Cray Performance Measurement and Analysis Tools

Jason Beech-Brandt

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
  - √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

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 Apprentice^2**

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**

Files Generated and the Naming Convention

File Suffix	Description
a.out+pat	Program instrumented for data collection
a.out…s.xf	Raw data for sampling experiment, available after application execution
a.out…t.xf	Raw data for trace (summarized or full) experiment, available after application execution
a.out…st.ap2	Processed data, generated by pat_report, contains application symbol information
a.out…s.apa	Automatic profiling pnalysis template, generated by pat_report (based on pat_build –O apa experiment)
a.out+apa	Program instrumented using .apa file
MPICH_RANK_ORDER.Custom	Rank reorder file generated by pat_report from automatic grid detection an reorder suggestions

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**

```
% module load perftools
```

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

```
% make clean
% make
```

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

```
% 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>

```
% 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.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_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: -u 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%  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_

# ----------------------------------------------------------------------

  -o preqx.cray-xt.PE-2.1.56HD.pgi-8.0.amd64.pat-5.0.0.2.x+apa
# New instrumented 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/preqx.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 <apafile>.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
```

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)

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/)
  - http://lists.eecs.utk.edu/pipermail/perfapi-devel/2011-January/004078.html

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: Prefetches
20: FP, D1, TLB, MIPS <<-new for Interlagos
21: FP, D1, TLB, Stalls
22: D1, TLB, MemBW

Example: HW counter data and Derived Metrics

```
  PAPI_TLB_DM  Data translation lookaside buffer misses
  PAPI_L1_DCA  Level 1 data cache accesses
  PAPI_FP_OPS  Floating point operations
  DC_MISS      Data Cache Miss
  User_Cycles  Virtual Cycles
==================================================================
USER
------------------------------------------------------------------
  Time%                                         98.3%
  Time                                       4.434402 secs
  Imb.Time                                         -- secs
  Imb.Time%                                        --
  Calls                         0.001M/sec     4500.0 calls
  PAPI_L1_DCM                  14.820M/sec   65712197 misses
  PAPI_TLB_DM                   0.902M/sec    3998928 misses
  PAPI_L1_DCA                 333.331M/sec 1477996162 refs
  PAPI_FP_OPS                 445.571M/sec 1975672594 ops
  User time (approx)            4.434 secs 11971868993 cycles  100.0%Time
  Average Time per Call                      0.000985 sec
  CrayPat Overhead : Time        0.1%
  HW FP Ops / User time       445.571M/sec 1975672594 ops   4.1%peak(DP)
  HW FP Ops / WCT             445.533M/sec
  Computational intensity        0.17 ops/cycle    1.34 ops/ref
  MFLOPS (aggregate)          1782.28M/sec
  TLB utilization              369.60 refs/miss   0.722 avg uses
  D1 cache hit,miss ratios      95.6% hits         4.4% misses
  D1 cache utilization (misses)  22.49 refs/miss   2.811 avg hits
==================================================================
```

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

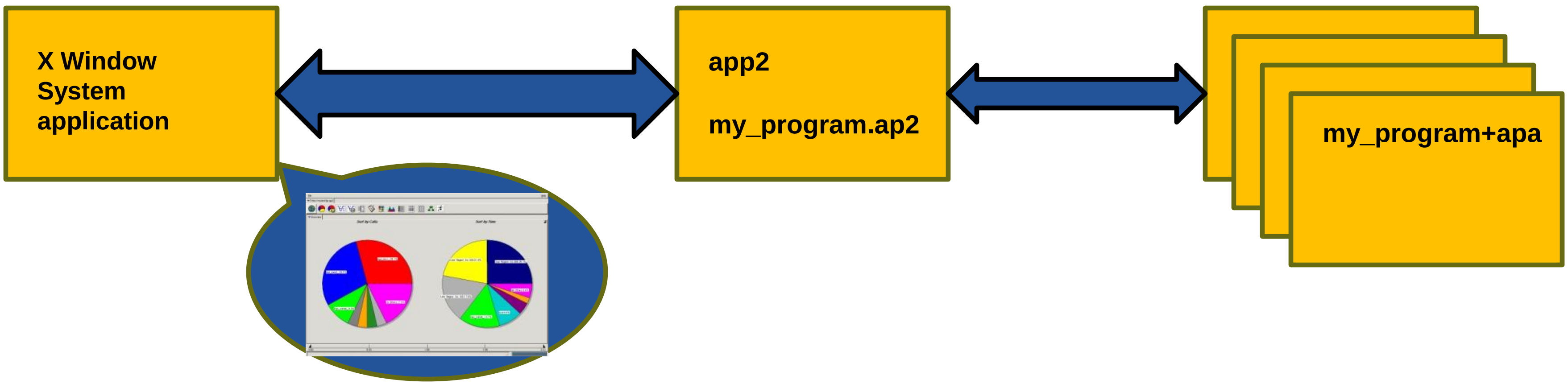
- **CPMD**
  - MPI, instrumented with pat_build –u, HWPC=1
  - 960 cores

	Perftools 5.1.3	Perftools 5.2.0
.xf -> .ap2	88.5 seconds	22.9 seconds
ap2 -> report	1512.27 seconds	49.6 seconds

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

	Perftools 5.1.3	Perftools 5.2.0
.xf -> .ap2	45.2 seconds	15.9 seconds
ap2 -> report	796.9 seconds	28.0 seconds

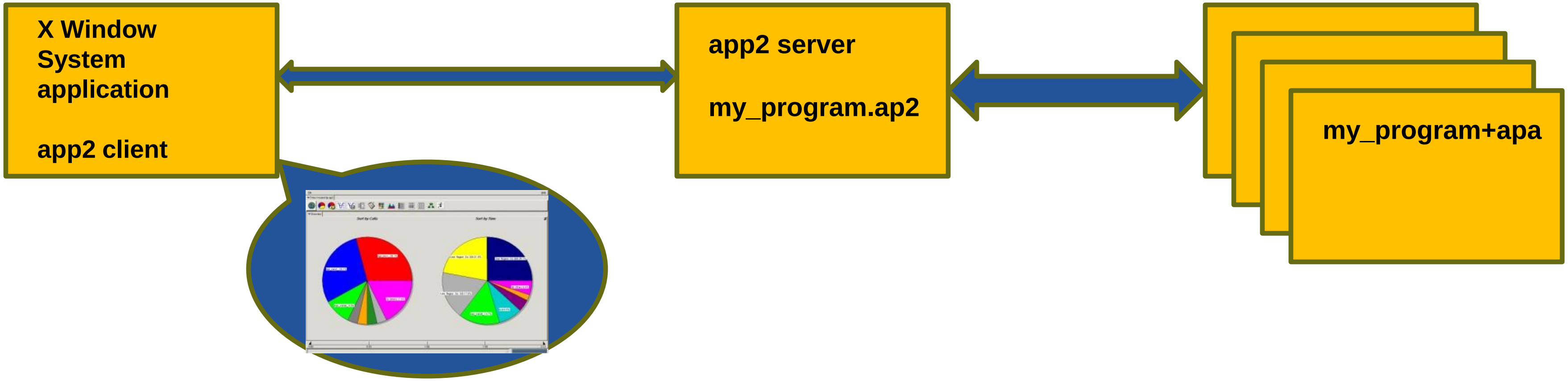
Old Client/Server (Cray Performance Tools 5.0.0)



- **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

pat_report: Job Execution Information

```
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:
…
```

pat_report: Table Notes

```
Notes for table 1:

  Table option:
    -O profile
  Options implied by table option:
    -d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE
  Other options:
    -T

  Options for related tables:
    -O profile_pe.th              -O profile_th_pe
    -O profile+src                -O load_balance
    -O callers                    -O callers+src
    -O calltree                   -O 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 avg for the PE values.
    (To specify different aggregations, see: 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:
      -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
...
```

Sampling Output (Table 1)

```
Notes for table 1:
...
Table 1:  Profile by Function

 Samp % | Samp |  Imb. |   Imb. |Group
        |      |  Samp | Samp % | Function
        |      |       |        |  PE='HIDE'

 100.0% |  775 |    -- |     -- |Total
|------------------------------------------
|  94.2% |  730 |    -- |     -- |USER
||-----------------------------------------
||  43.4% |  336 |  8.75 |   2.6% |mlwxyz_
||  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 1:  Profile by Function Group and Function

 Time % |        Time |Imb. Time |   Imb.  | Calls |Group
        |             |          | Time %  |       | Function
        |             |          |         |       |  PE='HIDE'

 100.0% | 104.593634 |        -- |      -- | 22649 |Total
|-----------------------------------------------------------
|  71.0% |  74.230520 |        -- |      -- | 10473 |MPI
||----------------------------------------------------------
||  69.7% |  72.905208 | 0.508369 |   0.7% |   125 |mpi_allreduce_
||   1.0% |   1.050931 | 0.030042 |   2.8% |    94 |mpi_alltoall_
||==========================================================
|  25.3% |  26.514029 |        -- |      -- |    73 |USER
||----------------------------------------------------------
||  16.7% |  17.461110 | 0.329532 |   1.9% |    23 |selfgravity_
||   7.7% |   8.078474 | 0.114913 |   1.4% |    48 |ffte4_
||==========================================================
|   2.5% |   2.659429 |        -- |      -- |   435 |MPI_SYNC
||----------------------------------------------------------
||   2.1% |   2.207467 | 0.768347 |  26.2% |   172 |mpi_barrier_(sync)
||==========================================================
|   1.1% |   1.188998 |        -- |      -- | 11608 |HEAP
||----------------------------------------------------------
||   1.1% |   1.166707 | 0.142473 |  11.1% |  5235 |free
|===========================================================
```

pat_report: Message Stats by Caller

```
Table 4:  MPI Message Stats by Caller

    MPI Msg |MPI Msg |  MsgSz |  4KB<= |Function
      Bytes |  Count |   <16B | MsgSz | Caller
            |        |  Count | <64KB |   PE[mmm]
            |        |        | Count |

 15138076.0 | 4099.4 |  411.6 | 3687.8 |Total
|-----------------------------------------------
| 15138028.0 | 4093.4 |  405.6 | 3687.8 |MPI_ISEND
||----------------------------------------------
||   8080500.0 | 2062.5 |    93.8 | 1968.8 |calc2_
3|             |        |         |        | MAIN_
||||--------------------------------------------
4|||   8216000.0 | 3000.0 | 1000.0 | 2000.0 |pe.0
4|||   8208000.0 | 2000.0 |     -- | 2000.0 |pe.9
4|||   6160000.0 | 2000.0 |  500.0 | 1500.0 |pe.15
||||============================================
||   6285250.0 | 1656.2 |   125.0 | 1531.2 |calc1_
3|             |        |         |        | MAIN_
||||--------------------------------------------
4|||   8216000.0 | 3000.0 | 1000.0 | 2000.0 |pe.0
4|||   6156000.0 | 1500.0 |     -- | 1500.0 |pe.3
4|||   6156000.0 | 1500.0 |     -- | 1500.0 |pe.5
||||============================================
.  .  .
```

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.

Rank Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	MPICH_RANK_REORDER_METHOD
Custom	7.80e+06	78.37%	3
SMP	5.59e+06	56.21%	1
Fold	2.59e+05	2.60%	2
RoundRobin	0.00e+00	0.00%	0

MPICH_RANK_ORDER File Example

```
# 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.

        Rank    On-Node    On-Node  MPICH_RANK_REORDER_METHOD
       Order  Bytes/PE  Bytes/PE%
                        of Total
                        Bytes/PE

       Custom   1.20e+09     32.21%  3
          SMP   8.70e+08     23.27%  1
         Fold   3.55e+07      0.95%  2
   RoundRobin   1.99e+05      0.01%  0
================  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
 Time%  |      Time  |     Imb.  |  Imb.  |  Calls  |Group
        |            |     Time  | Time%  |         | Function
        |            |           |        |         |  PE=HIDE
        |            |           |        |         |   Thread=HIDE

 100.0% | 176.687480 |        -- |     -- | 17108.0 |Total
|--------------------------------------------------------------------
|  85.3% | 150.789559 |        -- |     -- |     8.0 |USER
||-------------------------------------------------------------------
|  85.0% | 150.215785 | 24.876709 |  14.4% |     2.0 | jacobi_.LOOPS
||===================================================================
|  12.2% |  21.600616 |        -- |     -- | 16071.0 |MPI
||-------------------------------------------------------------------
|  11.9% |  21.104488 | 41.016738 |  67.1% |  3009.0 | mpi_waitall
||===================================================================
|   2.4% |   4.297301 |        -- |     -- |  1007.0 |MPI_SYNC
||-------------------------------------------------------------------
|   2.4% |   4.166092 |  4.135016 |  99.3% |  1004.0 | mpi_allreduce_(sync)
|====================================================================
```

Example Report – Loop Work Estimates (2)

Table 3: Inclusive Loop Time from -hprofile_generate

```
 Loop Incl |   Loop  |  Loop  |  Loop  |Function=/.LOOP[.]
    Time  |     Hit  | Trips  | Trips  | PE=HIDE
   Total  |          |  Min  |  Max  |
|------------------------------------------------------------
...
| 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
- pthreads — POSIX threads
- 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                          -O calltree
  defaults                    <Tables that would appear by default.>
  heap                        -O heap_program,heap_hiwater,heap_leaks
  io                          -O read_stats,write_stats
  lb                          -O load_balance
  load_balance                -O lb_program,lb_group,lb_function
  mpi                         -O mpi_callers
  ---
  D1_D2_observation           Observation about Functions with low
D1+D2 cache hit ratio
  D1_D2_util                  Functions with low D1+D2 cache hit ratio
  D1_observation              Observation about Functions with low D1
cache hit ratio
  D1_util                     Functions with low D1 cache hit ratio
  TLB_observation             Observation about Functions with low TLB
refs/miss
  TLB_util                    Functions with low TLB refs/miss
```

Heap Statistics

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

- **-g heap**
- **-g sheap**
- **-g shmem**
  - 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

Heap Statistics

```
Notes for table 5:

  Table option:
    -O 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 >


Table 5:  Heap Stats during Main Program

 Tracked |  Total |  Total | Tracked | Tracked |PE[mmm]
    Heap | Allocs |  Frees | Objects |  MBytes |
 HiWater |        |        |     Not |     Not |
  MBytes |        |        |   Freed |   Freed |

   9.794 |    915 |    910 |       4 |   1.011 |Total
|-----------------------------------------------------
|   9.943 |   1170 |   1103 |      68 |   1.046 |pe.0
|   9.909 |    715 |    712 |       3 |   1.010 |pe.22
|   9.446 |   1278 |   1275 |       3 |   1.010 |pe.43
|=====================================================
```

CrayPat API - For Fine Grain Instrumentation

- **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++**

```
include <pat_api.h>
…
ierr = PAT_region_begin(id, "label");
< code segment >
ierr = PAT_region_end(id);
```

PGAS (UPC, CAF) Support

PGAS 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

PGAS Default Report Table 1

```
Table 1:  Profile by Function

 Samp % | Samp | Imb. |   Imb. |Group
        |      | Samp | Samp % | Function
        |      |      |        |  PE='HIDE'

 100.0% |   48 |   -- |     -- |Total
|-------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||------------------------------------------
||  83.3% |   40 | 1.00 |   3.3% |all2all
||   6.2% |    3 | 0.50 |  22.2% |do_cksum
||   2.1% |    1 | 1.00 |  66.7% |do_all2all
||   2.1% |    1 | 0.50 |  66.7% |mpp_accum_long
||   2.1% |    1 | 0.50 |  66.7% |mpp_alloc
||==========================================
|   4.2% |    2 |   -- |     -- |ETC
||------------------------------------------
||   4.2% |    2 | 0.50 |  33.3% |bzero
|===========================================
```

PGAS Default Report Table 2

```
Table 2:  Profile by Group, Function, and Line

 Samp % | Samp | Imb. |   Imb. |Group
        |      | Samp | Samp % | Function
        |      |      |        |  Source
        |      |      |        |   Line
        |      |      |        |    PE='HIDE'

 100.0% |   48 |   -- |     -- |Total
|--------------------------------------------
|  95.8% |   46 |   -- |     -- |USER
||-------------------------------------------
||  83.3% |   40 |   -- |     -- |all2all
3|        |      |      |        | mpp_bench.c
4|        |      |      |        |  line.298
||   6.2% |    3 |   -- |     -- |do_cksum
3|        |      |      |        | mpp_bench.c
||||-----------------------------------------
4|||   2.1% |    1 | 0.25 |  33.3% |line.315
4|||   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
 Samp % | Samp |Group
        |      | Function
        |      |  Caller
        |      |   PE='HIDE'
 100.0% |   47 |Total
|---------------------------
|  93.6% |   44 |ETC
||--------------------------
||  85.1% |   40 |upc_memput
3|        |      | all2all:mpp_bench.c:line.298
4|        |      |  do_all2all:mpp_bench.c:line.348
5|        |      |   main:test_all2all.c:line.70
||   4.3% |    2 |bzero
3|        |      | (N/A):(N/A):line.0
||   2.1% |    1 |upc_all_alloc
3|        |      | mpp_alloc:mpp_bench.c:line.143
4|        |      |  main:test_all2all.c:line.25
||   2.1% |    1 |upc_all_reduceUL
3|        |      | mpp_accum_long:mpp_bench.c:line.185
4|        |      |  do_cksum:mpp_bench.c:line.317
5|        |      |   do_all2all:mpp_bench.c:line.341
6|        |      |    main:test_all2all.c:line.70
||==========================
```

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

Profile by Function Group and Function (with –T)

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

 Time % |       Time |Imb. Time |   Imb. |    Calls |Group
        |            |          | Time % |          | Function
        |            |          |        |          |  PE.Thread='HIDE'

 100.0% | 12.548996 |       -- |     -- |  7944.7 |Total
|-----------------------------------------------------------------------
|  97.8% | 12.277316 |       -- |     -- |  3371.8 |USER
||----------------------------------------------------------------------
||  35.6% |  4.473536 | 0.072259 |   1.6% |   498.0 |calc3_.LOOP@li.96
||  29.1% |  3.653288 | 0.070551 |   1.9% |   500.0 |calc2_.LOOP@li.74
||  28.3% |  3.545677 | 0.056303 |   1.6% |   500.0 |calc1_.LOOP@li.69
. . .
||======================================================================
|   1.2% |  0.155028 |       -- |     -- |  1000.5 |MPI_SYNC
||----------------------------------------------------------------------
||   1.2% |  0.154899 | 0.674518 |  82.0% |   999.0 |mpi_barrier_(sync)
||   0.0% |  0.000129 | 0.000489 |  79.8% |     1.5 |mpi_reduce_(sync)
||======================================================================
|   0.7% |  0.082943 |       -- |     -- |  3197.2 |MPI
||----------------------------------------------------------------------
||   0.4% |  0.047471 | 0.158820 |  77.6% |   999.0 |mpi_barrier_
||   0.1% |  0.015157 | 0.295055 |  95.9% |   297.1 |mpi_waitall_
. . .
||======================================================================
|   0.3% |  0.033683 |       -- |     -- |   374.5 |OMP
||----------------------------------------------------------------------
||   0.1% |  0.013098 | 0.078620 |  86.4% |   125.0 |calc2_.REGION@li.74(ovhd)
||   0.1% |  0.010298 | 0.052760 |  84.3% |   124.5 |calc3_.REGION@li.96(ovhd)
||   0.1% |  0.010287 | 0.068428 |  87.6% |   125.0 |calc1_.REGION@li.69(ovhd)
||======================================================================
|   0.0% |  0.000027 | 0.000128 |  83.0% |     0.8 |PTHREAD
|        |           |          |        |         | pthread_create
|=======================================================================
```

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

```
========================================================================
USER / calc3_.LOOP@li.96
------------------------------------------------------------------------
  Time%                                                 37.3%
  Time                                               6.826587 secs
  Imb.Time                                           0.039858 secs
  Imb.Time%                                              0.6%
  Calls                            72.9 /sec            498.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED       64.364M/sec     439531950 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                          10.760M/sec      73477950 fills
  PAPI_L1_DCM                    64.973M/sec     443686857 misses
  PAPI_L1_DCA                   135.699M/sec     926662773 refs
  User time (approx)              6.829 secs   15706256693 cycles  100.0%Time
  Average Time per Call                           0.013708 sec
  CrayPat Overhead : Time          0.0%
  D1 cache hit,miss ratios        52.1% hits            47.9% misses
  D1 cache utilization (misses)    2.09 refs/miss       0.261 avg hits
  D1 cache utilization (refills)   1.81 refs/refill     0.226 avg uses
  D2 cache hit,miss ratio         85.7% hits            14.3% misses
  D1+D2 cache hit,miss ratio      93.1% hits             6.9% misses
  D1+D2 cache utilization         14.58 refs/miss       1.823 avg hits
  System to D1 refill            10.760M/sec      73477950 lines
  System to D1 bandwidth        656.738MB/sec   4702588826 bytes
  D2 to D1 bandwidth           3928.490MB/sec  28130044826 bytes
========================================================================
```

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

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_EXPFILE_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 PAT_record() unless there is a need for different behaviors for sampling and tracing
    - int PAT_sampling_state (int state)
    - int 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)**

Trace On / Trace Off Example

```
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
??