More Profiling Tools at NERSC

Woo-Sun Yang
NERSC User Services Group

NUG Training
February 25, 2015
Overview

• To provide a quick start for profiling tools (other than VTune) on Edison
  – CrayPat
    • Reveal (not a profiling tool)
  – CrayPat-lite
  – IPM (not covered)
  – MAP
  – HPCToolkit
Performance analysis tools

• Measure code performance to identify performance bottlenecks and improve them

• Two types of measurement
  – Sampling
    • Sample where the program is executing (i.e., ‘program counter’) at regular time intervals (or certain events)
    • Low overhead
    • Useful in most cases for identifying performance hotspots
  – Tracing
    • Focus on the selected functions to see detailed info on their usage
    • User specifies a list of the functions to be traced
      – CrayPat’s APA (Automatic Program Analysis) suggests which functions to trace
    • Larger overhead, especially with functions which are called frequently

• Some tools available at NERSC
  – CrayPat: for sampling or tracing
  – IPM: for sampling
  – MAP: for sampling
  – HPCToolkit: for sampling
Workflow with CrayPat (Cray Performance Measurement and Analysis Tools)

1. ‘module load perftools’ **before** starting to build your code
2. Build your code; *.o must be kept as well as *.a, if any
3. Instrument your executable using ‘pat_build’
   - `pat_build [options] a.out`  # to create an instrumented binary,
     # a.out+pat
4. Execute your instrumented program
   - `aprun ... ./a.out+pat`  # in a batch job
   - Collected data saved in `a.out+pat+####-e.xf`
   (e: s for sampling and t for tracing)
5. Analyze the resulting data
   - `pat_report a.out+pat+####-e.xf`
CrayPat run types

• **Instrumentation types**
  
  – Sampling
  
  – Tracing: Specify a list of the functions to be traced
    
    • User functions: using pat_build’s `-T,-t, -u` (-u for all; can increase run time significantly)
    
    • Preset trace groups for popular functions: using pat_build’s `-g`
      
      – mpi, heap, io, omp, blas, lapack, ...

• **Sampling run traces MPI functions, some system functions, etc. by default**

• **Sampling run automatically generates a .apa file that contains pat_build flags to trace suggested functions and function groups for a tracing run**
Sampling with CrayPat

$ module rm darshan
$ module load perftools
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x
$ pat_build -f bgw.x
$ cat runit
...
aprun -n 1 ./bgw.x+pat
$ qsub runit
2446538.edique02
$ pat_report bgw.x+pat+48499-6113s.xf > my.rpt
$ more my.rpt
$ app2 bgw.x+pat+48499-6113s.ap2
$ rm bgw.x+pat+48499-6113s.xf

$ ls -l *.apa
-rw------- 1 wyang wyang 1799 Feb 24 14:22 bgw.x+pat+48499-6113s.apa

Unload darshan as it will interfere with perftools

Build an instr. binary; -f to overwrite if there is one already

Use the instr. binary

ASCII text report captured in my.rpt

See the report

Visualization of the results using a GUI tool, app2
Not needed as you now have a .ap2 file;
*.ap2 is self-contained and portable while .xf is not;
text report can be generated from .ap2, too

This text file contains pat_build options to generate an instrumented executable for a tracing run with suggested list of trace functions and function groups; see the next slide
Tracing with CrayPat (one way - using Automatic Program Analysis)

$ module rm darshan
$ module load perftools
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x

$ vi bgw.x+pat+48499-6113s.apa
$ pat_build -f -O bgw.x+pat+48499-6113s.apa

$ cat runit
...
#export PAT_RT_SUMMARY=0
  aprun -n 1 ./bgw.x+apa
$ qsub runit
2447437.edique02

$ pat_report bgw.x+apa+32565-3822t.xf > myt.rpt
$ more myt.rpt
$ app2 bgw.x+apa+32565-3822t.ap2
$ rm *.*xf

Unload darshan as it will interfere with perftools
Edit suggested trace functions/groups, if you want
Build a new instr. binary for tracing, guided by the sampling results
For more detailed data; data size can become huge
Use the new instr. binary for tracing
ASCII text report in myt.rpt
If you want...
Not needed as you now have .ap2 files
CrayPat results in the text report

```
% more my.rpt
...
```

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>Samp</td>
<td>Samp%</td>
<td></td>
<td></td>
</tr>
<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
<th>Function</th>
<th>Source</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIN__</td>
<td></td>
<td>scratchdirs/wyang/BGW/bgw.f90</td>
<td></td>
</tr>
</tbody>
</table>

100.0% | 5714.0 | -- | -- | Total

98.7% | 5639.0 | -- | -- | USER

98.7% | 5639.0 | -- | -- | MAIN__

3

4 | 2.6% | 146.0 | -- | -- | line.71
4 | 2.5% | 143.0 | -- | -- | line.73
4 | 17.4% | 996.0 | -- | -- | line.177
4 | 14.0% | 799.0 | -- | -- | line.178
4 | 2.7% | 153.0 | -- | -- | line.180
4 | 55.7% | 3183.0 | -- | -- | line.181
4 | 1.2% | 66.0 | -- | -- | line.211
4 | 1.1% | 60.0 | -- | -- | line.212

4

4

1.3% | 73.0 | -- | -- | ETC

1.2% | 71.0 | -- | -- | __svml_log4_e9

From a sampling run

Default sampling interval: 10,000 microsec or 0.01 sec
CrayPat results for a trace run: “Observations and suggestions” section

```bash
$ cat myt.rpt
...
```

### D1 cache utilization:

100.0% of total execution time was spent in 2 functions with D1 cache hit ratios below the desirable minimum of 75.0%. Cache utilization might be improved by modifying the alignment or stride of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>D1</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56.2%</td>
<td>0.0%</td>
<td>main</td>
</tr>
<tr>
<td>70.6%</td>
<td>100.0%</td>
<td>MAIN__</td>
</tr>
</tbody>
</table>

### D1 + D2 cache utilization:

100.0% of total execution time was spent in 1 functions with combined D1 and D2 cache hit ratios below the desirable minimum of 85.0%. Cache utilization might be improved by modifying the alignment or stride of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>D1+D2</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>84.8%</td>
<td>100.0%</td>
<td>MAIN__</td>
</tr>
</tbody>
</table>

### TLB utilization:

100.0% of total execution time was spent in 2 functions with fewer than the desirable minimum of 200 data references per TLB miss. TLB utilization might be improved by modifying the alignment or stride of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>LS per</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB DM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>166.59</td>
<td>0.0%</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAIN__</td>
</tr>
</tbody>
</table>
CrayPat results in the text report (2)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Time</td>
<td>54.383497 secs</td>
</tr>
<tr>
<td>Imb. Time%</td>
<td>--</td>
</tr>
<tr>
<td>Calls</td>
<td>0.018 /sec</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED:THREAD_P</td>
<td>165268349017</td>
</tr>
<tr>
<td>CPU_CLK_UNHALTED:REF_P</td>
<td>5165651720</td>
</tr>
<tr>
<td>DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK</td>
<td>220360299</td>
</tr>
<tr>
<td>DTLB_STORE_MISSES:MISS_CAUSES_A_WALK</td>
<td>51001827</td>
</tr>
<tr>
<td>L1d:REPLACEMENT</td>
<td>14217002486</td>
</tr>
<tr>
<td>L2_RQSTS:ALL_DEMAND_DATA_RD</td>
<td>11410446231</td>
</tr>
<tr>
<td>L2_RQSTS:DEMAND_DATA_RD_HIT</td>
<td>4058248212</td>
</tr>
<tr>
<td>MEM_UOPS_RETIRED:ALL_LOADS</td>
<td>48362710284</td>
</tr>
<tr>
<td>FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE</td>
<td>725400</td>
</tr>
<tr>
<td>FP_COMP_OPS_EXE:X87</td>
<td>1588974370</td>
</tr>
<tr>
<td>FP_COMP_OPS_EXE:SSE_FP_PACKED_DOUBLE</td>
<td>209746182</td>
</tr>
<tr>
<td>SIMD_FP_256:PACKED_SINGLE</td>
<td>34072888</td>
</tr>
<tr>
<td>SIMD_FP_256:PACKED_DOUBLE</td>
<td>51986600374</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>54.384 secs</td>
</tr>
<tr>
<td>CPU_CLK</td>
<td>3.199GHz</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>3865.660M/sec</td>
</tr>
<tr>
<td>Total SP ops</td>
<td>5.012M/sec</td>
</tr>
<tr>
<td>Total DP ops</td>
<td>3860.648M/sec</td>
</tr>
<tr>
<td>Computational intensity</td>
<td>1.61 ops/cycle</td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
<td>3865.66M/sec</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>178.22 refs/miss</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>70.6% hits</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>3.40 refs/miss</td>
</tr>
<tr>
<td>D2 cache hit,miss ratio</td>
<td>48.3% hits</td>
</tr>
<tr>
<td>D1+D2 cache hit,miss ratio</td>
<td>84.8% hits</td>
</tr>
<tr>
<td>D1+D2 cache utilization</td>
<td>6.58 refs/miss</td>
</tr>
<tr>
<td>D2 to D1 bandwidth</td>
<td>12806.058MiB/sec</td>
</tr>
<tr>
<td>Average Time per call</td>
<td>54.383497 secs</td>
</tr>
</tbody>
</table>

From a tracing run

Using the default performance counter group value (2) for PAT_RT_PERFCTR (among available values: 0, 1, 2, 6, 7,8, 9, 10, 11, 12, 13, 14 & 19)

SSE scalar double precision (a)
Single or double precision? (b)
SSE vector (128-bit wide) double precision (c)
AVX vector (256-bit wide) single precision (d)
AVX vector (256-bit wide) double precision (e)

Average fp vector length
≈ \[ \frac{1*(a)+1*(b)+2*(c)+8*(d)+4*(e)}{(a)+(b)+...+(e)} \]
= 3.906

Derived from the above numbers

See http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops
CrayPat .ap2 displayed with app2

$ app2 bgw.x+apa+32565-38225t.ap2
More things to do with CrayPat…

• **Automatic Rank Order Analysis**
  – Suggests a better MPI rank placement

• **CrayPat API**
  – Instrument and get tracing results only for selected regions of your code

• **Monitor a selected group of hardware counters (floating point operations, cache usage, etc.) or network performance counters**

• **For more info:**
  – Pat_help online help systems
    $ pat_help
  – (Old) CrayPat tutorial materials in the directory on NERSC machines:
    /project/projectdirs/training/NUG2012/perftools
  – ‘Using the PAPI Cray NPU Component’, [http://docs.cray.com/books/S-0046-10/S-0046-10.pdf](http://docs.cray.com/books/S-0046-10/S-0046-10.pdf)
CrayPat-lite

- A simplified version of CrayPat
  - No need for you to manually build an instrumented binary
  - *.ap2, *.rpt (text report) files are generated for you

```bash
$ module rm darshan
$ module load perftools-lite
$ export CRAYPAT_LITE=sample_profile
$ export CRAYPAT_LITE=event_profile
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x

$ cat runit
...
aprun -n 1 ./bgw.x
$ qsub runit
2448485.edique02

$ more runit.o2448485
$ more bgw.x+40813-5014s.rpt
$ app2 bgw.x+40813-5014s.ap2

Unload darshan as it will interfere with perftools
For sampling (default)
For tracing
‘bgw.x’ is an instr. binary
Performance summary included in stdout file
Same text report saved in bgw.x+*.rpt
If you want...
```
Cray Reveal

- Identifies potential loops for OpenMP parallelization
  - Based on analysis of CrayPat’s performance run
- Provides OpenMP directives for such loops
- Works only with Cray compiler at this time
- Additionally, it displays loopmark information in GUI, generated by Cray compiler (i.e., what kind of optimization is done to a loop, whether it is vectorized, how many times it is unrolled, etc.)
- See
  https://www.nersc.gov/users/software/debugging-and-profiling/craypat/reveal/
Workflow with Reveal

$ module rm darshan
$ module swap PrgEnv-intel PrgEnv-cray  
$ module load perftools

(1) Generate loop work estimates

$ ftn -c -O3 -h profile_generate bgw.f90
$ ftn -O3 bgw.o -h profile_generate -o bgw.x
$ pat_build -f -w bgw.x
$ cat runit
...
aprun -n 1 ./bgw.x+mat
$ qsub runit
2450557.edique02
$ pat_report bgw.x+pat+38560-5949t.xf > my.rpt

(2) Generate a “program library”

$ ftn -c -h pl=bgw.pl -O3 bgw.f90
$ ls -ld bgw.pl
drwx------- 2 wyang wyang 4096 Feb 24 22:48 bgw.pl

(3) Run Reveal to identify potential loops that can be turned into OpenMP parallel regions

$ reveal bgw.pl bgw.x+pat+38560-5949t.ap2

Works only under PrgEnv-cray
Build an instr. binary for tracing
Get performance data
Repeat for all source files

(1) Generate loop work estimates
(2) Generate a “program library”
(3) Run Reveal to identify potential loops that can be turned into OpenMP parallel regions
Reveal: scope a loop

```
176  schDtt = (0D0, 0D0)
177  do ig = 1, igmax
        I_epsRggp_int = I_epsR_array(ig, my_igp, ifreq)
        I_epsAggp_int = I_epsA_array(ig, my_igp, ifreq)
        schD = I_epsRggp_int - I_epsAggp_int
        schDtt = schDtt + matngmatmgpD(ig, my_igp)*schD
        enddo
        schdt_array(ifreq) = schdt_array(ifreq) + schDtt
        enddo
184  time_b = time_b + endtime_ch - starttime_ch
```

Info - Line 177
- A loop starting at line 177 was unrolled 3 times.
- A loop starting at line 177 was vectorized.
- The loop is flat.
Get an OpenMP directive
Add the directive to your code
• Allinea’s parallel profiling tool with GUI: sampling
• MAP licenses for ~ 512 MPI tasks
  – Shared by other users and among all machines
• Need to build two small libraries for sampling, MAP sampler and MPI wrapper libraries
  – make-profile-libraries
  – Need to follow a certain linking order
    • See the user manual for details
    • Usually OK if you follow the instructions printed when running the above commands
• For info:
  – $ALLINEA_TOOLS_DOCDIR/userguide.pdf (after loading the allineatools module)
  – https://www.nersc.gov/users/software/debugging-and-profiling/MAP/
Using MAP

$ module load allineatools/5.0-40932
$ make-profiler-libraries --lib-type=static  
  Build the 2 static libs that MAP needs
$ ftn -c -g -O3 -xAVX -openmp bgw.f90  
  Build using the MAP-generated option file
$ ftn -O3 -xAVX -openmp bgw.o -Wl,@./allinea-profiler.1d -o bgw.x

$ qsub -I -v DISPLAY -lmppwidth=24
$ cd $PBS_O_WORKDIR
$ module load allineatools/5.0-40932  
  Run with MAP
$ map ./bgw.x
$ ls -ltr
  Profiling results saved in a file
...  
-rw------- 1 wyang wyang 90885 Feb 25 00:59 bgw_1p_2015-02-25_00-58.map
MAP results
• Supports MPI and threading (OpenMP, pthreads)
• Sampling based
• Assembles performance measurements into a call path profile that associates the costs of each function call with its full calling context
• Your program should call MPI_Comm_rank() with MPI_COMM_WORLD preferably right after MPI_Init(), to let hpcrun know these MPI ranks
**HPCToolkit Workflow**

1. **Build your code**
   - Compile code with –g
   - For statically linked binary, run hpclink

2. **Run hpcstruct on your executable to analyze code structure**

3. **Run your code with hpcrun**
   Measurement data in hpctoolkit-*app*-measurements-$PBS_JOBID$

4. **Run hpcprof/hpcprof-mpi to correlate collected data with the code structure**
   Data base in hpctoolkit-*app*-database-$PBS_JOBID$

5. **View the result with hpcviewer or hpctraceviewer**
Events to be sampled

• Sample sources:
  – -e e1@p1 -e e2@p2 ...
  where e1, e2,... are events and p1, p2,... are sampling periods
  – Example: -e PAPI_TOT_CYC@15000000 -e PAPI_L2_TCM@400000
  – Alternatively, use the environment variable

```bash
$ export HPRUN_EVENT_LIST="PAPI_TOT_CYC@15000000;PAPI_L2_TCM@400000"
```

• PAPI event must be both available and not derived
  – Aim for a rate for approx. a few hundred samples per second
    • Several million or tens of millions for PAPI_TOT_CYC
    • A few hundred thousand for cache misses

• Proxy sampling for derived PAPI events for events that cannot trigger interrupts directly
  – For example, PAPI_FP_OPS on Intel CPUs
  – Sampling period not specified: e.g., ‘-e PAPI_FP_OPS’
% module load hpctoolkit
% ftn -c -g -openmp jacobi_mpiomp.f90
% hpclink ftn -openmp -o jacobi_mpiomp jacobi_mpiomp.o  
Analyze code in order to map collected data; will create jacobi_mpiomp.hpcstruct
# ftn -openmp -dynamic -o jacobi_mpiomp jacobi_mpiomp.o
Statically-linked binary
Dynamically-linked binary

% hpcstruct jacobi_mpiomp
Directory for measurement data

% cat runit

...  
export OMP_NUM_THREADS=12
Directory for database
mdir=res_m.$PBS_JOBID
ddir=res_d.$PBS_JOBID
aprun -n 4 -N 2 -S 1 -d 12 hpcrun -e PAPI_L2_TCM@10000 ... -o $mdir .
jadobi_mpiomp
hpcrun -d 12 hpcprof-mpi -S jacobi_mpiomp.hpcstruct -I $PWD/*' 
-o $ddir $mdir

% qsub runit
2442996.edique02
% hpcviewer res_d.2442996.edique02
View the results
!$omp parallel do private(utmp)
  do j=jsl,je1
    utmp = 0.25 * ( u(i+1,j) + u(i-1,j) )
    + u(i,j+1) + u(i,j-1) &
    + h^2 + f(i,j) )
    unew(i,j) = omega * utmp + ( 1. - omega ) * u(i,j)
  enddo
  enddo
!$omp end parallel do

  call set_bc(unew,n,js,je)
National Energy Research Scientific Computing Center