Debugging and Optimization Tools

Richard Gerber
NERSC User Services

David Skinner
NERSC Outreach, Software & Programming Group

UCB CS267
February 16, 2012
Outline

• Introduction
• Debugging
• Performance / Optimization

Videos, presentations, and references:

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

Also see the DOE Advanced Computational Tools:
http://acts.nersc.gov
• Today’s Talks
  – Strategies for parallel performance (D. Skinner)
  – Debugging and optimization tools (R. Gerber)

• Take Aways
  – Common problems to look out for
  – How tools work in general
  – A few specific tools you can try
  – Where to get more information
Debugging
• 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 computer in 1947).
Common Causes of Bugs

• **“Serial”**
  – Invalid memory references
  – Array reference out of bounds
  – Divide by zero
  – Use of uninitialized variables

• **Parallel**
  – Unmatched sends/receives
  – Blocking receive before corresponding send
  – Out of order collectives
  – Race conditions
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

- **Run It**
  - Check for proper behavior
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
  – GNU debugger, serial, command-line interface
  – See “man gdb”

Parallel debuggers Using X-Windows
  – DDT
  – Totalview
## Out of bounds reference in source code for program “flip”

```plaintext
allocate(put_seed(random_size))
```

```plaintext
bad_index = random_size+1
put_seed(bad_index) = 67
```

### Compiler runtime bounds checking

```plaintext
ftn -c -g -Ktrap=fp -Mbounds flip.f90
ftn -c -g -Ktrap=fp -Mbounds printit.f90
ftn -o flip flip.o printit.o -g
```

```plaintext
% qsub -I -qdebug -lmppwidth=48
% cd $PBS_O_WORKDIR
% 
% aprun -n 48 ./flip
```

0: Subscript out of range for array

```plaintext
put_seed (flip.f90: 50)
```

```plaintext
subscript=35, lower bound=1, upper bound=34, dimension=1
```

0: Subscript out of range for array

```plaintext
put_seed (flip.f90: 50)
```

```plaintext
subscript=35, lower bound=1, upper bound=34, dimension=1
```
• For a list of compiler options, see the man pages for the individual compilers
  – man pgcc
  – man pgCC
  – man pgf90
  – man gcc
  – man gfortran
  – Etc.
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

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

Set up the parallel run environment

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

Start the DDT debugger

```bash
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.
At hang, tasks are in 3 different places.

Task 3 is at line 47
DDT video

- http://vimeo.com/19978486
- Or http://vimeo.com/user5729706
- This is out of date; I need to change the NX server from “Euclid” to “nx.nersc.gov and “hopp2” to “hopper”
Other Debugging Tips

- **Try different compilers**
  - Diagnostic messages and language spec compliances differ

- **Look for memory corruption**
  - Bad memory reference in one place (array out of bounds) can make code crash elsewhere
  - It might appear that you’re crashing on a perfectly valid line of code

- **Check the arguments to your MPI calls**

- **Call the NERSC Consultants (800-66-NERSC or 510 486-8600)**
Performance / Optimization
Performance Questions

• How can we tell if a program is performing well?
• Or isn’t?
• If performance is not “good,” how can we pinpoint why?
• How can we identify the causes?
• What can we do about it?
• **Primary metric: application time**
  – but gives little indication of efficiency

• **Derived measures:**
  – rates (Ex.: messages per unit time, Flops per second, clocks per instruction), cache utilization

• **Indirect measures:**
  – speedup, parallel efficiency, scalability
Optimization Strategies

• Serial
  – Leverage ILP on the processor
  – Feed the pipelines
  – Exploit data locality
  – Reuse data in caches

• Parallel
  – Minimize latency effects (aggregate messages)
  – Maximize work vs. communication

• Both
  – Minimize data movement (recalculate vs. send)
  – Memory locality on NUMA processors - first touch
Identifying Targets for Optimization: Profiling

- **Sampling**
  - Regularly interrupt the program and record where it is
  - Build up a statistical profile of time spent in various routines
  - Concentrate first on longest running sections or routines

- **Tracing**
  - Insert hooks into program to record and time program events (logging)
  - Reasonable for sequential programs
  - Unwieldy for large parallel programs (too much data!)
Identifying Targets for Optimization

- **Hardware Event Counters**
  - Special registers count events on processor
  - E.g. number of floating point instructions
  - Many possible events
  - Only a few can be recorded at a time (~4 counters)
  - Can give you an idea of how efficiently you are using the processor hardware
Typical Process

• (Sometimes) Modify your code with macros, API calls, timers
• Compile your code
• Transform your binary for profiling / tracing with a tool
• Run the transformed binary
  – A performance data file is produced
• Interpret the results with a tool
Performance Tools @ NERSC

- **Vendor Tools:**
  - CrayPat on Crays

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

- **IPM: Integrated Performance Monitoring**
  - A low overhead, low effort NERSC tool
Introduction to 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)
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
Using Apprentice

- Optional visualization tool for Cray’s perftools data
- Use it in a X Windows environment
- Uses a data file as input (xxx.ap2) that is prepared by pat_report

```bash
app2 [--limit_per_pe tags] XXX.ap2
```
Apprentice Basic View

- Can select new (additional) data file and do a screen dump
- Worthless
- Useful
- Can select other views of the data
- Can drag the “calipers” to focus the view on portions of the run
• 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:
    
    ```
    qsub -I -lmppwidth=24
    module load perftools
    aprun -n 1 papi_avail
    ```

• AMD native events also provided; use ‘papi_native_avail’:
  
  ```
  aprun -n 1 papi_native_avail
  ```
TAU

- Tuning and Analysis Utilities
- Fortran, C, C++, Java performance tool
- Procedure
  - Insert macros
  - Run the program
  - View results with pprof
- More info than gprof
  - E.g. per process, per thread info; supports pthreads
You will have a homework assignment using TAU

- `module load tau`
- Define paths in Makefile
- Modify header file to define TAU macros
- Add macro calls to the code
- Compile and submit to batch queue
- Use `pprof` to produce readable output

Good reference

Experience with NERSC Users

- NERSC has about 5,000 users
  - All levels of sophistication and experience
  - We’re committed to supporting both the cutting edge & production HPC computing for the masses
- Users often ask for advice on which tools to use and we give them suggestions
- Our experience is that very few use programming/debugging/development tools
- A few users use a few tools a lot, but many try a tool only once
• Extremely effective?

• More likely: Too confusing, difficult, didn’t work, don’t know how to use, don’t know which to use, tied to a platform, compiler, or language

• It’s not that we don’t have tools that address specific issues
  – TAU, PAPI, HPC Toolkit
  – Craypat, IBM HPC tools, OpenSpeedShop, Intel
  – Valgrind (memory debugging)
  – GPU/CUDA tools & compilers
  – Vampirtrace

• But do most users have the resources to learn how to use these tools, esp. when they don’t know if there will be any benefit from any given one?
• Integrated Performance Monitoring
• MPI profiling, hardware counter metrics, IO profiling (?)
• IPM requires no code modification & no instrumented binary
  – Only a “module load ipm” before running your program on systems that support dynamic libraries
  – Else link with the IPM library
• IPM uses hooks already in the MPI library to intercept your MPI calls and wrap them with timers and counters
• **How it works (user perspective)**
  - `% module load IPM`
  - Run program as normal
  - Look at results on the web

• **It’s that easy!**
  - And extremely low overhead, so IPM is examining your production code

* (As long as your system supports dynamic load libs)
What IPM measures

• IPM “only” gives a high-level, entire-program-centric view
• Still, very valuable guidance
  – Shows whole-run info per MPI task, OpenMP thread, (CUDA under development)
  – Many pieces of data in one place
• Reveals what many users don’t know about their code
  – High-water memory usage (per task)
  – Load balance
  – Call imbalance
  – MPI time
  – I/O time
### IPM

- **host:** s05601/006035314C00_AIX
- **mpi_tasks:** 32 on 2 nodes
- **wallclock:** 29.975184 sec
- **%comm:** 27.72
- **gbytes:** 6.65863e-01 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 0.072204 0.0745817
- **pm_fpu0_cmpl:** 2.28827e+10
  - [time] [calls] <mpi> <wall>
  - MPI_Send 188.386 639616 71.29 19.76
  - MPI_Wait 69.5032 639616 26.30 7.29
  - MPI_Irecv 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_rank 0.00465156 32 0.00 0.00
  - MPI_Comm_size 0.000145341 32 0.00 0.00
### IPM Examples

Click on the metric you want.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate Floating Point Operations (Flop x 10**9)</td>
<td>1.313e+01</td>
<td>2.565e-02</td>
<td>6.10</td>
<td>1.713e-02</td>
<td>2.758e-02</td>
</tr>
<tr>
<td>GFlop/sec</td>
<td>1.492e-01</td>
<td>2.895e-04</td>
<td>6.10</td>
<td>1.934e-04</td>
<td>3.114e-04</td>
</tr>
<tr>
<td>Maximum Memory Usage (GBytes)</td>
<td>3.226e+01</td>
<td>6.301e-02</td>
<td>10.12</td>
<td>5.708e-02</td>
<td>1.947e-01</td>
</tr>
<tr>
<td>Time Spent in MPI Routines (sec)</td>
<td>4.044e+04</td>
<td>7.809e+01</td>
<td>4.05</td>
<td>9.801e+00</td>
<td>8.359e+01</td>
</tr>
<tr>
<td>Wallclock Time (sec)</td>
<td>4.537e+04</td>
<td>8.861e+01</td>
<td>0.19</td>
<td>8.848e+01</td>
<td>8.895e+01</td>
</tr>
</tbody>
</table>

Memory in units of gigabytes, time in seconds.

### Hardware counter statistics - 512 tasks

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAPI_FP_OPM</td>
<td>1.161799e+12</td>
<td>2.269139e+05</td>
<td>6.09</td>
<td>1.51503e+09</td>
<td>2.439529e+09</td>
</tr>
</tbody>
</table>

### MPI Time Statistics - 512 tasks

<table>
<thead>
<tr>
<th>Cell</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
<th>% of MPI</th>
<th>% of wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Bcast</td>
<td>3.517e+04</td>
<td>6.869e+01</td>
<td>4.68</td>
<td>4.342e-01</td>
<td>7.269e+01</td>
<td>86.969</td>
<td>77.520</td>
</tr>
<tr>
<td>MPI_Scatterv</td>
<td>2.589e+03</td>
<td>5.057e+00</td>
<td>5.79</td>
<td>1.059e+00</td>
<td>5.540e+00</td>
<td>6.403</td>
<td>5.707</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>2.176e+03</td>
<td>4.249e+00</td>
<td>17.82</td>
<td>1.250e+00</td>
<td>4.968e+00</td>
<td>5.380</td>
<td>4.795</td>
</tr>
<tr>
<td>MPI_Gather</td>
<td>4.312e+02</td>
<td>8.422e-01</td>
<td>36.44</td>
<td>3.552e-03</td>
<td>2.271e+00</td>
<td>1.066</td>
<td>0.950</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>5.250e+01</td>
<td>1.025e-01</td>
<td>11.96</td>
<td>7.182e-02</td>
<td>1.259e-01</td>
<td>0.130</td>
<td>0.116</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>1.033e+01</td>
<td>2.017e-02</td>
<td>10.21</td>
<td>1.217e-02</td>
<td>2.613e-02</td>
<td>0.026</td>
<td>0.023</td>
</tr>
<tr>
<td>MPI_Gather</td>
<td>1.021e+01</td>
<td>1.995e-02</td>
<td>502.07</td>
<td>1.391e-03</td>
<td>1.434e+00</td>
<td>0.025</td>
<td>0.023</td>
</tr>
<tr>
<td>MPI_Comm rank</td>
<td>4.563e+01</td>
<td>8.913e-04</td>
<td>4.74</td>
<td>7.799e-04</td>
<td>1.404e-03</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>MPI_Comm size</td>
<td>9.629e-02</td>
<td>1.881e-04</td>
<td>10.65</td>
<td>1.462e-04</td>
<td>4.859e-04</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Init</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Average MPI Time per Task

- **MPI_Bcast**: 86.969%
- **MPI_Scatterv**: 6.403%
- **MPI_Wait**: 5.380%
- **MPI_Gather**: 1.066%

**U.S. Department of Energy**

**Office of Science**

**Berkeley Lab**
### IPM Examples

**Task distribution of Aggregate Floating Point Operations (Flop x 10**8)** - 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**8)</td>
<td>3.012e+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>GFlop/sec</td>
<td>6.147e+00</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>Maximum Memory Usage (MB)</td>
<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>2.449e-01</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>6.31e+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>6.028e+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)**
IPM Examples

Time spent by each task in MPI_Allreduce 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: 48

|       | 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---
Task distribution of Maximum Memory Usage (GB/Bytes) - as a percentage of maximum

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

| Column Indices | 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 |
| Row Indices    |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
IPM Examples

Time spent by each task in MPI_Rcv 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: 32

|   | 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 | 74 | 76 | 69 | 70 | 77 | 77 | 70 | 71 | 60 | 62 | 60 | 61 | 64 | 65 | 64 | 65 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| 32 | 70 | 71 | 63 | 65 | 72 | 72 | 66 | 62 | 54 | 53 | 55 | 56 | 57 | 56 | 58 | 57 | 51 | 53 | 57 | 57 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 64 | 64 | 65 | 60 | 61 | 64 | 65 | 62 | 62 | 59 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 61 | 62 | 64 | 65 | 66 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 96 | 68 | 75 | 69 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 |
| 128 | 72 | 74 | 76 | 78 | 80 | 82 | 84 | 86 | 88 | 90 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 |
| 160 | 76 | 78 | 80 | 82 | 84 | 86 | 88 | 90 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 | 132 | 134 |
| 192 | 80 | 82 | 84 | 86 | 88 | 90 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 | 132 | 134 | 136 |
| 224 | 84 | 86 | 88 | 90 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 | 132 | 134 | 136 | 138 | 140 |
| 256 | 88 | 90 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 | 132 | 134 | 136 | 138 | 140 | 142 | 144 |
| 288 | 92 | 94 | 96 | 98 | 100 | 102 | 104 | 106 | 108 | 110 | 112 | 114 | 116 | 118 | 120 | 122 | 124 | 126 | 128 | 130 | 132 | 134 | 136 | 138 | 140 | 142 | 144 | 146 | 148 |

Time vs. MPI Rank for MPI_Rcv
Questions to You

• What tools do you use?
• What tools do you want?
• What would you like centers to support?
• Can you get to exascale without tools?
Users are asking for tools because HPC systems and programming models are changing

More and more components to worry about

- CPU (caches, FPUs, pipelining, …)
- Data movement to main memory, GPU memory, levels of cache
- I/O
- Network (message passing)
- CPU Threads (OpenMP)
- GPU performance
What I Want in a Tool

- Let the users help themselves
- Work for everyone all (most of?) the time
- Easy to use
- Useful
- Easy to interpret the results
- Affordable ($$ or manpower support costs)
- Simple, supplement existing complex tools
  - Point the way for a “deeper dive” in problem areas