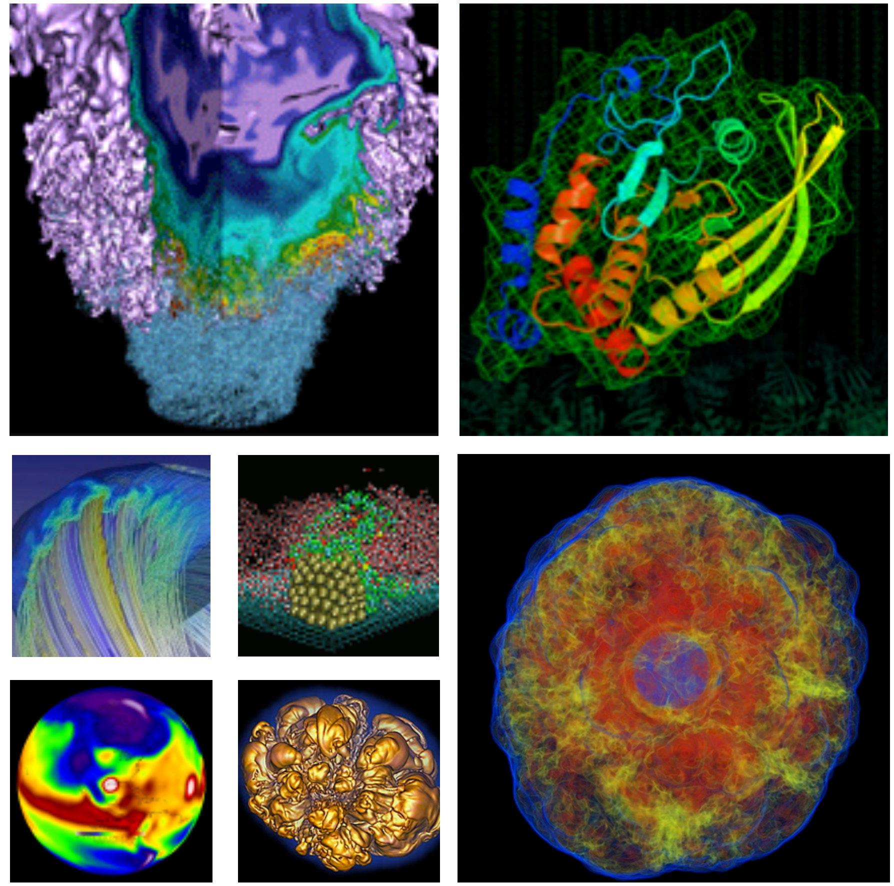
Debugging and Optimization Tools



Richard Gerber
NERSC
User Services Deputy Group Lead

NERSC

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Outline

- **Take-Aways**
- **Debugging**
- **Performance / Optimization**
- **NERSC "automatic" tools**

Videos, presentations, and references:

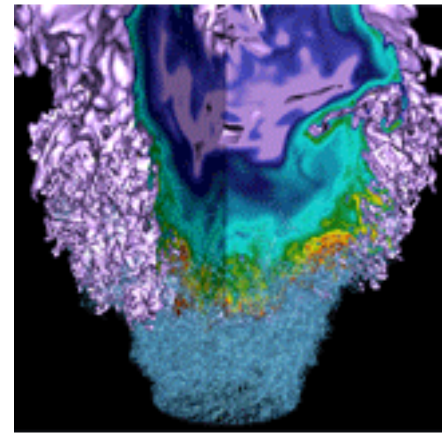
http://www.nersc.gov/users/training/courses/CS267/

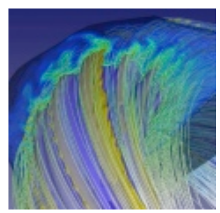
Also see the DOE Advanced Computational Tools:
http://acts.nersc.gov

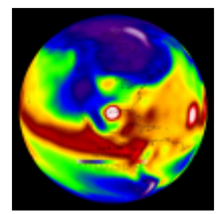
Take-Aways

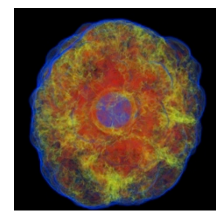
- **Tools can help you find errors in your program and locate performance bottlenecks**
- **In the world of HPC parallel computing, there are few widely adopted standard tools**
  - Totalview and DDT debuggers
  - PAPI, Tau, & vendor-specific performance tools
- **Common code problems**
- **How tools 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**

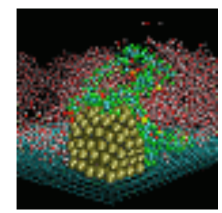
Debugging





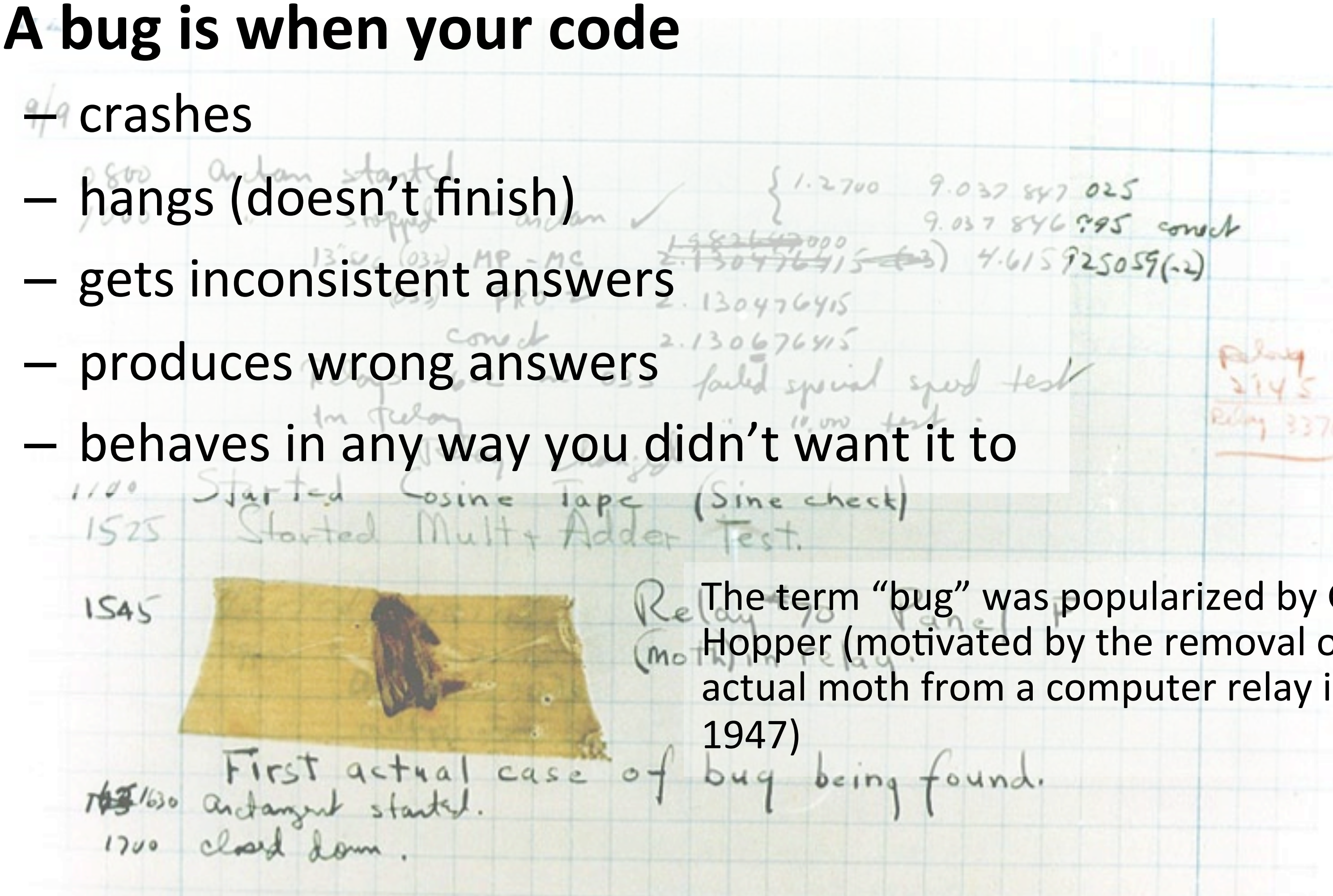






What is a Bug?

- **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

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

```
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

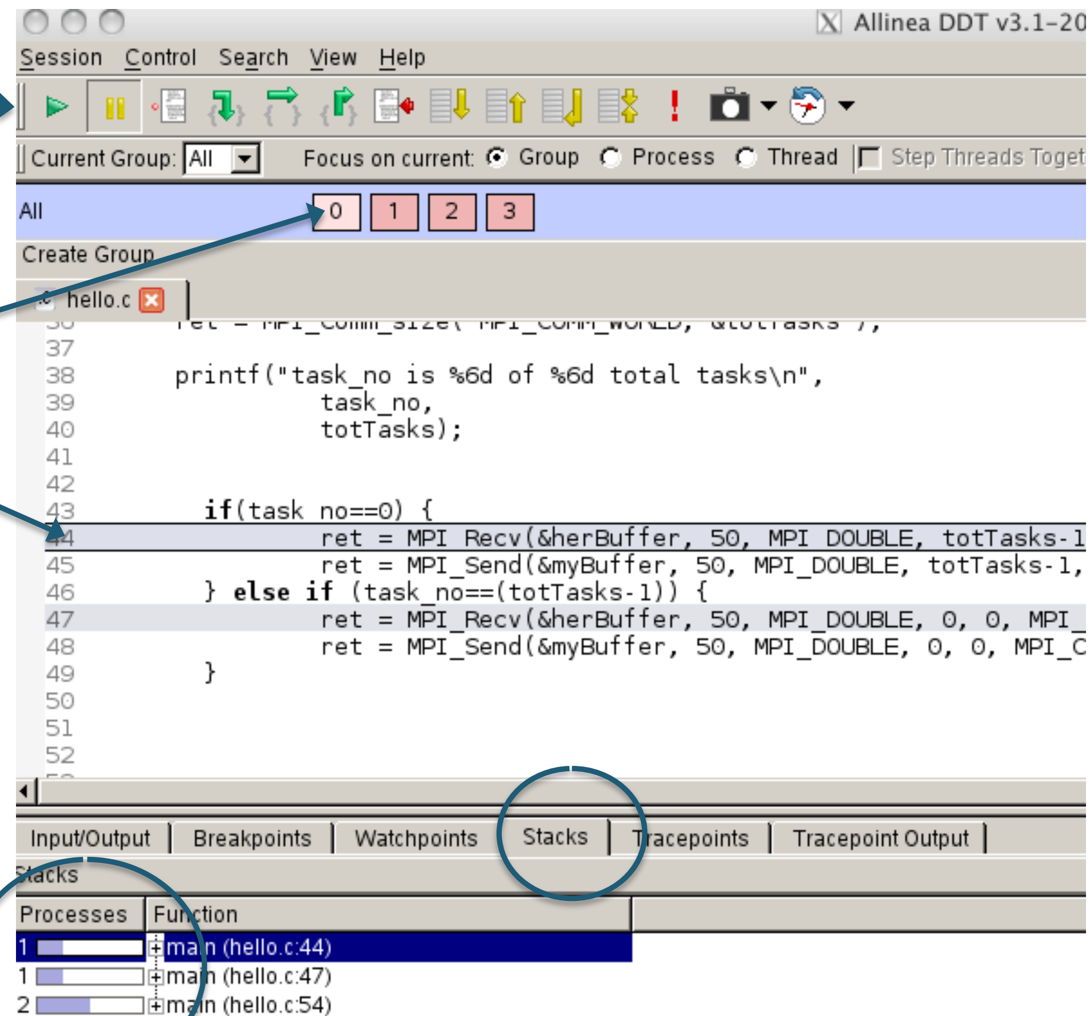
```
hopper% qsub -I -V -lmppwidth=24
hopper% cd $PBS_O_WORKDIR
```

Start the DDT debugger

```
hopper% module load ddt
hopper% ddt ./hello
```

DDT Screen Shot

Press Go and then Pause when code appears hung.



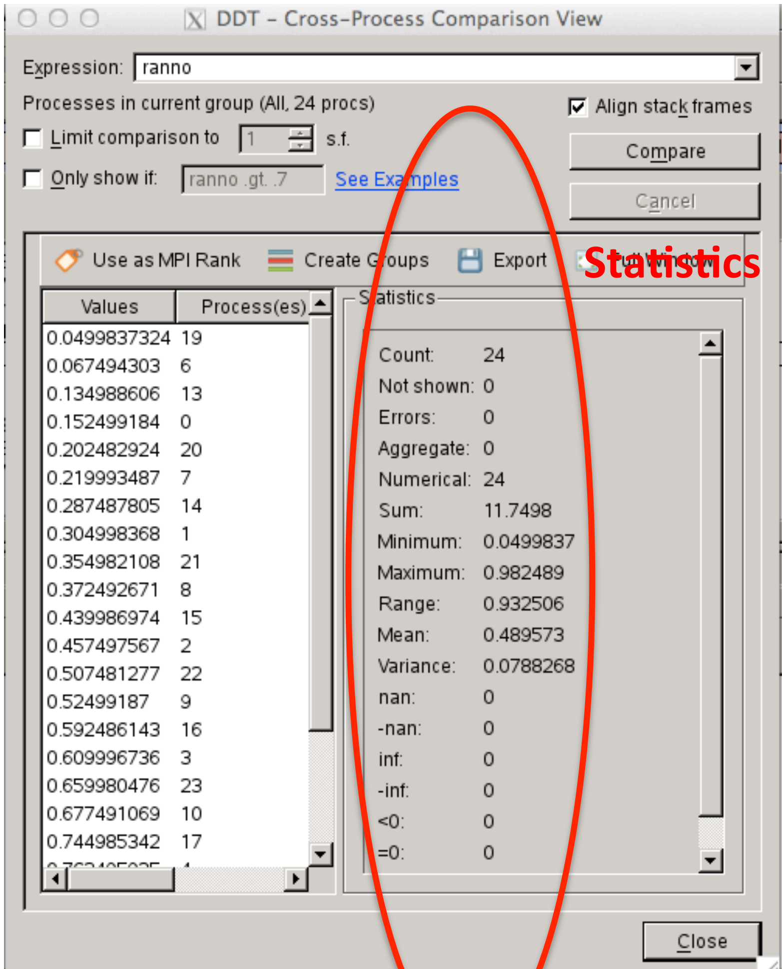
Task 0 is at line 44

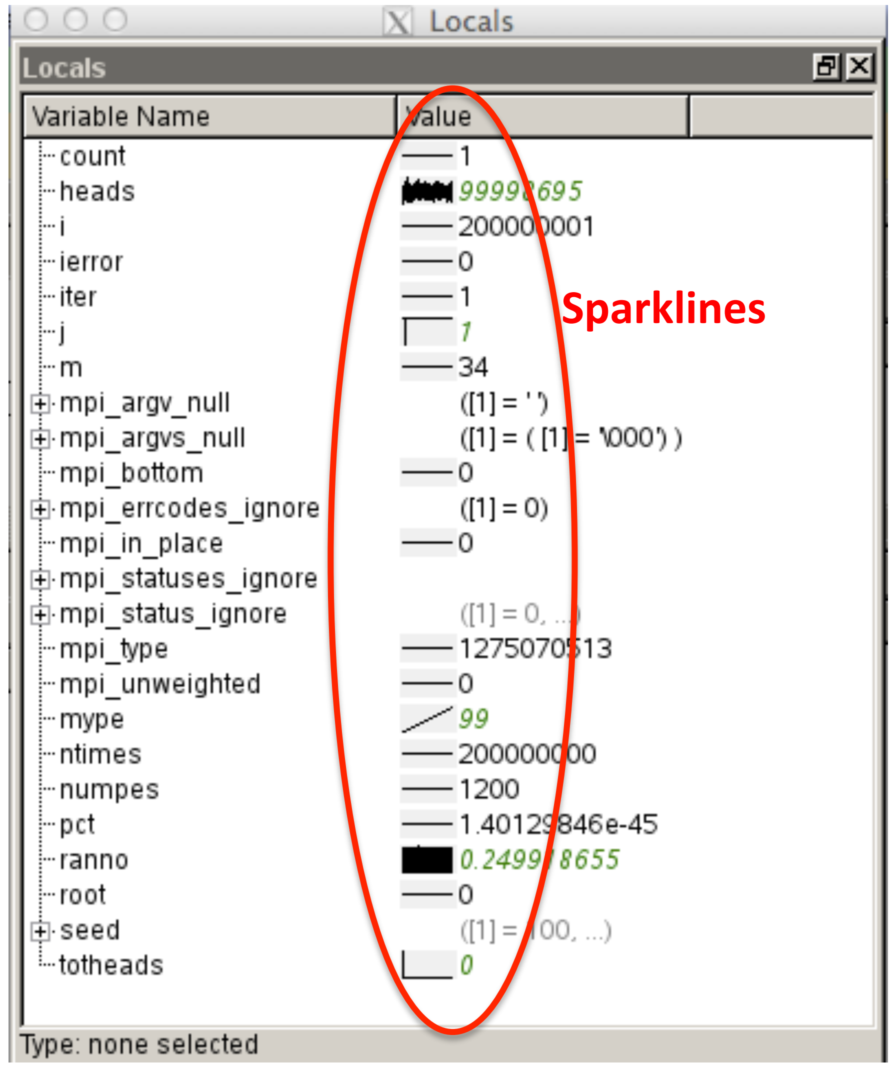
At hang, tasks are in 3 different places.

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)

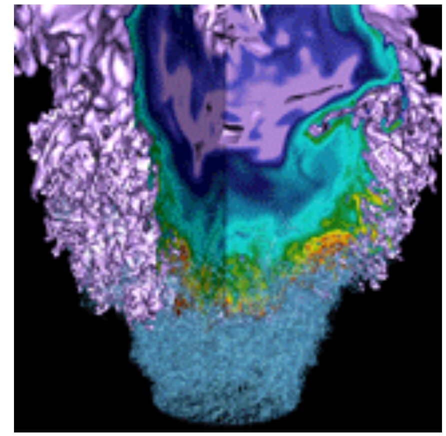


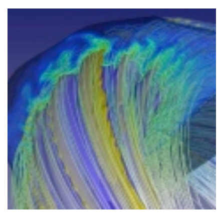


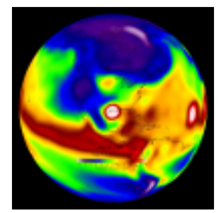
DDT video

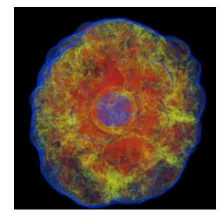
- **http://vimeo.com/19978486**
- **Or http://vimeo.com/user5729706**
- **Linked to from http://www.nersc.gov/users/training/courses/CS267/**

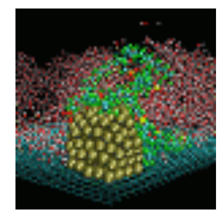
Performance / Optimization











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

```
# 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    5.33304e+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            2.45894e+08    7.68418e+06   6.98733e+06   2.05724e+07
# PM_CYC                  3.0575e+11    9.55467e+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_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
```

Completed Jobs on NERSC Web Site

NERSC job details

http://www.nersc.gov/REST/jobs/ipm_summary.php?stepid=619349.sdb&name=gflops&timestamp=1310766809

Metric	Sum	Mean	Std. Dev.	CV (%)	Minimum	Maximum
Aggregate Floating Point Operations (Flop x 10**9)	3.011e+02	1.470e-01	4.946e-03	3.36e+00	1.395e-01	2.161e-01
GFlop/sec	6.147e-01	3.002e-04	1.008e-05	3.36e+00	2.847e-04	4.411e-04
Maximum Memory Usage (GBytes)	4.101e+02	2.002e-01	9.606e-03	4.80e+00	1.781e-01	2.448e-01
Time Spent in MPI Routines (sec)	1.228e+06	5.995e+02	4.984e+01	8.31e+00	5.177e+02	6.801e+02
Wallclock Time (sec)	1.003e+06	4.898e+02	6.428e-02	1.31e-02	4.898e+02	4.927e+02

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

Task distribution of *Aggregate Floating Point Operations (Flop x 10**9)* - as a percentage of maximum

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

Table Columns: 64

	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	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	94	70	68	67	66	66	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65
64	66	65	65	66	66	65	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	67	66	66	66	66	66	66	67	66	66	67	67	67	67	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
128	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	68	67	68	67	68	68	68	67	68	67	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68
192	68	68	68	68	68	68	68	68	68	68	69	68	68	68	69	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	68	68	68	68	68	68	68	68	69	68	68	68	68	69	68	68	69	68	68	68	68	68	68	68	68	68	68
256	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	69	68	69	68	69	68	68	68	69	68	68	69	68	68	68	68	68	68	69	68	68	68	68	68	68	68	69	69	68	68	68	69	68	68	68	69	68	69	68	68	68	69	68
320	68	68	68	68	68	68	69	68	68	69	68	68	69	68	68	68	68	68	69	68	68	68	68	68	68	69	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	67	68	67
384	68	68	68	68	68	68	68	67	67	68	67	68	67	67	67	67	67	67	67	67	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	66	67	67	67	67	66	67	67	66	67	66	66	66	67	66
448	66	66	67	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	67	66	66	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	68	68	69	69	70	71	74	99
512	100	74	72	71	70	69	69	69	69	68	68	68	68	68	68	68	68	68	68	67	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	67	68	67	68	67	68	68	68	68	67	68	68	68
576	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	69	68	69	68	68	69	69	68	69	69	69	69	68	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69
640	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	70	69	69	70	69	69	70	69	70	70	69	70	69	69	70	69	70	69	70	69	69	70	69	70	69	70	70	70	70	69	70	69	70	70	69	70	69	70	70	70	70	70	70	70	70	70	70
704	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	69	70	69	70	70	70	70	70	70	70	69	70	69	70	70	69	70	69	69	70	69	70	70	69	70	69	69	70	69	70
768	69	69	70	69	69	70	69	69	69	70	70	69	70	69	70	70	70	70	69	70	69	70	69	70	69	70	69	70	69	70	70	69	70	69	70	69	70	70	70	70	70	70	70	70	70	69	70	70	70	69	69	70	70	69	70	70	70	69	70	70	69	70	70	69
832	70	70	70	69	70	69	70	69	69	70	69	69	70	69	69	69	69	69	70	69	69	69	69	69	69	70	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	68	69	69	69	69	69	69	69	68	69	68	69	69	69	68	69	68
896	68	69	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	67	68	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
960	66	67	67	66	67	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	65	66	65	66	65	66	65	65	65	65	65	66	65	66	65	66	66	66	66	66	66	66	66	67	67	68	70	95
1024	100	74	72	71	70	69	69	69	69	68	68	68	68	68	68	68	68	68	68	67	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	68	68	68	68	67	68	68	68	68	67	68	68	68
1088	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	69	68	69	68	69	68	68	69	69	68	69	69	69	69	68	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69
1152	69	69	69	69	69	69	69	69	69	69	69	69	69	69	70	70	69	70	69	69	70	69	69	70	69	70	70	69	70	69	69	70	69	70	69	70	70	69	70	69	70	69	70	70	70	70	70	70	69	70	70	69	70	69	70	70	70	70	70	70	70	70	70	70
1216	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	69	70	70	70	70	70	70	70	69	70	69	70	70	69	70	70	69	70	69	70	70	69	70	69	69	70	70	70
1280	69	69	70	69	69	70	70	69	70	70	70	69	70	69	70	70	70	70	69	70	69	70	69	70	69	70	69	70	69	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	69	70	70	70	69	70	70	70	69	70	70	70	69	70	70	69	70	70	69
1344	70	70	70	70	70	70	70	69	69	70	69	69	70	69	69	69	69	69	70	69	69	69	69	69	69	70	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	68	69	68
1408	68	69	69	68	68	68	69	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	67	68	68	68	68	67	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
1472	66	67	67	66	67	67	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	65	66	65	66	65	66	65	66	65	65	65	66	65	66	65	66	66	66	66	66	66	66	66	67	67	68	70	95
1536	94	70	68	67	66	66	66	66	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	65	66	65	66	65	66	65	66	66	66	66	65	66	66	66
1600	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	67	66	66	66	67	66	67	66	67	67	66	67	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	67	67	67	67	68
1664	68	67	67	67	68	68	68	68	68	67	68	68	68	68	68	68	67	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	68	68	68	68	68	68	69	69	68	69	68	68	69	68	69	68	68	68	68	68	68
1728	69	68	69	68	69	68	68	68	69	68	69	68	68	68	69	68	68	68	68	68	69	68	68	68	69	69	69	68	69	68	68	68	69	68	68	69	68	68	69	68	68	69	68	69	68	69	68	69	68	68	69	68	68	69	68	68	69	68	69	68	68	69	68	68
1792	68	68	69	68	68	68	68	68	68	68	69	68	69	68	69	69	68	69	68	69	68	69	68	69	68	69	68	69	68	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	68	69	69	69	68	69	69	69	69	69	69	69	68	69	69	68	69	69	69
1856	69	69	69	69	69	69	69	68	69	69	69	68	69	69	69	68	69	69	69	69	69	68	69	69	69	69	68	69	68	69	69	69	68	68	69	68	68	68	69	68	68	68	69	68	68	68	69	68	68	68	68	68	68	68	68	68	68	68	68	68	69	68	68	68
1920	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	67	68	68	68	68	67	67	67	67	68	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67
1984	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	66	67	67	67	67	67	67	66	67	67	67	66	67	67	67	67	67	67	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	68	68	68	68	69	69	69	70	71	74	99

gflops vs. MPI Rank

Statistics Across Tasks

NERSC job details

19349.sdb&name=gflops&timestamp=1310766809

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

**9) - as a percentage of maximum

e.

IPM Examples

NERSC job details

http://www.nersc.gov/REST/jobs/mpi_functions.php?stepid=441560.sdb&f_name=MPI_Recv&timestamp=1306769707

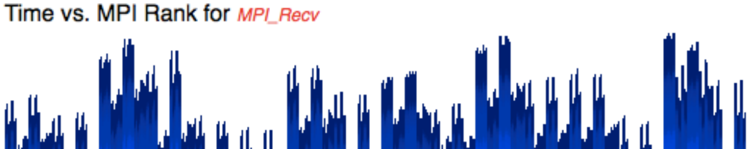
Time spent by each task in *MPI_Recv* as a percentage of the maximum value

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

Table Columns: 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	65	64	78	79	72	72	78	79	73	73	63	64	63	64	66	65	66	66
32	70	71	63	65	72	72	65	66	52	54	53	55	56	57	58	58	71	73	67	68	72	73	67	69	55	55	54	55	58	58	57	57
64	91	92	88	90	91	93	90	91	76	77	74	76	79	81	79	79	95	98	92	96	98	98	96	96	78	81	77	80	84	84	83	82
96	86	87	80	80	86	89	81	82	62	63	61	61	65	67	65	65	91	94	83	86	94	94	87	86	64	66	63	65	70	69	69	68
128	69	71	64	65	70	72	65	66	54	56	53	55	60	60	59	59	69	70	64	65	69	70	65	66	56	57	55	56	60	59	58	58
160	65	67	57	59	67	68	60	61	42	43	43	44	47	46	47	47	66	67	58	60	67	67	60	60	43	44	45	46	46	46	48	48
192	86	87	80	81	88	89	82	83	67	68	66	67	71	72	70	70	85	88	81	84	89	88	84	84	68	71	66	69	74	73	72	71
224	79	79	71	71	80	80	72	72	49	49	49	50	52	52	53	52	76	79	70	73	79	79	73	74	50	52	48	52	55	55	54	54
256	83	84	81	83	85	85	84	85	69	71	69	70	74	75	74	74	85	86	85	87	86	87	86	87	73	73	71	71	76	75	74	73
288	72	73	66	68	73	74	69	70	60	62	59	60	64	65	63	63	75	76	69	71	76	76	71	71	63	63	61	61	65	65	64	64
320	94	95	93	94	93	96	94	95	77	78	75	75	80	81	78	78	99	99	96	96	99	99	98	97	79	80	77	78	83	82	81	80
352	82	83	74	75	83	84	75	76	61	62	66	66	65	65	69	69	84	85	76	77	85	85	77	78	63	64	66	67	66	66	69	69
384	84	86	76	78	86	88	79	80	64	64	63	64	68	68	67	67	85	86	77	78	85	86	78	79	65	65	66	66	68	67	69	68
416	69	70	60	62	71	72	63	64	45	46	46	47	49	50	50	51	68	69	61	62	69	70	62	62	47	47	46	47	49	49	47	49
448	98	98	99	100	99	98	100	100	85	85	78	77	89	89	81	80	94	94	95	96	94	94	97	97	87	87	78	79	91	90	82	81
480	82	82	72	73	82	84	74	74	54	54	52	52	56	57	54	55	83	83	72	73	84	84	71	73	54	55	53	54	57	57	55	56
512	88	89	89	91	90	91	92	93	78	80	78	80	83	84	84	84	93	94	93	95	94	94	95	94	80	81	82	83	85	84	86	85
544	90	91	79	81	91	93	82	83	63	65	63	64	67	69	68	68	96	97	84	84	98	97	85	85	66	66	66	67	69	68	70	69
576	76	77	74	75	76	77	76	76	65	65	66	67	67	67	70	70	79	82	77	80	83	82	81	81	67	69	69	71	72	72	74	73
608	76	77	69	69	76	78	70	71	57	57	57	58	60	59	60	60	79	81	72	75	82	81	75	75	60	62	58	61	64	63	63	62
640	92	94	85	87	96	96	90	89	75	77	75	77	81	81	81	81	93	94	87	88	94	94	88	88	77	78	76	77	81	80	80	79
672	86	87	78	78	88	88	80	80	57	58	58	59	62	62	63	63	87	87	78	79	88	87	79	79	59	59	60	60	62	61	63	63
704	78	79	74	75	80	80	76	75	63	64	63	63	68	67	67	66	78	81	74	77	81	81	77	77	64	67	63	66	70	69	69	67
736	70	71	65	65	71	71	66	67	52	52	50	51	55	54	53	53	71	73	64	67	74	73	68	68	53	55	52	54	57	57	57	56
768	94	96	87	88	96	98	90	90	71	73	70	71	75	77	75	75	95	95	87	87	96	96	87	88	74	75	73	74	79	78	77	77
800	79	80	70	71	81	83	72	73	57	58	62	63	62	62	67	67	80	79	71	71	80	80	71	72	60	60	64	64	63	63	66	67
832	82	83	77	77	82	84	78	78	63	64	64	64	66	66	67	66	86	86	79	80	87	86	81	80	65	66	64	65	69	68	68	67
864	69	70	65	65	70	70	65	66	56	56	56	56	59	59	58	58	73	73	67	67	73	73	67	67	58	59	58	59	61	59	61	59
896	92	92	85	86	93	94	88	88	71	72	69	70	76	75	74	74	92	92	87	88	94	93	88	89	73	73	70	71	76	76	74	74
928	73	74	64	65	75	76	66	67	43	44	48	49	47	48	53	54	74	73	64	65	73	74	64	66	45	45	51	52	47	48	53	54
960	74	74	70	70	75	75	71	71	63	64	61	61	66	66	64	63	76	77	72	73	77	77	73	73	63	64	62	62	66	66	66	64
992	63	63	57	57	63	64	58	58	41	41	42	42	43	42	44	43	61	61	56	57	62	62	56	57	41	41	42	42	43	42	43	43

Time vs. MPI Rank for *MPI_Recv*

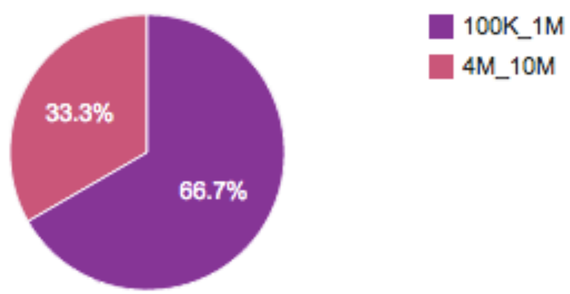


User-Space I/O Profiling

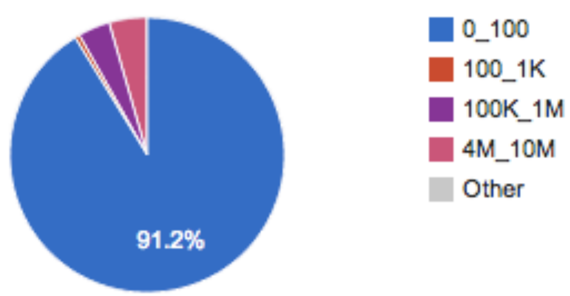
IO Summary from Darshan

Exec. Runtime	MB Read	MB Written	Read Time (s)	Write Time (s)	Read Rate (MB/s)	Write Rate (MB/s)
02-17 16:37:42 – 02-17 16:40:25	1909	18670.36	4.97671	212.215	383.59	87.98

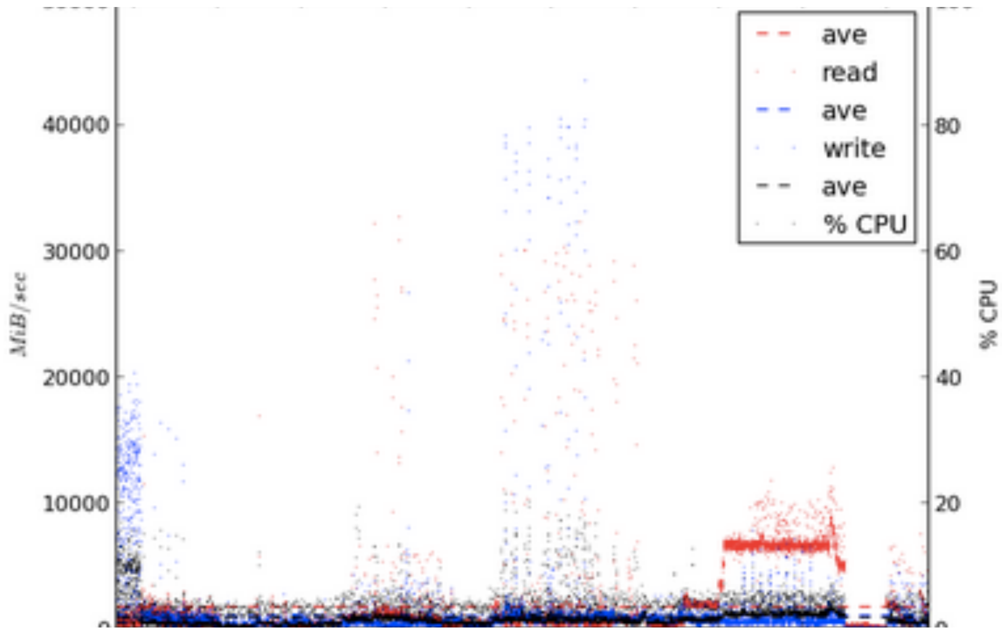
Number of Reads Per Size Range

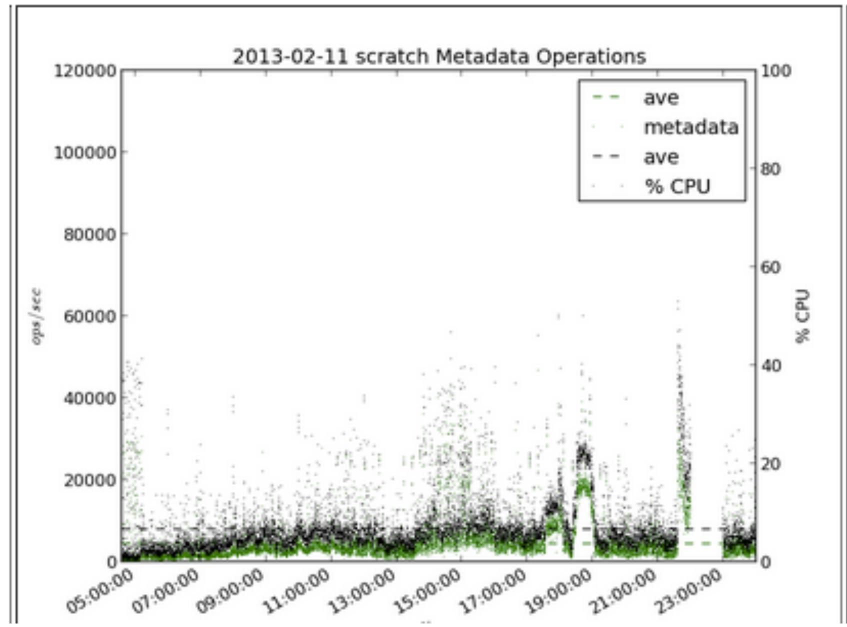


Number of Writes Per Size Range



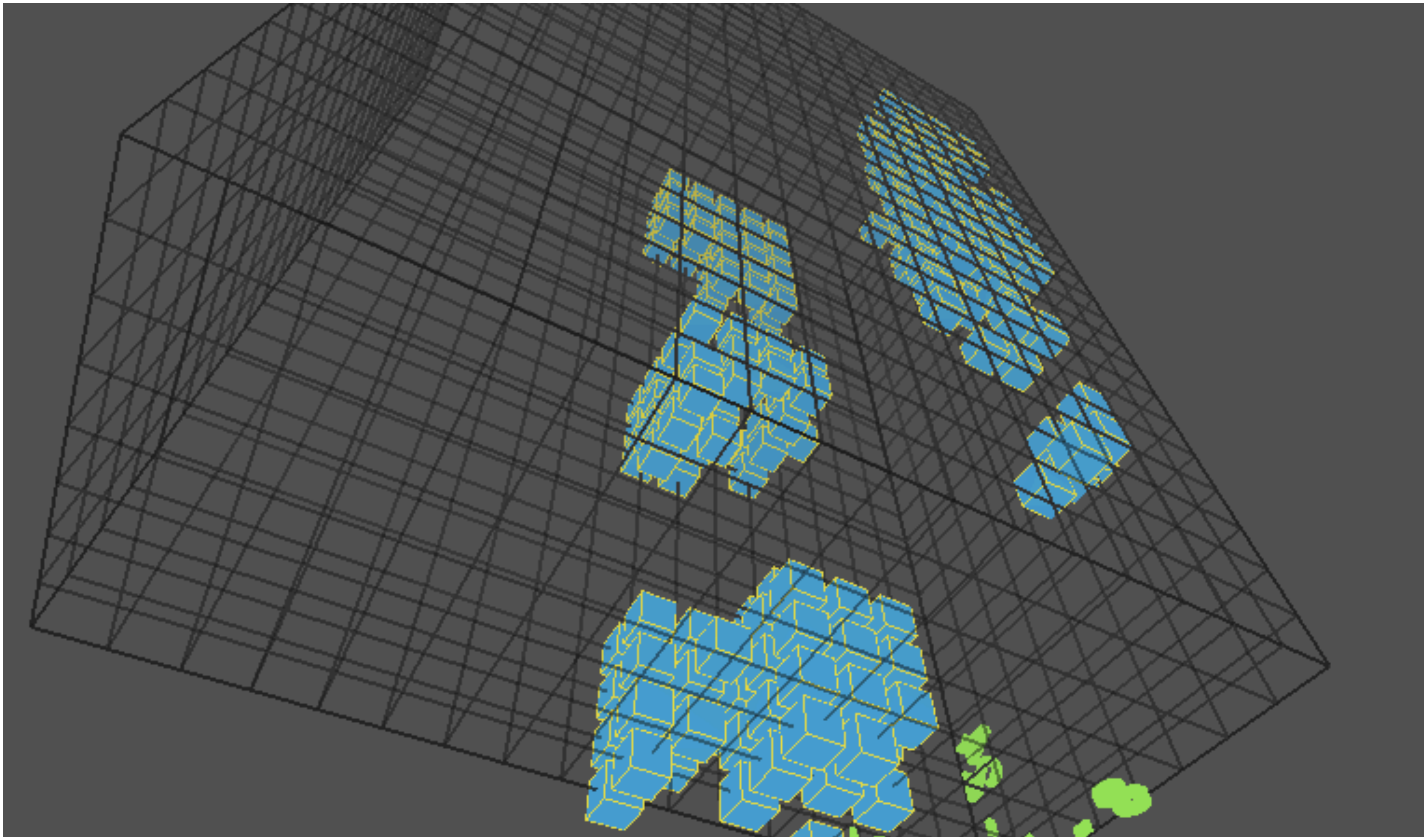
System-Level I/O Monitoring





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

Job Physical Topology



National Energy Research Scientific Computing Center