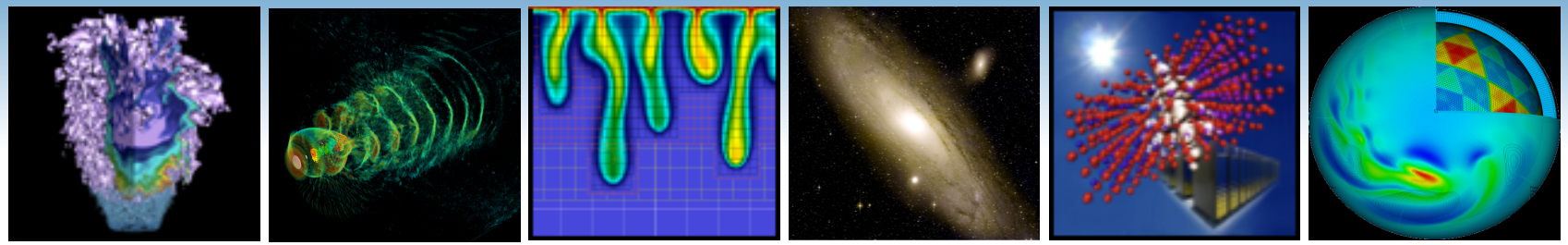


Using CrayPat

Woo-Sun Yang
NERSC User Services Group

February 1-2, 2012
NUG Training

NERSC Oakland Scientific Facility

U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC National Energy Research Scientific Computing Center

BERKELEY LAB

Introduction to CrayPat

- **Two types of collecting code performance data**
  - Sampling
    - Typically, sampling program counters (locations in the code) at specified time intervals
    - A few different variants exist in sampling methods
    - Can see how much time was spent in each function
  - Tracing
    - Focus placed on individual functions
    - Records performance data on entry to and exit from specified functions
    - More accurate about performance of the function
    - Need to specify which functions to trace

Introduction to CrayPat (cont'd)

- **CrayPat can perform "sampling" or "tracing" experiments**
  - Need to build an executable accordingly
  - Environment variable PAT_RT_EXPERIMENT denotes the type

- **Supports Fortran, C, C++, UPC, MPI, Coarray Fortran, OpenMP, Pthreads, SHMEM**

- **Terminology**
  - Instrument: create a new executable that is capable of collecting performance data
  - Experiment: act of gathering performance data by running an instrumented executable

CrayPat Outline

- Sampling (and example)
- Tracing
- .xf files
- pat_report
- Tracing examples: Heap, MPI, OpenMP
- APA (Automatic Program Analysis)
- CrayPat API
- Monitoring hardware performance counters

- Exercises provided:

```
Exercise info in this box;
available in $EXAMPLES/NUG2012/
perftools ($EXAMPLES defined
with 'module load training')
```

Using CrayPat

1. **Access the tools**
   - `module load perftools`
2. **Build your application; keep .o and .a files (compile and link in separate steps)**
   - `make clean`
   - `make`
3. **Instrument application**
   - `pat_build ... a.out`
   - Result is a new file, `a.out+pat`
4. **Run instrumented application to get top time consuming routines**
   - `aprun ... a.out+pat`
   - Result is a new file XXXXX.xf (or a directory containing .xf files)

Adjust script for +pat

5. **Run pat_report on that new file; view results**
   - `pat_report  XXXXX.xf  > my_profile`
   - `vi my_profile`
   - Result is also a new file: XXXXX.ap2

CrayPat Notes

- **Key points to remember:**
  - MUST load module prior to building your code; otherwise, you'll see

```
ERROR: Missing required ELF section 'link information' from
  the program ...
Load the correct 'perftools' module and rebuild the program
```

  - MUST load module prior to looking at man pages
  - MUST run your application in $SCRATCH
  - MUST leave relocatable binaries (*.o) when compiling

Running a Sampling Experiment

- **To build, use –S or simply without any tracing flag for pat_build**
  - `pat_build –S a.out` or
  - `pat_build a.out`

- **Run a.out+pat**
  - aprun –n … ./a.out+pat
  - By default you're running a **`samp_pc_time`** experiment (i.e., PAT_RT_EXPERIMENT = **`samp_pc_time`**) where PC's are sampled at a specified time interval, PAT_RT_INTERVAL (default: 10,000 microseconds)
  - See the intro_craypat man page for other sampling types

- **pat_report on .xf from a sampling experiment generates .apa file for suggested pat_build options for detailed tracing experiments (later on this)**

Sampling Example

```
> module load perftools
> ftn -c jacobi_serial.f90
> ftn -o jacobi_serial
  jacobi_serial.o
> pat_build jacobi_serial
```

Run jacobi_serial+pat in a batch job.

```
> pat_report
  jacobi_serial+pat+14196-6018s.xf
```

```
Table 1:  Profile by Function

 Samp%  | Samp  | Imb.  |  Im
        |       | Samp  | Samp%  | Function
        |       |       |        |  PE=HIDE

 100.0% |    19 |    -- |     -- |Total
|-----------------------------------------
| 100.0% |    19 |    -- |     -- |USER
||----------------------------------------
||  73.7% |    14 |  0.00 |   0.0% |MAIN_
||  26.3% |     5 |  0.00 |   0.0% |compute_diff_
|=========================================
```

Function Group

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

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

 100.0% |    19 |    -- |     -- |Total
|-----------------------------------------------
| 100.0% |    19 |    -- |     -- |USER
||----------------------------------------------
||  73.7% |    14 |    -- |     -- |MAIN_
3|        |       |       |        | jacobi_serial.f90
||||--------------------------------------------
4|||  52.6% |    10 |  0.00 |   0.0% |line.36
4|||  21.1% |     4 |  0.00 |   0.0% |line.54
||||============================================
||  26.3% |     5 |  0.00 |   0.0% |compute_diff_
3|        |       |       |        | jacobi_serial.f90
4|        |       |       |        |  line.147
|===============================================
```

```
Table 3:  Wall Clock Time, Memory High Water Mark

 Process  |  Process  |Total
    Time  |    HiMem  |
          | (MBytes)  |

 0.598643 |    21.574 |Total
```

Exercise in perftools/sampling

Tidbits about Sampling

- **Binary built for sampling doesn't work for a tracing experiment**

- **Binary built for tracing can be used for sampling ("trace-enhanced sampling")**
  - Set PAT_RT_EXPERIMENT to a sampling type
  - set PAT_RT_SAMPLING_MODE to 1 (raw user sampling) or 3 (bubble user sampling: aggregate all samples inside a call to a traced function)

pat_build for Tracing

- **To records performance data on entry and exit from functions**
  - Type of experiment: '`trace`'; no need to set PAT_RT_EXPERIMENT to it

- **`-T` *`function`*: list functions to trace**
  - `pat_build –w –T field_,grad_ a.out (will explain about –w soon)`
  - Learn the Unix `nm` or `readelf`:
    `nm mycode.o |grep " T "`
- **`-T !`*`function`*: do not trace *function***
  - `pat_build -u -T \!field_ a.out`
    - trace all user functions except field_
    - '\' to escape the '`!`' character in csh/tcsh
- **`-t` *`tracefile`*: trace all functions listed in the file *`tracefile`***

pat_build for Tracing

- **To records performance data on entry and exit from functions**
  - Type of experiment: '`trace`'; no need to set PAT_RT_EXPERIMENT to it
- **`-w`**
  - To trace user-defined functions if they are specified with `–t` and `–T`
  - If none is specified, "trace" the 'main' (i.e., entire code as a whole)
- **`-u`: trace all user functions routine by routine**
  - For source files owned by the user
  - Use care: binary runs longer and can fail

```
WARNING: Tracing small, frequently called functions
can add excessive overhead
WARNING: To set a minimum size, say 800 bytes, for
traced functions, use:
-D trace-text-size=800
```

pat_build for Tracing

- **`-T` *`function`*: trace *function***
  - `pat_build –w –T field_,grad_ a.out`
  - Learn the Unix `nm` or `readelf`:
    `nm mycode.o |grep " T "`

- **`-T !`*`function`*: do not trace *function***
  - `pat_build -u -T \!field_ a.out`
    - trace all user functions except field_
    - '\' to escape the '!' character in csh/tcsh

- **`-t` *`tracefile`*: trace all functions listed in the file *`tracefile`*.**

CrayPat Trace Function Groups

- **Predefined trace function groups**
- **`-g`: trace all functions in certain groups (e.g., MPI):**
  - `pat_build -g mpi,heap -u a.out`
  - `pat_build -g mpi -T \!MPI_Barrier a.out` trace all MPI calls except MPI_Barrier
- **See `$CRAYPAT_ROOT/lib/Trace*` for files that list what routines are traced**

- mpi
- omp
- pthreads
- caf
- upc
- shmem
- ga
- heap
- blas
- blacs
- lapack
- scalapack
- fftw
- petsc
- io
- netcdf
- hdf5
- …

Other pat_build Options

- **`-f`: overwrite an existing instrumented program**
- **`-o` *`instr_prog`*:**
  - use a different name for the instrumented executable instead of a.out+pat
  - can put ***`instr_prog`*** at the end of the command line without '-o' will do, too
- **`-O` *`optfile`*: use the pat_build options in the file *`optfile`*.**
  - Special argument '`-O apa`' will be discussed later

`Exercise in perftools/pat_build_examples`

.xf Files

- **Experiment data files; binary files**
- **Number of .xf files**
  - Single file (for ≤ 256 PEs) or directory containing multiple (~ sqrt(PEs)) files
  - Can be changed with `PAT_RT_EXPFILE_MAX`
- **Name convention**
  - **a.out+pat+<UNIX_PID>-<NODE_ID>[st].xf** (or a.out+apa +….xf; see APA)
  - s or t: for sampling or tracing
- **New one each time you run your application**
  - Can run as many experiments as you want
- **Can create .xf file(s) in a different directory other than the execution location by setting `PAT_RT_EXPFILE_DIR`**

pat_report

- **Generates from .xf data file(s)**
  - ASCII text report
  - .ap2 file
    - To be viewed with Apprentice2 GUI
    - To be used with pat_report to generate ASCII text report again later on
  - .apa file, in case of a sampling experiment

- **Important to create .ap2 file right after .xf file becomes available**
  - .xf file requires the instrumented executable in the original directory (not portable)
  - .ap2 doesn't (self-contained and portable)

pat_report Options

- **Too many options!**
- **`-T`: disable all thresholds for display**
  - By default, text output doesn't have any functions where less than 5% of run time is spent
- **`-d`: data items to display (time data, heap data, counter data,…)**
- **`-b`: how data is aggregated and labeled (group, function, pes, thread, …)**
- **`-s`: details of report appearance (aggregation, format,…)**
- **`-O|-b|-d|-s –h`: list all available cases for each option**

pat_report Options

- **-o: predefined report types**
  - This is what we should use
  - `Still many options!`
  - `profile`, `callers` (`ca`), `calltree` (`ct`), `ca+src`, `ct+src`, `heap`, `load_balance`, `mpi_callers`, `hwpc`, `nids`, …
  - `heap` and `load_balance` have a few "sub-reports"
    - `load_balance` = `load_balance_program` (for entire program)
      + `load_balance_group` (per function group)
      + `load_balance_function` (per function)
  - Examples:

```
-O profile: -d ti%@0.95,ti,imb_ti,imb_ti%,tr
            -b gr,fu,pe=HIDE
-O callers+src: -d ti%@0.95,ti,tr -b gr,fu,ca,pe=HIDE
                -s show_ca='fu,so,li'
-O load_balance_function: -d ti%@0.95,ti,tr
                          -b gr,fu,pe=[mmm]
```

pat_report Options

```
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,th=HIDE
  Other options:
...
Table 1:  Profile by Function Group and Function

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

 100.0% | 42.713439 |       -- |     -- |  525.0 |Total
|-----------------------------------------------------------------------
|  98.7% | 42.152167 |       -- |     -- |  107.0 |USER
||----------------------------------------------------------------------
||  75.3% | 32.142701 | 0.063853 |   0.2% |    1.0 |MAIN_
||  22.0% |  9.401592 | 0.013632 |   0.1% |   50.0 |compute_diff_
||   0.8% |  0.352469 | 0.014394 |   4.0% |    1.0 |init_fields_
||   0.6% |  0.250150 | 0.000025 |   0.0% |    1.0 |exit
||   0.0% |  0.004694 | 0.016204 |  78.6% |   51.0 |set_bc_
||   0.0% |  0.000515 | 0.036169 | 100.0% |    1.0 |read_params_
```

The actual –d and –b options corresponding to a certain report type can be found here, just above each table.

You can add a few extra –b or –s options options to customize for your needs.

Heap Memory Example

```
> module load perftools
> ftn -c jacobi_serial.f90
> ftn -o jacobi_serial
  jacobi_serial.o
> pat_build -g heap -u
  jacobi_serial
```

Run jacobi_serial+pat in a batch job.

```
> pat_report
  jacobi_serial+pat+9396-1090t.xf
```

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

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

 100.0% | 56.943925 |                     64.0 |Total
|-----------------------------------------------------------------------
|  99.9% | 56.891366 |       -- |     -- |   46.0 |USER
||----------------------------------------------------------------------
||  74.9% | 42.669334 | 0.000000 |   0.0% |    1.0 |MAIN_
||  23.1% | 13.136614 | 0.000000 |   0.0% |   20.0 |compute_diff_
||   1.4% |  0.792690 | 0.000000 |   0.0% |    1.0 |init_fields_
||======================================================================
|   0.1% |  0.052556 |       -- |     -- |   16.0 |HEAP
|   0.0% |  0.000003 |       -- |     -- |    2.0 |ETC
|=======================================================================
```

Heap function group

Exercise in perftools/heap

Heap Memory Example (cont'd)

```
Table 2:  Heap Stats during Main Program

Total


=============================================
  Total
---------------------------------------------
  Tracked Heap HiWater MBytes      1055.712
  MBytes Not Tracked                  0.000
  Total Allocs                           11
  Allocs Not Tracked                      0
  Total Frees                             7
  Inferred Frees                          0
  Tracked Objects Not Freed               4
  Tracked MBytes Not Freed          352.586
```

```
Table 3:  Heap Leaks during Main Program

 Tracked   | Tracked   | Tracked   |Caller
  MBytes   |  MBytes   | Objects   |
     Not   |     Not   |     Not   |
  Freed%   |   Freed   |   Freed   |

   100.0% |     1.023 |         3 |Total
|-----------------------------------------------
|    97.8% |     1.000 |         1 |allhdr
|     1.3% |     0.013 |         1 |__hpfio_alloc_fcb
=================================================
```

Sometimes not easy to understand

Exercise in perftools/heap

MPI Code Example

- Profiling by MPI functions
- MPI message stats
- load imbalance among MPI tasks

```
> module load perftools
> ftn -c jacobi_mpi.f90
> ftn -o jacobi_mpi
  jacobi_mpi.o
> pat_build -g mpi -u jacobi_mpi
```

Run jacobi_mpi+pat in a batch job.

```
> pat_report
  jacobi_mpi+pat+21036-6112t.xf
```

Exercise in perftools/mpi

MPI Code Example (cont'd)

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

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

 100.0% | 42.713439 |       -- |     -- |  525.0 |Total
|-----------------------------------------------------------
|  98.7% | 42.152167 |       -- |     -- |  107.0 |USER
||----------------------------------------------------------------
||  75.3% | 32.142701 | 0.063853 |   0.2% |    1.0 |MAIN_
||  22.0% |  9.401592 | 0.013632 |   0.1% |   50.0 |compute_diff_
...
||================================================================
|   1.2% |  0.516397 |       -- |     -- |   54.0 |MPI_SYNC
||----------------------------------------------------------------
||   0.9% |  0.372011 | 0.371891 | 100.0% |    4.0 |mpi_bcast_(sync)
||   0.3% |  0.144385 | 0.061092 |  42.3% |   50.0 |mpi_allreduce_(sync)
||================================================================
|   0.1% |  0.044876 |       -- |     -- |  364.0 |MPI
||----------------------------------------------------------------
||   0.1% |  0.039829 | 0.042106 |  52.1% |  100.0 |MPI_SENDRECV
||   0.0%                         81.0% |    1.0 |mpi_finalize
||   0.0%                         13.6% |   50.0 |MPI_ALLREDUCE
||   0.0% |  0.000138 | 0.000014 |   9.3% |  104.0 |MPI_COMM_SIZE
||   0.0% |  0.000041 | 0.000022 |  35.6% |    4.0 |mpi_bcast
||   0.0% |  0.000037 | 0.000008 |  18.3% |  104.0 |mpi_comm_rank
||   0.0% |  0.000001 | 0.000000 |  27.7% |    1.0 |MPI_INIT
|================================================================
```

No per-PE info

Function Groups

- Load imbalance
  - imb = max – avg
  - imb% = imb/max * npes/(npes-1)*100%
  - Imb% =0 for perfectly balanced
  - imb%=100 for all work being done by a single PE
- MPI_SYNC group for time spent waiting at the barrier before entering the collectives
- Actual time spent in the collectives goes to the MPI function group
- Not to separate these groups, set PAT_RT_MPI_SYNC to 0 before aprun

MPI Code Example (cont'd)

PEs with max, min, median values

```
Table 2:  Load Balance with MPI Message Stats

 Time%  |     Time  |   MPI   |   MPI Msg  |  Avg MPI  |Group
        |           |   Msg   |     Bytes  | Msg Size  | PE=[mmm]
        |           | Count   |            |           |

 100.0% | 42.713578 |  152.6 | 17039427.1 | 111652.60 |Total
|-----------------------------------------------------------------
|  98.7% | 42.152195 |    0.0 |        0.0 |        -- |USER
||----------------------------------------------------------------
||  98.9% | 42.245474 |    0.0 |        0.0 |        -- |pe.0
||  98.7% | 42.153875 |    0.0 |        0.0 |        -- |pe.38
||  98.6% | 42.103769 |    0.0 |        0.0 |        -- |pe.49
||================================================================
|   1.2% |  0.516411 |    0.0 |        0.0 |        -- |MPI_SYNC
||----------------------------------------------------------------
||   2.1% |  0.881347 |    0.0 |        0.0 |        -- |pe.20
||   1.2% |  0.515863 |    0.0 |        0.0 |        -- |pe.11
||   0.3% |  0.120332 |    0.0 |        0.0 |        -- |pe.0
||================================================================
|   0.1% |  0.044972 |  152.6 | 17039427.1 | 111652.60 |MPI
||----------------------------------------------------------------
||   0.2% |  0.085398 |  154.0 | 17279416.0 | 112204.00 |pe.22
||   0.1% |  0.043775 |  154.0 | 17279416.0 | 112204.00 |pe.47
||   0.0% |  0.012142 |  104.0 |  8639816.0 |  83075.15 |pe.0
```

Average over PEs

MPI Code Example (cont'd)

```
Table 3:  MPI Message Stats by Caller

   MPI Msg  |   MPI   | MsgSz   | 64KB<=  |Function
     Bytes  |   Msg   |  <16B   |  MsgSz  | Caller
            | Count   | Count   |   <1MB  |  PE=[mmm]
            |         |         |  Count  |

 17039427.1 |  152.6 |   54.0 |    98.6 |Total
|-----------------------------------------------------
| 17039211.1 |   98.6 |    0.0 |    98.6 |MPI_SENDRECV
|            |        |        |         | MAIN_
3            |        |        |         |  main
||||---------------------------------------------------
4||| 17279200.0 |  100.0 |    0.0 |   100.0 |pe.52
4||| 17279200.0 |  100.0 |    0.0 |   100.0 |pe.16
4|||  8639600.0 |   50.0 |    0.0 |    50.0 |pe.0
||||===================================================
|      200.0 |   50.0 |   50.0 |     0.0 |MPI_ALLREDUCE
|            |        |        |         | compute_diff_
3            |        |        |         |  MAIN_
4            |        |        |         |   main
|||||---------------------------------------------------
5||||      200.0 |   50.0 |   50.0 |     0.0 |pe.53
5||||      200.0 |   50.0 |   50.0 |     0.0 |pe.17
5||||      200.0 |   50.0 |   50.0 |     0.0 |pe.18
|||||===================================================
|       16.0 |    4.0 |    4.0 |     0.0 |mpi_bcast
|            |        |        |         | read_params_
3            |        |        |         |  MAIN_
4            |        |        |         |   main
|||||---------------------------------------------------
5||||       16.0 |    4.0 |    4.0 |     0.0 |pe.53
5||||       16.0 |    4.0 |    4.0 |     0.0 |pe.17
Processing table 6 of 6
      16.0 |    4.0 |    4.0 |     0.0 |pe.18
|=====================================================
```

Bins by message size

per PE

Level of depth

callers

OpenMP Code Example

```
> module load perftools
> ftn -mp=nonuma -c jacobi_omp.f90
> ftn -mp=nonuma -o jacobi_omp
   jacobi_omp.o
> pat_build -g omp -u jacobi_omp
```

Run jacobi_omp+pat in a batch job

```
> pat_report –O profile_pe.th
   jacobi_omp+pat+12056-910t.xf
```

`-O profile_pe.th` to show imbalance over all threads

Exercise in perftools/openmp

OpenMP Code Example

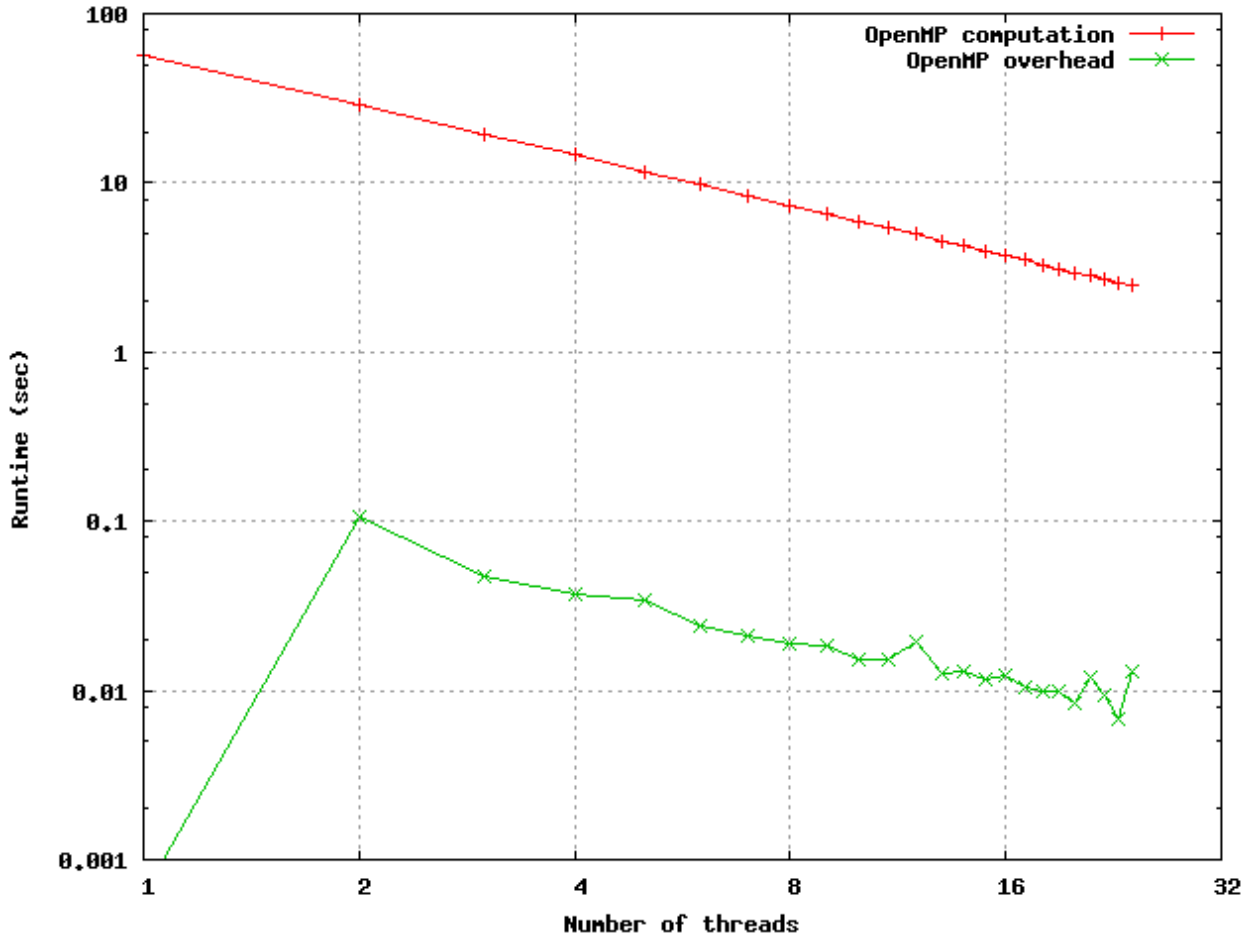
```
Table 1:  Profile by Function Group and Function

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

 100.0% | 2.869827 |       -- |     -- |  380.0 |Total
|-----------------------------------------------------------------------------
|  96.8% | 2.776862 |       -- |     -- |  253.0 |USER
||----------------------------------------------------------------------------
||  41.8% | 1.198865 | 0.002731 |   0.2% |   20.0 |MAIN_.LOOP@li.35
||  20.9% | 0.599795 | 0.008961 |   1.5% |   20.0 |MAIN_.LOOP@li.56
||  19.3% | 0.553771 | 0.000599 |   0.1% |   20.0 |compute_diff_.LOOP@li.166
||   9.0% | 0.258708 | 0.000000 |   0.0% |    1.0 |exit
||   2.1% | 0.060690 | 0.000000 |   0.0% |    1.0 |MAIN_
||   1.2% | 0.035291 | 0.000000 |   0.0% |    1.0 |read_params_
||   1.2% | 0.034470 | 0.000391 |   1.2% |    1.0 |init_fields_.LOOP@li.104
||   1.2% | 0.033483 | 0.000371 |   1.1% |    1.0 |init_fields_.LOOP@li.108
...
||============================================================================
|   3.1% | 0.089058 |       -- |     -- |  104.0 |OMP
||----------------------------------------------------------------------------
||   2.3% | 0.064671 | 0.000000 |   0.0% |    1.0 |init_fields_.REGION@li.104(ovhd)
||   0.4% | 0.011519 | 0.000000 |   0.0% |   20.0 |MAIN_.REGION@li.56(ovhd)
||   0.3% | 0.007579 | 0.000000 |   0.0% |   20.0 |MAIN_.REGION@li.35(ovhd)
||   0.1% | 0.003349 | 0.000000 |   0.0% |   20.0 |compute_diff_.REGION@li.166(ovhd)
||   0.0% | 0.001303 | 0.000000 |   0.0% |    1.0 |omp_set_num_threads
||   0.0% | 0.000387 | 0.000000 |   0.0% |   21.0 |set_bc_.REGION@li.138(ovhd)
||   0.0% | 0.000249 | 0.000000 |   0.0% |   21.0 |set_bc_.REGION@li.145(ovhd)
||============================================================================
|   0.1% | 0.003907 | 0.000000 |   0.0% |   23.0 |PTHREAD
||----------------------------------------------------------------------------
|   0.1% | 0.003907 | 0.000000 |   0.0% |   23.0 | pthread_create
|=============================================================================
```

No per-thread info

- 24 thread case
- Good load balance in parallel regions
- Overhead

OpenMP Code Example

Main loop in jacobi_omp.f90 ("first touched"); ngrid=6144 and maxiter=20



Automatic Program Analysis (APA)

- One may not know in advance where a large run time is spent; tracing all the functions can be overwhelming due to large overhead

1. Have the tool detect the most time consuming functions in the application with a quick sampling experiment

2. Feed this information back to the tool to instrument for focused data collection

3. Get performance information on the most significant parts of the application

- APA does this for you (of course, you can do the same thing by hand)

Automatic Program Analysis (APA)

1. **pat_build -O apa a.out**
   - Produces the instrumented executable a.out+pat for sampling
2. **aprun -n … a.out+pat**
   - Produces data file, e.g., a.out+pat+10766-981s.xf
3. **pat_report a.out+pat+10766-981s.xf**
   - Produces a.out+pat+10766-981s.apa (suggested options for tracing)
   - Produces a.out+pat+10766-981s.ap2
4. **Edit a.out+pat+10766-981s.apa, if necessary (next slide)**
5. **pat_build -O a.out+pat+10766-981s.apa**
   - Produces a.out+apa for tracing
6. **aprun -n … a.out+apa**
   - Produces a.out+apa+5899-1000t.xf
7. **pat_report a.out+apa+5899-1000t.xf > out**

Exercise in perftools/apa

.apa File

- **Recommended pat_build options for tracing**
- **Customize it for your need**
  - Include/exclude functions
  - Add/change options
- **Note that the suggestions may not be valid for a very different task/thread configuration**
  - Do APA using the targeted task/thread configuration

```
#  You can edit this file, if desired, and use it
#  to reinstrument the program for tracing like this:
#
#          pat_build -O jacobi_mpi+pat+10766-981s.apa
#
#  These suggested trace options are based on data from:
#
#    jacobi_mpi+pat+10766-981s.ap2
# ---------------------------------------------------------------------
#       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.
...
  -w  # Enable tracing of user-defined functions.
      # Note: -u should NOT be specified as an additional option.

# 66.89% 1869  bytes
         -T MAIN_

# 19.79% 304  bytes
#         -T compute_diff_
...
# ---------------------------------------------------------------------

-o jacobi_mpi+apa # New instrumented program.

/scratch2/scratchdirs/wyang/perftools/apa/jacobi_mpi # Original program.
```

Command to run

PAT_RT_HWPC set to 1

mpi trace group chosen

Trace ‘MAIN_’, but not ‘compute_diff_’

Sampling run result

Program to instrument

Instrumented program to create

CrayPat Application Program Interface (API)

- **Assume your code contains initialization and solution sections**
- **Want to analyze performance only of solution**
- **How to do this? Several approaches:**
  - Init section is only one routine (or just a few routines): eliminate it (or them) from the profile.
  - Init section is many routines: Use API to define a profile region that excludes init
  - What happens if some routines shared by init and solve? Use API to turn profiling on and off as needed

`Exercise in perftools/api`

Using the API

```
  include 'mpif.h'
  include 'pat_apif.h'
  ...
  call pat_record(PAT_STATE_OFF, ierr)
  call mpi_init(ierr)
  ...
! Main solver loop.

  call pat_record(PAT_STATE_ON, ierr)

  h = 1.0 / n

  do k=1,maxiter
    call pat_region_begin(1, 'compute_unew', ierr)
    ...
    call pat_region_end(1, ierr)

    call set_bc(unew,n,js,je)
    ...
    call compute_diff(u,unew,n,js,je,diffnorm)
    ...
    call pat_region_begin(2, 'set_u_to_unew', ierr)
    u(:,:) = unew(:,:)
    call pat_region_end(2, ierr)
    ...
  enddo
  call pat_record(PAT_STATE_OFF, ierr)
  ...
```

Arbitrary, user-assigned id's

```
> module load perftools
> ftn -c jacobi_mpi_api.f90
> ftn -o jacobi_mpi_api jacobi_mpi_api.o
> pat_build -g mpi -u jacobi_mpi_api
```

Run jacobi_mpi_api+pat in a batch job.

```
> pat_report
  jacobi_mpi_api+pat+15442-638t.xf
```

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

100.0% | 2.561574 |       -- |     -- |  122.0 |Total
|-----------------------------------------------------------------------
|  99.3% | 2.543540 |       -- |     -- |   42.0 |USER
||----------------------------------------------------------------------
||  50.8% | 1.300256 | 0.005706 |   0.4% |   10.0 |#1.compute_unew
||  23.4% | 0.599314 | 0.007033 |   1.2% |   10.0 |#2.set_u_to_unew
||  21.8% | 0.557499 | 0.000488 |   0.1% |   10.0 |compute_diff_
||   3.4% | 0.085854 | 0.000412 |   0.5% |    1.0 |MAIN_
||   0.0% | 0.000613 | 0.001408 |  71.1% |   10.0 |set_bc_
||   0.0% | 0.000004 | 0.000001 |  21.5% |    1.0 |main
||======================================================================
|   0.5% | 0.013464 | 0.012735 |  94.6% |   10.0 |MPI_SYNC
||----------------------------------------------------------------------
|   0.5% | 0.013464 | 0.012735 |  94.6% |   10.0 | mpi_allreduce_(sync)
||======================================================================
|   0.2% | 0.004570 |       -- |     -- |   70.0 |MPI
||----------------------------------------------------------------------
||   0.2% | 0.004133 | 0.005798 |  59.6% |   20.0 |MPI_SENDRECV
||   0.0% | 0.000405 | 0.000068 |  14.6% |   10.0 |MPI_ALLREDUCE
||   0.0% | 0.000026 | 0.000004 |  12.9% |   20.0 |MPI_COMM_SIZE
||   0.0% | 0.000007 | 0.000011 |  63.0% |   20.0 |mpi_comm_rank
|======================================================================
```


U.S. DEPARTMENT OF ENERGY | Office of Science

Hardware Performance Counters

- **Special registers available on the processor that count certain hardware related events**
- **Minimal overhead**
  - They're running all the time
  - Typically one clock period to read
- **Potentially rich source of performance information**

Types of Counters

- Cycles
- Instruction count
- Memory references, cache hits/misses
- Floating-point instructions
- Resource utilization

PAPI Event Counters

- **PAPI (Performance API) provides a standard interface for use of the performance counters in major microprocessors**

- **Predefined actual and derived counters supported on the system**
  - To see the list, run 'papi_avail' on compute node via aprun:

```
module load perftools
aprun –n 1 papi_avail
```

- **AMD native events also provided; use 'papi_native_avail':**

```
aprun –n 1 papi_native_avail
```

Hardware Performance Monitoring

- **Specify hardware counters to be monitored during sampling or tracing**
  - Default is "off"
  - Choose up to 4 events (if multiplexing is not used)

- **Can specify individual events:**

```
setenv PAT_RT_HWPC "PAPI_FP_OPS,PAPI_L1_DCM"
aprun –n … a.out+pat   (or a.out+apa)
```

- **Or predefined event group number (next slide):**

```
setenv PAT_RT_HWPC 1
aprun –n … a.out+pat   (or a.out+apa)
```

- **Enable multiplexing (monitoring more than 4 events) by setting PAT_RT_HWP_MPX to a nonzero value**

Exercise in perftools/hwpc

Predefined Counter Groups for PAT_RT_HWPC

0 Summary with instruction metrics
1 Summary with translation lookaside buffer (TLB) metrics
2 L1 and L2 cache metrics
3 Bandwith information
4 *** DO NOT USE, not supported on Quad-core or later AMD Opteron processors ***
5 Floating point mix
6 Cycles stalled, resources idle
7 Cycles stalled, resources full
8 Instructions and branches
9 Instruction cache
10 Cache hierarchy
11 Floating point operations mix (2)
12 Floating point operations mix (vectorization)
13 Floating point operations mix (single precision)
14 Floating point operations mix (double precision)
15 L3 cache (socket-level)
16 L3 cache, core-level reads (DO NOT USE; not currently supported)
17 L3 cache, core-level misses (DO NOT USE; not currently supported)
18 L3 cache, core-level fills caused by L2 evictions (DO NOT USE; not currently supported)
19 Prefetches

Hardware Performance Monitoring

PAT_RT_HWPC = 1

```
========================================================================
  USER / MAIN_
------------------------------------------------------------------------
  Time%                                          71.1%
  Time                                        7.720603 secs
  Imb. Time                                   0.000000 secs
  Imb. Time%                                      0.0%
  Calls                          0.130 /sec        1.0 calls
  PAPI_L1_DCM                    8.599M/sec    66391434 misses
  PAPI_TLB_DM                    1.177M/sec     9090793 misses
  PAPI_L1_DCA                  963.315M/sec  7437364182 refs
  PAPI_FP_OPS                 2147.898M/sec 16583040492 ops
  Average Time per Call                       7.720603 secs
  CrayPat Overhead : Time          0.0%
  User time (approx)             7.721 secs 16213256211 cycles  100.0% Time
  HW FP Ops / User time       2147.898M/sec 16583040492 ops   25.6%peak(DP)
  HW FP Ops / WCT             2147.898M/sec
  Computational intensity         1.02 ops/cycle    2.23 ops/ref
  MFLOPS (aggregate)           2147.90M/sec
  TLB utilization               818.12 refs/miss   1.598 avg uses
  D1 cache hit,miss ratios       99.1% hits         0.9% misses
  D1 cache utilization (misses) 112.02 refs/miss  14.003 avg hits
```

Measured

Derived

avg uses (or hits): How many times each word is used before the next miss

`pat_report –s data_size=4` … (because single precision was used)

cacheline: 64 Bytes
L1 cache: 64 KB, dedicated for each core
L2 cache: 512 KB, dedicated for each core
page: 4 KB (2 MB if huge page)

Hardware Performance Monitoring

CrayPat is making suggestions about how to improve code performance

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

D1 cache utilization:

    4.8% of total execution time was spent in 2 functions with D1 cache
    hit ratios below the desirable minimum of 90.0%. Cache utilization
    might be improved by modifying the alignnment or stride of
    references to data arrays in these functions.

        D1    Time%   Function
     cache
       hit
     ratio

       76.5%     0.2%  set_bc_
       89.9%     4.6%  init_fields_


TLB utilization:

    4.8% of total execution time was spent in 4 functions with fewer
    than the desirable minimum of 512 data references per TLB miss. TLB
    utilization might be improved by modifying the alignnment or stride
    of references to data arrays in these functions.

        LS    Time%   Function
       per
       TLB
        DM

        4.31     0.2%  set_bc_
      111.87     0.0%  main
      173.06     0.0%  read_params_
      273.26     4.6%  init_fields_

================  End Observations  ====================================
```

Hardware Performance Monitoring

PAT_RT_HWPC = 2

```
  USER / MAIN_
------------------------------------------------------------------------------
  Time%                                              70.3%
  Time                                            7.423632 secs
  Imb. Time                                       0.000000 secs
  Imb. Time%                                          0.0%
  Calls                           0.135 /sec           1.0 calls
  PAPI_L1_DCM                     7.889M/sec      58565651 misses
  PAPI_L1_DCA                  1001.024M/sec    7431227479 refs
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED       31.382M/sec     232971825 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                          76.822M/sec     570296612 fills
  Average Time per Call                            7.423632 secs
  CrayPat Overhead : Time          0.0%
  User time (approx)              7.424 secs   15589626660 cycles  100.0% Time
  D1 cache hit,miss ratios        99.2% hits          0.8% misses
  D1 cache hit,refill ratio       89.2% hits         10.8% refills
  D1 cache utilization (misses)  126.89 refs/miss   15.861 avg hits
  D1 cache utilization (refills)   9.25 refs/refill  1.156 avg uses
  D2 cache hit,miss ratio         29.0% hits         71.0% misses
  D1+D2 cache hit,miss ratio      99.4% hits          0.6% misses
  D1+D2 cache utilization        178.72 refs/miss   22.340 avg hits
  System to D1 refill            76.822M/sec     570296612 lines
  System to D1 bandwidth       4688.834MB/sec  36498983165 bytes
  D2 to D1 bandwidth           1915.435MB/sec  14910196796 bytes
```

Hardware Performance Monitoring

PAT_RT_HWPC = 5

```
  USER / MAIN_
----------------------------------------------------------------------
  Time%                                        70.3%
  Time                                      7.435543 secs
  Imb. Time                                 0.000000 secs
  Imb. Time%                                    0.0%
  Calls                       0.134 /sec         1.0 calls
  PAPI_FML_INS              248.649M/sec  1848836079 ops
  PAPI_FAD_INS              310.812M/sec  2311054900 ops
  PAPI_FDV_INS                0.134 /sec           1 ops
  RETIRED_MMX_AND_FP_INSTRUCTIONS:
    PACKED_SSE_AND_SSE2     993.163M/sec  7384702020 instr
  Average Time per Call                     7.435543 secs
  CrayPat Overhead : Time   0.0%
  User time (approx)          7.436 secs  15614648846 cycles  100.0% Time
  HW FP Ops / User time     559.461M/sec   4159890979 ops    6.7%peak(DP)
  HW FP Ops / WCT           559.461M/sec
  FP Multiply / FP Ops                         44.4%
  FP Add / FP Ops                              55.6%
  MFLOPS (aggregate)         559.46M/sec
```

Relative ratios for multiplies and adds

Don't use the MFLOPS number obtained with PAT_RT_HWPC=5; it's a underestimated value (2147.898 MFLOPS with PAT_RT_HWPC=0 or 1)

Hardware Performance Monitoring

PAT_RT_HWPC = 12

```
  USER / MAIN_
-----------------------------------------------------------------------------
  Time%                                          71.7%
  Time                                        7.975292 secs
  Imb. Time                                   0.000000 secs
  Imb. Time%                                      0.0%
  Calls                        0.125 /sec          1.0 calls
  RETIRED_SSE_OPERATIONS:
    SINGLE_ADD_SUB_OPS:
    SINGLE_MUL_OPS           520.908M/sec   4154398320 ops
  RETIRED_SSE_OPERATIONS:
    DOUBLE_ADD_SUB_OPS:
    DOUBLE_MUL_OPS             1.505 /sec           12 ops
  RETIRED_SSE_OPERATIONS:
    SINGLE_ADD_SUB_OPS:
    SINGLE_MUL_OPS:OP_TYPE  2079.301M/sec  16583040480 ops
  RETIRED_SSE_OPERATIONS:
    DOUBLE_ADD_SUB_OPS:
    DOUBLE_MUL_OPS:OP_TYPE     1.505 /sec           12 ops
  Average Time per Call                       7.975292 secs
  CrayPat Overhead : Time       0.0%
  User time (approx)           7.975 secs  16748133557 cycles  100.0% Time
```

Add and multiply instructions issued

Adds and multiplies performed

vector length (for sp) for 128-bit wide SSE2 vector operation

= 16,583,040,480 / 4,154,398,320 = 3.99

Compiled with 'ftn –fastsse …'

Guidelines to Identify the Need for Optimization

Derived metric	Optimization needed when*	PAT_RT_HWPC
Computational intensity	< 0.5 ops/ref	0, 1
L1 cache hit ratio	< 90%	0, 1, 2
L1 cache utilization (misses)	< 1 avg hit	0, 1, 2
L1+L2 cache hit ratio	< 92%	2
L1+L2 cache utilization (misses)	< 1 avg hit	2
TLB utilization	< 0.9 avg use	1
(FP Multiply / FP Ops) or (FP Add / FP Ops)	< 25%	5
Vectorization	< 1.5 for dp; 3 for sp	12 (13, 14)

* Suggested by Cray

avg uses (or hits): How many times each word is used until the next miss

Monitoring Network Performance Counters

- **Use PAT_RT_NWPC instead of PAT_RT_HWPC**

- **See 'Using the Cray Gemini Hardware Counters', S-0025-10**
  - http://docs.cray.com