>67.1%</td>
<td>3009.0</td>
<td>mpi_waitall</td>
<td></td>
</tr>
<tr>
<td>2.4%</td>
<td>4.297301</td>
<td>--</td>
<td>--</td>
<td>1007.0</td>
<td>MPI_SYNC</td>
<td></td>
</tr>
<tr>
<td>2.4%</td>
<td>4.166092</td>
<td>4.135016</td>
<td>99.3%</td>
<td>1004.0</td>
<td>mpi_allreduce_(sync)</td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Inclusive Loop Time from -hprofile_generate

<table>
<thead>
<tr>
<th>Loop Incl</th>
<th>Loop</th>
<th>Loop</th>
<th>Loop</th>
<th>Function=/&gt;.LOOP[.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Hit</td>
<td>Trips</td>
<td>Trips</td>
<td>PE=HIDE</td>
</tr>
<tr>
<td>Total</td>
<td>Min</td>
<td>Max</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
... |
| 175.676881 | 2 | 0 | 1003 | jacobi_LOOP.07.li.267 |
| 0.917107  | 1003 | 0 | 260 | jacobi_LOOP.08.li.276 |
| 0.907515  | 129888 | 0 | 260 | jacobi_LOOP.09.li.277 |
| 0.446784  | 1003 | 0 | 260 | jacobi_LOOP.10.li.288 |
| 0.425763  | 129888 | 0 | 516 | jacobi_LOOP.11.li.289 |
| 0.395003  | 1003 | 0 | 260 | jacobi_LOOP.12.li.300 |
| 0.374206  | 129888 | 0 | 516 | jacobi_LOOP.13.li.301 |
| 126.250610| 1003 | 0 | 256 | jacobi_LOOP.14.li.312 |
| 126.223035| 127882 | 0 | 256 | jacobi_LOOP.15.li.313 |
| 124.298650| 16305019 | 0 | 512 | jacobi_LOOP.16.li.314 |
| 20.875086 | 1003 | 0 | 256 | jacobi_LOOP.17.li.336 |
| 20.862715 | 127882 | 0 | 256 | jacobi_LOOP.18.li.337 |
| 19.428085 | 16305019 | 0 | 512 | jacobi_LOOP.19.li.338 |
```
```
Other Interesting Performance Data
Program Instrumentation – Sampling

- 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
-g tracegroup (subset)

- blas: Basic Linear Algebra subprograms
- CAF: Co-Array Fortran (Cray CCE compiler only)
- HDF5: manages extremely large and complex data collections
- heap: dynamic heap
- io: includes stdio and sysio groups
- lapack: Linear Algebra Package
- math: ANSI math
- mpi: MPI
- omp: OpenMP API
- omp-rtl: OpenMP runtime library (not supported on Catamount)
- pthreads: POSIX threads (not supported on Catamount)
- shmem: SHMEM
- sysio: I/O system calls
- system: system calls
- upc: Unified Parallel C (Cray CCE compiler only)

For a full list, please see man pat_build
Specific Tables in pat_report

heidi@kaibab:/lus/scratch/heidi> pat_report -O -h

pat_report: Help for -O option:
Available option values are in left column, a prefix can be specified:

- ct
- defaults
- heap
- io
- lb
- load_balance
- mpi
---

-D1_D2_observation
-D1_D2_util
-D1_observation

Observation about Functions with low D1+D2 cache hit ratio
Functions with low D1+D2 cache hit ratio
Observation about Functions with low D1 cache hit ratio
Functions with low D1 cache hit ratio
Observation about Functions with low TLB refs/miss
Functions with low TLB refs/miss

<Tables that would appear by default.>
-0 calltree
-0 heap_program,.heap_hiwater,heap_leaks
-0 read_stats,write_stats
-0 load_balance
-0 lb_program,lb_group,lb_function
-0 mpi_callers
Heap Statistics

- g heap
  - calloc, cfree, malloc, free, malloc_trim, malloc_usable_size, mallocpt, memalign, posix_memalign, pvalloc, realloc, valloc

- g heap
- g sheap
- g shmemb
  - shfree, shfree_nb, shmalloc, shmalloc_nb, shrealloc

- g upc  (automatic with –O apa)
  - upc_alloc, upc_all_alloc, upc_all_free, uc_all_lock_alloc, upc_all_lock_free, upc_free, upc_global_alloc, upc_global_lock_alloc, upc_lock_free
Notes for table 5:

Table option:
-0 heap_hiwater
Options implied by table option:
-d am@,ub,ta,ua,tf,nf,ac,ab -b pe=[mmm]

This table shows only lines with Tracked Heap HiWater MBytes > 0.

Table 5: Heap Stats during Main Program

<table>
<thead>
<tr>
<th>Tracked Heap</th>
<th>Total Allocs</th>
<th>Total Frees</th>
<th>Tracked Objects</th>
<th>Tracked MBytes</th>
<th>PE MBytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiWater MBytes</td>
<td>Freed</td>
<td>Freed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.794</td>
<td>915</td>
<td>910</td>
<td>4</td>
<td>1.011</td>
<td>Total</td>
</tr>
<tr>
<td>9.943</td>
<td>1170</td>
<td>1103</td>
<td>68</td>
<td>1.046</td>
<td>pe.0</td>
</tr>
<tr>
<td>9.909</td>
<td>715</td>
<td>712</td>
<td>3</td>
<td>1.010</td>
<td>pe.22</td>
</tr>
<tr>
<td>9.446</td>
<td>1278</td>
<td>1275</td>
<td>3</td>
<td>1.010</td>
<td>pe.43</td>
</tr>
</tbody>
</table>

Cray Inc.
CrayPat API - For Fine Grain Instrumentation

- **Fortran**
  ```fortran
  include "pat_apif.h"
  ...
  call PAT_region_begin(id, "label", ierr)
  do i = 1,n
  ...
  enddo
  call PAT_region_end(id, ierr)
  ```

- **C & C++**
  ```c
  include <pat_api.h>
  ...
  ierr = PAT_region_begin(id, "label");
  < code segment >
  ierr = PAT_region_end(id);
  ```
PGAS (UPC, CAF) Support
Profiles of a PGAS program can be created to show:
- Top time consuming functions/line numbers in the code
- Load imbalance information
- Performance statistics attributed to user source by default
- Can expose statistics by library as well
  - To see underlying operations, such as wait time on barriers

Data collection is based on methods used for MPI library
- PGAS data is collected by default when using Automatic Profiling Analysis (pat_build –O apa)
- Predefined wrappers for runtime libraries (caf, upc, pgas) enable attribution of samples or time to user source

UPC and SHMEM heap tracking available
- -g heap will track shared heap in addition to local heap
## Table 1: Profile by Function

| Group | Samp % | Samp | Imb.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100.0%</td>
<td>48</td>
<td>--</td>
</tr>
<tr>
<td>USER</td>
<td>95.8%</td>
<td>46</td>
<td>--</td>
</tr>
<tr>
<td>all2all</td>
<td>83.3%</td>
<td>40</td>
<td>1.00</td>
</tr>
<tr>
<td>do_cksum</td>
<td>6.2%</td>
<td>3</td>
<td>0.50</td>
</tr>
<tr>
<td>do_all2all</td>
<td>2.1%</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>mpp_accum_long</td>
<td>2.1%</td>
<td>1</td>
<td>0.50</td>
</tr>
<tr>
<td>mpp_alloc</td>
<td>2.1%</td>
<td>1</td>
<td>0.50</td>
</tr>
<tr>
<td>ETC</td>
<td>4.2%</td>
<td>2</td>
<td>--</td>
</tr>
<tr>
<td>bzero</td>
<td>4.2%</td>
<td>2</td>
<td>0.50</td>
</tr>
</tbody>
</table>
### Table 2: Profile by Group, Function, and Line

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Source</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE='HIDE'</td>
</tr>
</tbody>
</table>

| 100.0% | 48 | -- | -- | Total |
|---------------------------------------------|
| 95.8% | 46 | -- | -- | USER |
|---------------------------------------------|
| 83.3% | 40 | -- | -- | all2all |
| 6.2% | 3 | -- | -- | do_cksum |
| 2.1% | 1 | 0.25 | 33.3% | line.315 |
| 4.2% | 2 | 0.25 | 16.7% | line.316 |
## PGAS Report Showing Library Functions with Callers

Table 1: Profile by Function and Callers, with Line Numbers

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Caller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE='HIDE'</td>
</tr>
<tr>
<td>100.0%</td>
<td>47</td>
<td>Total</td>
</tr>
<tr>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93.6%</td>
<td>44</td>
<td>ETC</td>
</tr>
<tr>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>85.1%</td>
<td>40</td>
<td>upc_memput</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>all2all:mp_bench.c:line.298</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>do_all2all:mp_bench.c:line.348</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>main:test_all2all.c:line.70</td>
</tr>
<tr>
<td>4.3%</td>
<td>2</td>
<td>bzero</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>(N/A):(N/A):line.0</td>
</tr>
<tr>
<td>2.1%</td>
<td>1</td>
<td>upc_all_alloc</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>mpp_alloc:mp_bench.c:line.143</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>main:test_all2all.c:line.25</td>
</tr>
<tr>
<td>2.1%</td>
<td>1</td>
<td>upc_all_reduceUL</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>mpp_accum_long:mp_bench.c:line.185</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>do_cksum:mp_bench.c:line.317</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>do_all2all:mp_bench.c:line.341</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>main:test_all2all.c:line.70</td>
</tr>
<tr>
<td>---------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OpenMP Support
OpenMP Data Collection and Reporting

- Measure overhead incurred entering and leaving
  - Parallel regions
  - Work-sharing constructs within parallel regions

- Show per-thread timings and other data

- Trace entry points automatically inserted by Cray and PGI compilers
  - Provides per-thread information

- Can use sampling to get performance data without API
  (per process view… no per-thread counters)
  - Run with OMP_NUM_THREADS=1 during sampling
  - Watch for calls to omp_set_num_threads()
OpenMP Data Collection and Reporting (2)

- Load imbalance calculated across all threads in all ranks for mixed MPI/OpenMP programs
  - Can choose to see imbalance to each programming model separately

- Data displayed by default in pat_report (no options needed)
  - Focus on where program is spending its time
  - Assumes all requested resources should be used
Imbalance Options for Data Display (pat_report –O ...)

- **profile_pe.th (default view)**
  - Imbalance based on the set of all threads in the program

- **profile_pe_th**
  - Highlights imbalance across MPI ranks
  - Uses max for thread aggregation to avoid showing under-performers
  - Aggregated thread data merged into MPI rank data

- **profile_th_pe**
  - For each thread, show imbalance over MPI ranks
  - Example: Load imbalance shown where thread 4 in each MPI rank didn’t get much work
### 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>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PE.Thread='HIDE'</td>
</tr>
<tr>
<td>100.0%</td>
<td>12.548996</td>
<td>--</td>
<td>--</td>
<td>7944.7</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>97.8%</td>
<td>12.277316</td>
<td>--</td>
<td>--</td>
<td>3371.8</td>
<td>USER</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.6%</td>
<td>4.473536</td>
<td>0.072259</td>
<td>1.6%</td>
<td>498.0</td>
<td>calc3_.LOOP@li.96</td>
<td></td>
</tr>
<tr>
<td>29.1%</td>
<td>3.653288</td>
<td>0.070551</td>
<td>1.9%</td>
<td>500.0</td>
<td>calc2_.LOOP@li.74</td>
<td></td>
</tr>
<tr>
<td>28.3%</td>
<td>3.545677</td>
<td>0.056303</td>
<td>1.6%</td>
<td>500.0</td>
<td>calc1_.LOOP@li.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>0.155028</td>
<td>--</td>
<td>--</td>
<td>1000.5</td>
<td>MPI_SYNC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>0.154899</td>
<td>0.674518</td>
<td>82.0%</td>
<td>999.0</td>
<td>mpi_barrier_(sync)</td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000129</td>
<td>0.000489</td>
<td>79.8%</td>
<td>1.5</td>
<td>mpi_reduce_(sync)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.7%</td>
<td>0.082943</td>
<td>--</td>
<td>--</td>
<td>3197.2</td>
<td>MPI</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.4%</td>
<td>0.047471</td>
<td>0.158820</td>
<td>77.6%</td>
<td>999.0</td>
<td>mpi_barrier_</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.015157</td>
<td>0.295055</td>
<td>95.9%</td>
<td>297.1</td>
<td>mpi_waitall_</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3%</td>
<td>0.033683</td>
<td>--</td>
<td>--</td>
<td>374.5</td>
<td>OMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.013098</td>
<td>0.078620</td>
<td>86.4%</td>
<td>125.0</td>
<td>calc2_.REGION@li.74(ovhd)</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.010298</td>
<td>0.052760</td>
<td>84.3%</td>
<td>124.5</td>
<td>calc3_.REGION@li.96(ovhd)</td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.010287</td>
<td>0.068428</td>
<td>87.6%</td>
<td>125.0</td>
<td>calc1_.REGION@li.69(ovhd)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000027</td>
<td>0.000128</td>
<td>83.0%</td>
<td>0.8</td>
<td>PTHREAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OpenMP Parallel DOs `<function>..<region>@<line>` automatically instrumented

OpenMP overhead is normally small and is filtered out on the default report (< 0.5%). When using “–T” the filter is deactivated
Hardware Counters Information at Loop Level

<table>
<thead>
<tr>
<th>USER / calc3_.LOOP@li.96</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time %</td>
</tr>
<tr>
<td>Time</td>
</tr>
<tr>
<td>Imb. Time</td>
</tr>
<tr>
<td>Imb. Time %</td>
</tr>
<tr>
<td>Calls</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS:</td>
</tr>
<tr>
<td>L2_MODIFIED:L2_OWNED:</td>
</tr>
<tr>
<td>L2_EXCLUSIVE:L2_SHARED</td>
</tr>
<tr>
<td>DATA_CACHE_REFILLS_FROM_SYSTEM:</td>
</tr>
<tr>
<td>ALL</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
</tr>
<tr>
<td>User time (approx)</td>
</tr>
<tr>
<td>Average Time per Call</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
</tr>
<tr>
<td>D1 cache utilization (refills)</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
</tr>
<tr>
<td>System to D1 refill</td>
</tr>
<tr>
<td>System to D1 bandwidth</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
</tr>
</tbody>
</table>

========================================================================
Caveats

● **No support for nested parallel regions**
  ● To work around this until addressed disable nested regions by setting `OMP_NESTED=0`
  ● Watch for calls to `omp_set_nested()`

● **If compiler merges 2 or more parallel regions, OpenMP trace points are not merged correctly**
  ● To work around this until addressed, use `-h thread1`

● **We need to add tracing support for barriers (both implicit and explicit)**
  ● Need support from compilers
Try Adding OpenMP to an MPI Code When…

● **When code is network bound**
  - Look at collective time, excluding sync time: this goes up as network becomes a problem
  - Look at point-to-point wait times: if these go up, network may be a problem

● **When MPI starts leveling off**
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue

● **Adding OpenMP to memory bound codes may aggravate memory bandwidth issues, but you have more control when optimizing for cache**
Questions
??
Cray Performance Measurement and Analysis Tools

Heidi Poxon
Manager & Technical Lead, Performance Tools
Cray Inc.
Trace Analysis
Tracing

- **Only true function calls can be traced**
  - Functions that are inlined by the compiler or that have local scope in a compilation unit cannot be traced

- **Enabled with pat_build –g, -u, -T or –w options**

- **Full trace (sequence of events) enabled by setting PAT_RT_SUMMARY=0**
  - Warning: trace files are not scalable
    - Tend to generate huge performance files
Suggestions for Controlling Large Traces

Several environment variables are available to limit trace files to a more reasonable size:

- **PAT_RT_CALLSTACK**
  - Limit the depth to trace the call stack

- **PAT_RT_HWPC**
  - Avoid collecting hardware counters (unset)

- **PAT_RT_RECORD_PE**
  - Collect trace for a subset of the PEs

- **PAT_RT_TRACE_FUNCTION_ARGS**
  - Limit the number of function arguments to be traced

- **PAT_RT_TRACE_FUNCTION_LIMITS**
  - Avoid tracing indicated functions

- **PAT_RT_TRACE_FUNCTION_MAX**
  - Limit the maximum number of traces generated for all functions for a single process
Suggestions for Controlling Large Traces (2)

- **PAT_RT_TRACE_THRESHOLD_PCT**
  - Specifies a % of time threshold to enforce when executing in full trace mode

- **PAT_RT_TRACE_THRESHOLD_TIME**
  - Specifies a time threshold to enforce when executing in full trace mode

- Set **PAT_RT_EXPRFILE_MAX** to the number of ranks (or any larger number)
  - Data for only 1 MPI rank stored in each .xf file

- Use **pat_region API** to start and stop tracing within a program
Controlling large traces - Additional API Functions

- **int PAT_state (int state)**
  - State can have one of the following:
    - PAT_STATE_ON
    - PAT_STATE_OFF
    - PAT_STATE_QUERY

- **int PAT_record (int state)**
  - Controls the state for all threads on the executing PE. As a rule, use \texttt{PAT_record()} unless there is a need for different behaviors for sampling and tracing
    - int \texttt{PAT_sampling_state (int state)}
    - int \texttt{PAT_tracing_state (int state)}

- **int PAT_trace_function (const void *addr, int state)**
  - Activates or deactivates the tracing of the instrumented function

- **int PAT_flush_buffer (void)**
include "pat_apif.h"
! Turn data recording off at the beginning of execution.
call PAT_record( PAT_STATE_OFF, istat )
...
! Turn data recording on for two regions of interest.
call PAT_record( PAT_STATE_ON, istat )
...
call PAT_region_begin( 1, "step 1", istat )
...
call PAT_region_end( 1, istat )
...
call PAT_region_begin( 2, "step 2", istat )
...
call PAT_region_end( 2, istat )
...
! Turn data recording off again.
call PAT_record( PAT_STATE_OFF, istat )
...
Questions
??