Using Craypat & Reveal on Cori

Rebecca Hartman-Baker
User Engagement Group Leader

February 23, 2017
Outline

I. Profiling with Craypat
II. Using Reveal for OpenMP
I. PROFILING WITH CRAYPAT

Profile of Cochise in the Chiricahua Mountains by Ken Bosma, http://www.flickr.com/photos/kretyen/2879059366/
I. Profiling with Craypat

• Introduction
• Simple profiling
• Full-service profiling
Introduction

• Craypat is Cray’s Performance Analysis Tool
• Evaluate program behavior on Cray supercomputer
  – Under any PrgEnv
• Find hotspots, load imbalance, inefficiencies
  – I/O, memory usage
  – MPI communications
  – Flops
  – Recommendation for rank reordering (sometimes)
• Profiler with limited tracing abilities
  – Tracing tools with better performance: MAP, VampirTrace
Simple Profiling with CrayPat

- `perftools-lite` module easier to use & does (almost) everything in `perftools`
- Compile code with `perftools-lite` module loaded
- Run code as normal
- Output:
  - Stdout & `*.rpt` file: report with execution time, memory high-water mark, aggregate FLOPS rate, top time-consuming user functions, MPI info, etc.
  - `*.ap2` file: can be viewed with Apprentice 2
  - (Possibly) `MPICH_RANK_REORDER` file
CrayPat/X: Version 6.4.0 Revision bc8f5bd 05/24/16 17:52:13
Experiment: lite lite/sample_profile
Number of PEs (MPI ranks): 64
Numbers of PEs per Node: 64
Numbers of Threads per PE: 1
Number of Cores per Socket: 68
Execution start time: Thu Oct 13 09:30:31 2016
System name and speed: nid04403 1401 MHz (approx)
Intel knl CPU Family: 6 Model: 87 Stepping: 1
MCDRAM: 7.2 GHz, 16 GiB available as quad, flat (0% cache)

Avg Process Time: 558.16 secs
High Memory: 1,899.7 MBytes 29.7 MBytes per PE
I/O Read Rate: 4.032070 MBytes/sec
I/O Write Rate: 3.618872 MBytes/sec
### Table 1: Profile by Function Group and Function (top 10 functions shown)

<table>
<thead>
<tr>
<th>Group</th>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>100.0%</td>
<td>55,700.3</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td>49.3%</td>
<td>27,466.0</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
</tr>
<tr>
<td></td>
<td>15.6%</td>
<td>8,679.3</td>
<td>1,276.7</td>
<td>13.0%</td>
<td>__cray_HCOSS_01</td>
</tr>
<tr>
<td></td>
<td>12.2%</td>
<td>6,821.2</td>
<td>1,167.8</td>
<td>14.8%</td>
<td>__cray_COS_V_01</td>
</tr>
<tr>
<td></td>
<td>8.9%</td>
<td>4,948.8</td>
<td>581.2</td>
<td>10.7%</td>
<td>__COS_Z</td>
</tr>
<tr>
<td></td>
<td>2.3%</td>
<td>1,285.3</td>
<td>335.7</td>
<td>21.0%</td>
<td>gotoblas_daxpy_k_knl</td>
</tr>
<tr>
<td></td>
<td>1.9%</td>
<td>1,071.1</td>
<td>235.9</td>
<td>18.3%</td>
<td>gotoblas_blas_memory_alloc_knl</td>
</tr>
<tr>
<td></td>
<td>1.9%</td>
<td>1,039.9</td>
<td>185.1</td>
<td>15.3%</td>
<td>gotoblas_dger_k_knl</td>
</tr>
<tr>
<td>USER</td>
<td>35.5%</td>
<td>19,747.6</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td>23.0%</td>
<td>12,803.4</td>
<td>1,592.6</td>
<td>11.2%</td>
<td>intgrd_</td>
</tr>
<tr>
<td></td>
<td>7.6%</td>
<td>4,229.4</td>
<td>1,033.6</td>
<td>20.0%</td>
<td>dfshre_</td>
</tr>
<tr>
<td></td>
<td>3.1%</td>
<td>1,707.7</td>
<td>501.3</td>
<td>23.1%</td>
<td>drlhre_</td>
</tr>
<tr>
<td>MPI</td>
<td>14.7%</td>
<td>8,169.4</td>
<td>21,597.6</td>
<td>73.7%</td>
<td>MPI_RECV</td>
</tr>
<tr>
<td></td>
<td>14.7%</td>
<td>8,169.4</td>
<td>21,597.6</td>
<td>73.7%</td>
<td>MPI_Recv</td>
</tr>
</tbody>
</table>
Simple Profiling Recipe

- Load/unload modules:
  - `module unload darshan`
  - `module load perftools-base perftools-lite`
- Compile and run your code as usual
Full-Service Profiling

- **Motivation:**
  - Need more info than perftools-lite provides
  - Want to ignore certain subroutines
  - Focus on particular class of functions
  - Tracing rather than profiling

- **Super-deluxe profiling recipe**

- **pat_build options**
Super-Deluxe Profiling Recipe (1)

• **Load/unload modules:**
  
  ```
  module unload darshan
  module load perftools-base perftools
  ```

• **Compile code as usual, making sure to preserve object files**

• **pat_build -O apa myapp**
  
  ```
  Generates executable called myapp+pat
  ```

• **Run myapp+pat**
  
  ```
  Results in output file with name like
  myapp+pat+#####.xf or directory called
  myapp+pat+#####
  ```
Super-Deluxe Profiling Recipe (2)

- `pat_report myapp+pat+*.xf`
  - Generates `myapp+pat+*.apa`
- `pat_build -O myapp+pat+*.apa`
  - Generates executable called `myapp+apa`
- Run `myapp+apa`
- `pat_report myapp+apa+*.xf`
pat_build Options

• `pat_build -O apa myapp`
  – Craypat output for `myapp+pat` will be sampling to determine which subroutines can be ignored in full run. Additional file, `*.apa`, produced from `pat_report`
  – After this run, execute `pat_build -O *.apa` file to re-instrument `myapp+pat` into `myapp+apa` and run `myapp+apa` to get performance info

• `pat_build -g tracegroup myapp`
  – `tracegroup` is group of functions that can be automatically traced by CrayPat. Options include: `blas, fftw, mpi, netcdf, petsc`

• `pat_build -w myapp`
  – Do tracing experiment instead of profiling
II. PARALLELIZATION WITH CRAY REVEAL

“Happiness Revealed,” by Leonard Farshore, [https://flic.kr/p/9z7isd](https://flic.kr/p/9z7isd)
Cray Reveal

• Tool for porting to shared-memory or offload programming models
• Combine profiling info from Craypat and Cray compiler annotation to determine where to place OpenMP directives (generated automatically)

• Works ONLY with Cray programming environment
Using Cray Reveal

1. Compile code with Craypat instrumentation and create program library
2. Run representative job
3. Run Reveal
4. Insert directives, consider loop reordering, and analyze performance from optimizations
Cray Reveal Recipe (1)

• Load/unload modules:
  – module unload darshan
  – module swap PrgEnv-intel PrgEnv-cray
  – module load perftools-base perftools

• Compile & link with
  – \( -h\) profile_generate (to instrument), and
  – \( -h\) pl=/directory/path/myapp.pl (for compiler feedback)

• Instrument binary for tracing:
  – pat_build -w ./myapp
  – Creates instrumented application: myapp+pat
Cray Reveal Recipe (2)

- Run instrumented application (myapp+pat) as normal
  - Ideally this is job requiring 5-15 mins runtime, performing important subroutines in similar proportions to typical run

- This creates file called
  myapp+pat+####-##t.xf (or directory myapp+pat+####-##t/ for large runs)

- Create report with loop statistics
  - pat_report myapp+pat+* > loops_report.txt
  - Generates .ap2 file & generates text report in output file
Cray Reveal Recipe (3)

- Run Reveal
  - `reveal /directory/path/myapp.pl` (compiler info only)
  - `reveal /directory/path/myapp.pl myapp+#######+##t.ap2` (compiler + profiler info)
Opening Screen

New to Reveal?
Try "Getting Started" in the "Help" Menu
Scoping Window
Compiler Annotations & Explanations

- A loop starting at line 133 was not vectorized because it contains a call to function `funsun` on line 135.
- A loop starting at line 144 was not vectorized because it contains a call to a subroutine or function on line 152.
- A loop starting at line 152 was partially vectorized with a single vector iteration.
Partial Success in Subroutine
Successful Scoping
Directives Generated by Reveal

OpenMP Directive

! Directive inserted by Cray Reveal. May be incomplete.
!$OMP parallel do default(none)
!$OMP& private (i)
!$OMP& shared (ndim, center, hwidth, x, g)
Unsuccessful Scoping
Unsuccessful Scoping Directive

! Directive inserted by Cray Reveal. May be incomplete.
!$OMP parallel do default(none)
!$OMP& unresolved (ndim,numfun,values,errors,centrs,hwidts,greate,dir,
!$OMP& work,center,hwidth,subrgn@dtrhre_,value,error)
!$OMP& private (i,index)
!$OMP& shared (ndiv,sbrgns)
Cray Reveal Recipe (4)

• Insert directives

• Examine compiler feedback to determine potential loop reordering
  – E.g., row- vs. column-ordered memory access patterns
  – Moving conditionals outside of loops
  – Cray compiler good at loop optimizations but requires some human help at times

• Analyze performance after optimizations
  – (Lather, rinse, repeat)
National Energy Research Scientific Computing Center