Debugging and Optimization Tools

Richard Gerber
NERSC
User Services Deputy Group Lead
Outline

• Take-Aways
• Debugging
• Performance / Optimization
• NERSC “automatic” tools

Videos, presentations, and references:

http://www.nersc.gov/users/training/courses/CS267/

Also see the DOE Advanced Computational Tools:
http://acts.nersc.gov
Take-Aways

• Tools can help you find errors in your program and locate performance bottlenecks
• In the world of HPC parallel computing, there are few widely adopted standard tools
  – Totalview and DDT debuggers
  – PAPI, Tau, & vendor-specific performance tools
• Common code problems
• How tools work in general
• Use the tools that works for you and are appropriate for your problem
• Be suspicious of outliers among parallel tasks
• Where to get more information
What is a Bug?

• A bug is when your code
  
  crashes
  – hangs (doesn’t finish)
  – gets inconsistent answers
  – produces wrong answers
  – behaves in any way you didn’t want it to

The term “bug” was popularized by Grace Hopper (motivated by the removal of an actual moth from a computer relay in 1947)
Common Causes of Bugs

• “Serial” (Sequential might be a better word)
  – Invalid memory references
  – Array reference out of bounds
  – Divide by zero
  – Use of uninitialized variables

• Parallel  Let’s concentrate on these
  – Unmatched sends/receives
  – Blocking receive before corresponding send
  – Out of order collectives
  – Race conditions
  – Unintentionally modifying shared memory structures
What to Do if You Have a Bug?

• **Find It**
  – You want to locate the part of your code that isn’t doing what it’s designed to do

• **Fix It**
  – Figure out how to solve it and implement a solution

• **Check It**
  – Run it to check for proper behavior
Find It: Tools

• **printf, write**
  – Versatile, sometimes useful
  – Doesn’t scale well
  – Not interactive
  – Fishing expedition

• **Compiler / Runtime**
  – Bounds checking, exception handling
  – Dereferencing of NULL pointers
  – Function and subroutine interface checking

• **Serial gdb + friends**
  – GNU debugger, serial, command-line interface
  – See “man gdb”

• **Parallel debuggers**
  – DDT
  – Totalview
  – Intel Inspector

• **Memory debuggers**
  – MAP
  – Valgrind

See NERSC web site
https://www.nersc.gov/users/software/debugging-and-profiling/
Parallel Programming Bug

This code hangs because both Task 0 and Task N-1 are blocking on MPI_Recv

```c
if(task_no == 0) {
    ret = MPI_Recv(&herBuffer, 50, MPI_DOUBLE, totTasks-1, 0, MPI_COMM_WORLD, &status);
    ret = MPI_Send(&myBuffer, 50, MPI_DOUBLE, totTasks-1, 0, MPI_COMM_WORLD);
}
else if (task_no == (totTasks-1)) {
    ret = MPI_Recv(&herBuffer, 50, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
    ret = MPI_Send(&myBuffer, 50, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
}
```
Compile & Start DDT

Compile for debugging

```
hopper% make
cc -c -g hello.c
cc -o hello -g hello.o
```

Set up the parallel run environment

```
hopper% qsub -I -V -lmppwidth=24
hopper% cd $PBS_O_WORKDIR
```

Start the DDT debugger

```
hopper% module load ddt
hopper% ddt ./hello
```
Press Go and then Pause when code appears hung.

Task 0 is at line 44

At hang, tasks are in 3 different places.
What About Massive Parallelism?

• With 10K+ tasks/threads/streams it’s impossible to examine every parallel instance
• Make us of statistics and summaries
• Look for tasks that are doing something different
  – Amount of memory used
  – Number of calculations performed (from counters)
  – Number of MPI calls
  – Wall time used
  – Time spent in I/O
  – One or a few tasks paused at a different line of code
• We (NERSC) have been advocating for this statistical view for some time
Vendors are starting to listen (DDT)
DDT video

- http://vimeo.com/19978486

- Or http://vimeo.com/user5729706

- Linked to from http://www.nersc.gov/users/training/courses/CS267/
Performance / Optimization
Performance Questions

• How can we tell if a program is performing well? Or isn’t? What is “good”?

• If performance is not “good,” can we identify the causes?

• What can we do about it?
Is Your Code Performing Well?

• No single answer, but
  – Does is scale well?
  – Is MPI time <20% of total run time?
  – Is I/O time <10% of total run time?
  – Is it load balanced?
  – If GPU code, does GPU+Processor perform better than 2 Processors?

• “Theoretical” CPU performance vs. “Real World” performance in a highly parallel environment
  – Cache-based x86 processors: >10% is pretty good
  – GPUs: >1% pretty good
What can we do about it

• Minimize latency effects (aggregate messages)
• Maximize work vs. communication
• Minimize data movement (recalculate vs. send)
• Use the “most local” memory
• Use large-block I/O
• Use a balanced strategy for I/O
  – Avoid “too many” tasks accessing a single file, but “too many” files performs poorly ~1000s
  – Use “enough” I/O tasks to maximum I/O bandwidth, but “too many” causes contention 1/node
Can We Identify the Causes? Use Tools

• **Vendor Tools:**
  – CrayPat on Crays
  – INTEL VTune

• **Community Tools:**
  – TAU (U. Oregon via ACTS)
  – PAPI (Performance API)
  – gprof

• **NERSC “automatic” and/or easy-to-use tools**
  – e.g. IPM

See NERSC web site
https://www.nersc.gov/users/software/debugging-and-profiling/
Example: CrayPat

• Suite of tools that provides a wide range of performance-related information

• Can be used for both sampling and tracing
  – with or without hardware or network performance counters
  – Built on PAPI

• Supports Fortran, C, C++, UPC, MPI, Coarray Fortran, OpenMP, Pthreads, SHMEM

• Man pages
  – intro_craypat(1), intro_app2(1), intro_papi(1)
Using CrayPat

1. **Access the tools**
   - module load perftools

2. **Build your application; keep .o files**
   - 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)

5. **Run pat_report on that new file; view results**
   - pat_report  XXXXX.xf  >  my_profile
   - view my_profile
   - Also produces a new file: XXXXX.ap2
Tools for the Masses

• Using even the best tools can be tedious
  – “Follow these 10 steps to perform the basic analysis of your program” – from a supercomputer center web site for a well-known tool

• NERSC wants to enable easy access to information that can help you improve your parallel code
  – automatic data collection
  – provide useful tools through the web

• Efforts
  – IPM (MPI profiling, chip HW counters, memory used)
  – Accounting & UNIX resource usage
  – System-level I/O monitoring
  – User-level I/O profiling (Darshan)
### IPM

<table>
<thead>
<tr>
<th># host</th>
<th>s05601/006035314C00_AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_tasks</td>
<td>32 on 2 nodes</td>
</tr>
<tr>
<td>start</td>
<td>11/30/04/14:35:34</td>
</tr>
<tr>
<td>wallclock</td>
<td>29.975184 sec</td>
</tr>
<tr>
<td>stop</td>
<td>11/30/04/14:36:00</td>
</tr>
<tr>
<td>%comm</td>
<td>27.72</td>
</tr>
<tr>
<td>gbytes</td>
<td>6.65863e-01 total</td>
</tr>
<tr>
<td>gflop/sec</td>
<td>2.33478e+00 total</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># total</th>
<th>[avg]</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>wallclock</td>
<td>953.272</td>
<td>29.7897</td>
<td>29.6092</td>
</tr>
<tr>
<td>user</td>
<td>837.25</td>
<td>26.1641</td>
<td>25.71</td>
</tr>
<tr>
<td>system</td>
<td>60.6</td>
<td>1.89375</td>
<td>1.52</td>
</tr>
<tr>
<td>mpi</td>
<td>264.267</td>
<td>8.25834</td>
<td>7.73025</td>
</tr>
<tr>
<td>%comm</td>
<td>27.7234</td>
<td>25.8873</td>
<td>29.3705</td>
</tr>
<tr>
<td>gbytes</td>
<td>0.665863</td>
<td>0.0729619</td>
<td>0.072204</td>
</tr>
<tr>
<td>gflop/sec</td>
<td>2.33478</td>
<td>0.0729619</td>
<td>0.072204</td>
</tr>
<tr>
<td>PM_FPU0_CMPL</td>
<td>2.28827e+10</td>
<td>7.15084e+08</td>
<td>7.07373e+08</td>
</tr>
<tr>
<td>PM_FPU1_CMPL</td>
<td>1.70657e+10</td>
<td>5.33304e+08</td>
<td>5.28487e+08</td>
</tr>
<tr>
<td>PM_FPU_FMA</td>
<td>3.00371e+10</td>
<td>9.38666e+08</td>
<td>9.27762e+08</td>
</tr>
<tr>
<td>PM_INST_CMPL</td>
<td>2.78819e+11</td>
<td>8.71309e+09</td>
<td>8.20981e+09</td>
</tr>
<tr>
<td>PM_LD_CMPL</td>
<td>1.25478e+11</td>
<td>3.92118e+09</td>
<td>3.74541e+09</td>
</tr>
<tr>
<td>PM_ST_CMPL</td>
<td>7.45961e+10</td>
<td>2.33113e+09</td>
<td>2.21164e+09</td>
</tr>
<tr>
<td>PM_TLB_MISS</td>
<td>2.45894e+08</td>
<td>7.68418e+06</td>
<td>6.98733e+06</td>
</tr>
<tr>
<td>PM_CYC</td>
<td>3.0575e+11</td>
<td>9.55467e+09</td>
<td>9.36585e+09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># time</th>
<th>[calls]</th>
<th>&lt;mpi&gt;</th>
<th>&lt;wall&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send</td>
<td>188.386</td>
<td>639616</td>
<td>71.29</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>69.5032</td>
<td>639616</td>
<td>26.30</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>6.34936</td>
<td>639616</td>
<td>2.40</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0.0177442</td>
<td>32</td>
<td>0.01</td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>0.00540609</td>
<td>32</td>
<td>0.00</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>0.00465156</td>
<td>32</td>
<td>0.00</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>0.000145341</td>
<td>32</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Completed Jobs on NERSC Web Site

NERSC job details


<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>CV (%)</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate Floating Point Operations (Flop x 10^{10})</td>
<td>3.011e+02</td>
<td>1.470e-01</td>
<td>4.946e-01</td>
<td>3.136e+00</td>
<td>1.395e+01</td>
<td>2.161e-01</td>
</tr>
<tr>
<td>Maximum Memory Usage (GB)</td>
<td>4.101e+02</td>
<td>2.002e-01</td>
<td>9.606e-01</td>
<td>4.800e+00</td>
<td>1.781e-01</td>
<td>2.448e-01</td>
</tr>
<tr>
<td>Time Spent in MPI Routines (sec)</td>
<td>1.228e+06</td>
<td>5.995e-02</td>
<td>4.984e+00</td>
<td>8.310e+00</td>
<td>5.177e+02</td>
<td>6.801e+02</td>
</tr>
<tr>
<td>Wallclock Time (sec)</td>
<td>1.003e+06</td>
<td>4.898e+02</td>
<td>6.428e-02</td>
<td>1.31e-02</td>
<td>4.898e+02</td>
<td>4.927e+02</td>
</tr>
</tbody>
</table>

CV = Coefficient of Variance = (Standard Deviation / Mean)

Task distribution of Aggregate Floating Point Operations (Flop x 10^{10}) - as a percentage of maximum

The MPI rank is the sum of the column and row indices in the table.

Table Columns: "64"
Statistics Across Tasks

<table>
<thead>
<tr>
<th>Sum</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>CV (%)</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.011e+02</td>
<td>1.470e-01</td>
<td>4.946e-03</td>
<td>3.36e+00</td>
<td>1.395e-01</td>
<td>2.161e-01</td>
</tr>
<tr>
<td>6.147e-01</td>
<td>3.002e-04</td>
<td>1.008e-05</td>
<td>3.36e+00</td>
<td>2.847e-04</td>
<td>4.411e-04</td>
</tr>
<tr>
<td>4.101e+02</td>
<td>2.002e-01</td>
<td>9.606e-03</td>
<td>4.80e+00</td>
<td>1.781e-01</td>
<td>2.448e-01</td>
</tr>
<tr>
<td>1.228e+06</td>
<td>5.995e+02</td>
<td>4.984e+01</td>
<td>8.31e+00</td>
<td>5.177e+02</td>
<td>6.801e+02</td>
</tr>
<tr>
<td>1.003e+06</td>
<td>4.898e+02</td>
<td>6.428e-02</td>
<td>1.31e-02</td>
<td>4.898e+02</td>
<td>4.927e+02</td>
</tr>
</tbody>
</table>

**9) - as a percentage of maximum

ea
IPM Examples

Time spent by each task in MPI_Recv as a percentage of the maximum value.

The MPI rank represented by each cell in the table is the sum of the cell's column and row indices.

Table Columns:

Time vs. MPI Rank for MPI_Recv
# User-Space I/O Profiling

### IO Summary from Darshan

<table>
<thead>
<tr>
<th>Exec. Runtime</th>
<th>MB Read</th>
<th>MB Written</th>
<th>Read Time (s)</th>
<th>Write Time (s)</th>
<th>Read Rate (MB/s)</th>
<th>Write Rate (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>02-17 16:37:42 - 02-17 16:40:25</td>
<td>1909</td>
<td>18670.36</td>
<td>4.97671</td>
<td>212.215</td>
<td>383.59</td>
<td>87.98</td>
</tr>
</tbody>
</table>

#### Number of Reads Per Size Range
- 33.3%
- 66.7%

#### Number of Writes Per Size Range
- 0_100
- 100_1K
- 100K_1M
- 100K_1M
- 4M_10M
- Other
- 91.2%
System-Level I/O Monitoring

Users can see the system-wide I/O activity while their job ran to look for contention.
Job Physical Topology