Debugging and Optimization Tools

Richard Gerber
NERSC
User Services Group Lead

Thanks to Woo-Sun Yang and Helen He
Outline

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

Videos, presentations, and references:

http://www.nersc.gov/users/training/courses/CS267/
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
Debugging
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
This is just the oddest bug I've ever seen. It wasn't happening until I found the mistake reading the source code.

A Schrödinger bug?

More than that, the bug manifests itself only after I "fix" the broken code. It works otherwise.

Then the broken code is not really broken.

It is just look at it.

What the hell is that?!!

Oh my God! Stop looking at it! The paradox may rip the space time continuum.

I guess I forgot a little debug statement!

http://www.geekherocomic.com/
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

• **Memory debuggers**
  - MAP
  - Valgrind

See NERSC web site
https://www.nersc.gov/users/software/debugging-and-profiling/
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);
}
```
NERSC NX SERVICE - X-WINDOWS
ACCELERATION AT NERSC

Introduction

NX is a computer program that handles remote X Window System connections and it provides three benefits for NERSC users:

- **SPEED**: NX can greatly improve the performance of X Windows, allowing users with slow, high latency connections (e.g., on cell phone network, traveling in Africa) to use complex X Windows programs (such as rotating a plot in Matlab).
- **SESSION**: NX provides sessions that allow a user to disconnect from the session and reconnect to it at a later time while keeping the state of all running applications inside the session.
- **DESKTOP**: NX gives users a virtual desktop that's running at NERSC. You can customize the desktop according to your work requirement.
Compile & Start DDT

Compile for debugging

```bash
edison% make
cc -c -g hello.c
cc -o hello hello.o
```

Set up the parallel run environment

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

Start the DDT debugger

```bash
edison% module load ddt
edison% ddt ./hello
```
At hang, tasks are in 3 different places.

Task 0 is at line 44.

Press Go and then Pause when code appears hung.
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)
Debuggers on NERSC machines

• **Parallel debuggers with a graphical user interface**
  – DDT (Distributed Debugging Tool)
  – TotalView

• **Specialized debuggers on Hopper and Edison**
  – STAT (Stack Trace Analysis Tool)
    • Collect stack backtraces from all (MPI) tasks
  – ATP (Abnormal Termination Processing)
    • Collect stack backtraces from all (MPI) tasks when an application fails
  – CCDB (Cray Comparative Debugger)
    • Comparative debugging

• **Valgrind**
  – Suite of debugging and profiler tools

https://www.nersc.gov/users/training/courses/CS267/ for links to recent training presentations
STAT (Stack Trace Analysis Tool)

- Gathers stack backtraces (showing the function calling sequences leading up to the ones in the current stack frames) from all (MPI) processes and merges them into a single file (*.dot)
  - Results displayed graphically as a call tree showing the location in the code that each process is executing and how it got there
  - Can be useful for debugging a hung application
  - With the info learned from STAT, can investigate further with DDT or TotalView

- Works for MPI, CAF and UPC, but not OpenMP

- For more info:
  - 'intro_stat', 'STAT', 'statview' and 'statgui' man pages
Hung application with STAT (Cont’d)

- Rank 0 is here
- Rank 1 & 2 are here
- Rank 3 is here

Slide Courtesy of Woo-Sun Yang
ATP (Abnormal Termination Processing)

- ATP gathers stack backtraces from all processes of a failing application
  - Invokes STAT underneath
  - Output in atpMergedBT.dot and atpMergedBT_line.dot (which shows source code line numbers), which are to be viewed with statview

- By default, the atp module is loaded on Hopper and Edison, but ATP is not enabled; to enable:
  - `setenv ATP_ENABLED 1`  # csh/tcsh
  - `export ATP_ENABLED=1`  # sh/bash/ksh

- For more info
  - ‘intro_atp’ man page
CCDB (Cray Comparative Debugger)

- Find a bug introduced in a version, by running two versions side by side and comparing data between them
- GUI
- Supports MPI; doesn’t support threading
- For info:
  - ccdb man page and help pages
  - lgdb man page and help pages
Running CCDB

```
% qsub -IV -lmppwidth=48,walltime=30:00 -q debug
% cd $PBS_O_WORKDIR
% module load cray-ccdb
% ccdb
```

Request enough nodes to run two apps. simultaneously

Slide Courtesy of Woo-Sun Yang
• **Suite of debugging and profiler tools**
  • Tools include
    – **memcheck**: memory error and memory leaks detection
    – **cachegrind**: a cache and branch-prediction profiler
    – **callgrind**: a call-graph generating cache and branch prediction profiler
    – **massif, dhat (exp-dhat)**: heap profilers
    – **helgrind, drd**: pthreads error detectors
  • **For info:**

[Image of slide courtesy of Woo-Sun Yang]
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 it 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% of theoretical is pretty good
  - GPUs, Xeon Phi: >1% in today’s real full HPC applications pretty good? This your challenge!
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
Performance Landscape
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, Darshan

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`
Cray perftools and perftools-lite

• Reports:
  – execution time
  – memory high water mark
  – aggregate FLOPS rate
  – top time consuming user function
  – MPI information
  – IO information
  – hardware performance counters
  – load balance ...

• Start with perftools-lite
• Available on Hopper and Edison.
• Documentation:

Number of PEs (MPI ranks): 240
Numbers of PEs per Node: 24 PEs on each of 10 Nodes
Numbers of Threads per PE: 1
Number of Cores per Socket: 12
Execution start time: Sun Feb 2 13:38:33 2014
System name and speed: nid01665 2401 MHz
Wall Clock Time: 290.82940 secs
High Memory: 243.36 MBytes
MFLOPS (aggregate): Not supported (see observation below)
I/O Read Rate: 46.30 MBytes/Sec
I/O Write Rate: 5.91 MBytes/Sec

Table 1: Profile by Function Group and Function (top 10 functions shown)
100.0% | 28484.6 | -- | -- | Total
|--------------------------------|
| 61.8% | 17598.4 | -- | -- | USER
|--------------------------------|
| 36.3% | 10328.2 | 58.8 | 0.6% | decompmod_initdecomp_

...
Allinea MAP

- Allinea MAP is a parallel MPI profiler with GUI, small overhead.
- Reports: Memory usage, MPI usage, CPU time, CPU instructions, I/O, etc. as a function of time.
- Available on Hopper, Edison, and Carver.

Documentations:
http://www.nersc.gov/users/software/debugging-and-profiling/MAP/
http://www.allinea.com/products/map/
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**
  – Work with vendors (e.g., CRAY ARU, Allinea Perf. Report)
  – IPM (MPI profiling, chip HW counters, memory used)
  – Accounting & UNIX resource usage
  – System-level I/O monitoring
  – User-level I/O profiling (Darshan)
## NERSC Completed Jobs

<table>
<thead>
<tr>
<th>#</th>
<th>Host</th>
<th>JobID</th>
<th>Job Name</th>
<th>User</th>
<th>Nds</th>
<th>Complete</th>
<th>Wall hrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>edison</td>
<td>782557</td>
<td>my_job</td>
<td>hdixit</td>
<td>11</td>
<td>02/17/14 13:14</td>
<td>0.160</td>
</tr>
<tr>
<td>1</td>
<td>edison</td>
<td>782431</td>
<td>cp_ref_big</td>
<td>hsinyu</td>
<td>13</td>
<td>02/17/14 13:14</td>
<td>0.849</td>
</tr>
<tr>
<td>2</td>
<td>edison</td>
<td>782591</td>
<td>tvsoi</td>
<td>wangyu</td>
<td>2</td>
<td>02/17/14 13:14</td>
<td>0.005</td>
</tr>
<tr>
<td>3</td>
<td>edison</td>
<td>782560</td>
<td>my_job</td>
<td>hdixit</td>
<td>11</td>
<td>02/17/14 13:13</td>
<td>0.131</td>
</tr>
<tr>
<td>4</td>
<td>edison</td>
<td>719618</td>
<td>nacl_big</td>
<td>camevav</td>
<td>16</td>
<td>02/17/14 13:12</td>
<td>11.453</td>
</tr>
<tr>
<td>5</td>
<td>edison</td>
<td>782403</td>
<td>slab-Ni-pa</td>
<td>hdixit</td>
<td>11</td>
<td>02/17/14 13:12</td>
<td>1.006</td>
</tr>
<tr>
<td>6</td>
<td>edison</td>
<td>782590</td>
<td>NL_test</td>
<td>khkim</td>
<td>86</td>
<td>02/17/14 13:12</td>
<td>0.008</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>APID</th>
<th>Command</th>
<th>Nodes</th>
<th>Tasks</th>
<th>Threads per Task</th>
<th>Tasks per Node</th>
<th>Max Task Mem (MB)</th>
<th>Run Time (secs)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>3140812</td>
<td>gribmeanp.x</td>
<td>3</td>
<td>64</td>
<td>1</td>
<td>24</td>
<td>235</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Command line:**

```
/opt/cray/alps/default/bin/aprun -n 64 -N 24
/scratch1/scratchdirs/whitaker/edison/newstuff/bin/gribmeanp.x
/scratch1/scratchdirs/whitaker/gfsexkfl_t126_1999iau_bias
```

**Node list:**

`2628-2630`
IPM: An Easy to Use Performance Tool

Just load the module, relink, and run.

# host   : s05601/006035314C00_AIX  mpi_tasks : 32 on 2 nodes
# start  : 11/30/04/14:35:34  wallclock : 29.975184 sec
# stop   : 11/30/04/14:36:00  %comm  : 27.72
# gbytes : 6.65863e-01 total  gflop/sec : 2.33478e+00 total
#  [total]  <avg>  min max
# wallclock  953.272  29.7897  29.6092  29.9752
# user    837.25  26.1641  25.71  26.92
# system  60.6  1.89375  1.52  2.59
# mpi    264.267  8.25834  7.73025  8.70985
# %comm  27.7234  25.8873  29.3705
# gflop/sec  2.33478  0.0729619  0.072204  0.0745817
# gbytes  0.665863  0.0208082  0.0195503  0.0237541
# PM_FPU0_CMPL  2.28827e+10  7.15084e+08  7.07373e+08  7.30171e+08
# PM_FPU1_CMPL  1.70657e+10  5.3304e+08  5.28487e+08  5.42882e+08
# PM_FPU_FMA  3.00371e+10  9.3866e+08  9.27762e+08  9.62547e+08
# PM_INST_CMPL  2.78819e+11  8.71309e+09  8.20981e+09  9.21761e+09
# PM_LD_CMPL  1.25478e+11  3.92118e+09  3.74541e+09  4.11658e+09
# PM_ST_CMPL  7.45961e+10  2.33113e+09  2.21164e+09  2.46327e+09
# PM_TLB_MISS  2.45896e+10  9.55461e+09  9.36585e+09  9.62227e+09
#  [time]  [calls]  <mpi>  <wall>
# MPI_Send  188.386  639616  71.29  19.76
# MPI_Wait  69.5032  639616  26.30  7.29
# MPI_Recv  6.34936  639616  2.40  0.67
# MPI_Barrier  0.0177442  32  0.01  0.00
# MPI_Reduce  0.00540609  32  0.00  0.00
# MPI_Comm_size  0.00465156  32  0.00  0.00
# MPI_Comm_rank  0.000145341  32  0.00  0.00
## IPM Data on NERSC Web Site

### NERSC job details

[Image of NERSC logo]


Task distribution of Aggregate Floating Point Operations (Flop x 10^19) as a percentage of maximum.

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

<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**19)</td>
<td>3.01e+02</td>
<td>1.47e-01</td>
<td>4.94e+01</td>
<td>3.36e+00</td>
<td>1.395e-01</td>
<td>2.161e-01</td>
</tr>
<tr>
<td>Maximum Memory Usage (GBytes)</td>
<td>4.101e+02</td>
<td>2.002e-01</td>
<td>9.606e-01</td>
<td>4.804e+00</td>
<td>1.781e-01</td>
<td>2.448e+00</td>
</tr>
<tr>
<td>Time Spent in MPI Routines (sec)</td>
<td>1.228e+06</td>
<td>5.995e+02</td>
<td>4.986e+01</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>4.898e+02</td>
<td>4.898e+02</td>
<td>4.927e+02</td>
</tr>
</tbody>
</table>

\[ CV = \text{Co-efficient of Variance} = \frac{\text{Standard Deviation}}{\text{Mean}} \]

### gflops vs. MPI Rank

[Image of gflops vs. MPI Rank graph]
### 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

e.
IPM Examples

Time spent by each task in \textit{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:

|   | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 76 | 76 | 73 | 70 | 72 | 77 | 71 | 72 | 71 | 71 | 71 | 71 | 60 | 62 | 57 | 48 | 47 | 44 | 42 | 40 | 38 | 36 | 34 | 32 | 30 | 28 | 26 | 24 | 22 | 20 | 18 | 16 | 14 |
| 1 | 51 | 52 | 50 | 49 | 48 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| 2 | 64 | 91 | 80 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 |
| 3 | 90 | 86 | 87 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 |
| 4 | 128 | 129 | 128 | 127 | 126 | 125 | 124 | 123 | 122 | 121 | 120 | 119 | 118 | 117 | 116 | 115 | 114 | 113 | 112 | 111 | 110 | 109 | 108 | 107 | 106 | 105 | 104 | 103 |
| 5 | 160 | 159 | 158 | 157 | 156 | 155 | 154 | 153 | 152 | 151 | 150 | 149 | 148 | 147 | 146 | 145 | 144 | 143 | 142 | 141 | 140 | 139 | 138 | 137 | 136 | 135 | 134 | 133 |

Time vs. MPI Rank for \textit{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

- 91.2%
Darshan Report

Estimated I/O Time: 29.5%
Darshan Report

Number of Transaction in Each Size Range

Amount Written with Different Transaction Sizes
Summary

• Debugging and Parallel Code Optimization can be hard
• Tools can help
  – See NERSC web pages for recommendations
  – Use the ones that work for you
• Be aware of some of the more common errors and best practices
• Look for outliers in parallel programs
• Refer to NERSC web pages for details
  – http://www.nersc.gov/users/software/debugging-and-profiling/
  – http://www.nersc.gov/users/training/courses/CS267/