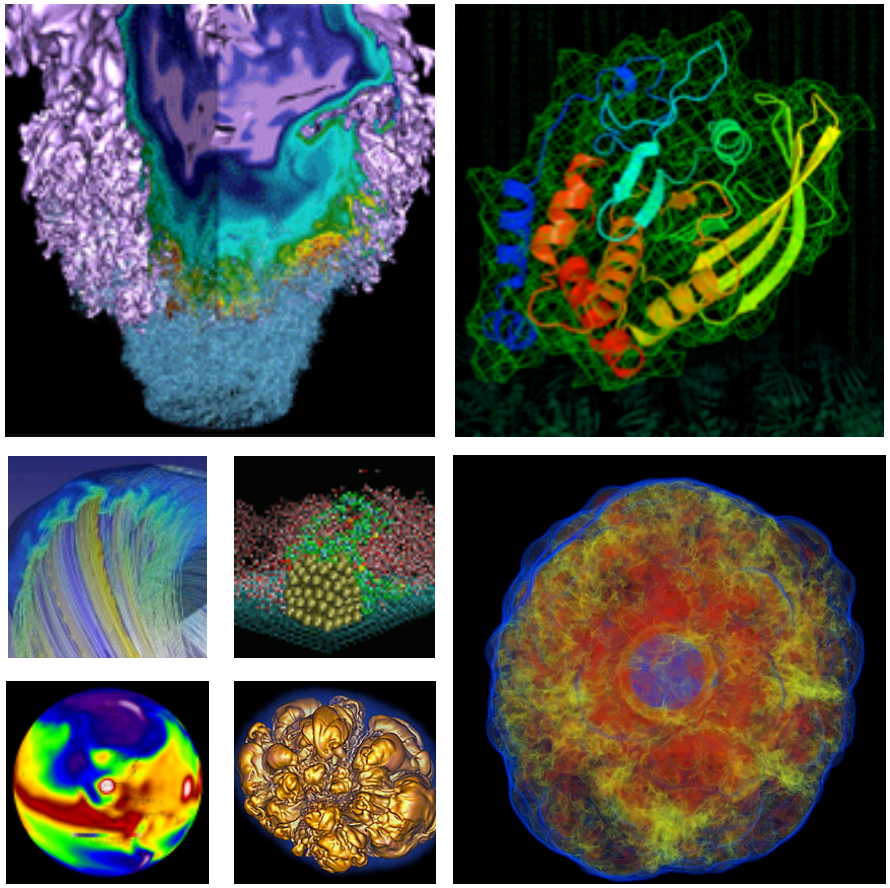
Debugging and Optimization Tools



Richard Gerber
NERSC
User Services Group Lead

Thanks to Woo-Sun Yang and Helen He

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Outline

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

Videos, presentations, and references:

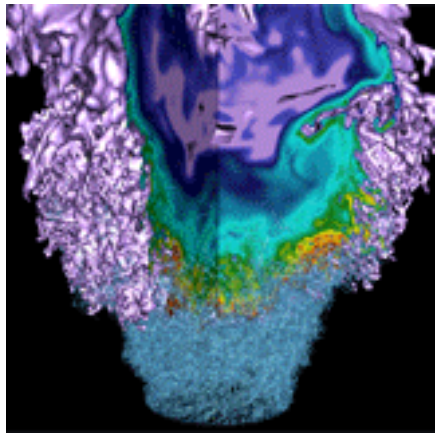
http://www.nersc.gov/users/training/courses/CS267/

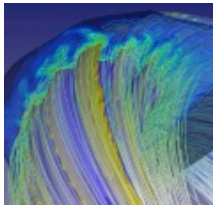
Help at NERSC
consult@nersc.gov
https://help.nersc.gov

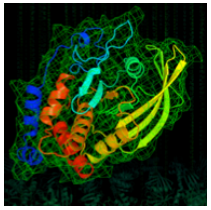
Take-Aways

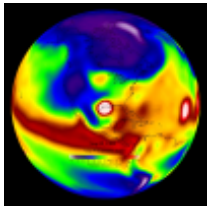
- **Tools can help you find errors in your program and locate performance bottlenecks**
- **In the world of HPC parallel computing, there are few widely adopted standard tools – try some and see what works for you**
  - Totalview and DDT debuggers
  - PAPI, Tau, various research tools, & vendor-specific performance tools
- **Common code problems**
- **How tools work in general**
- **Be suspicious of outliers among parallel tasks**
- **Where to get more information**

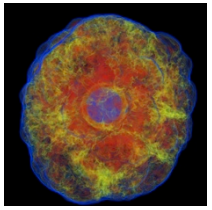
Debugging

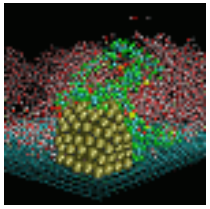






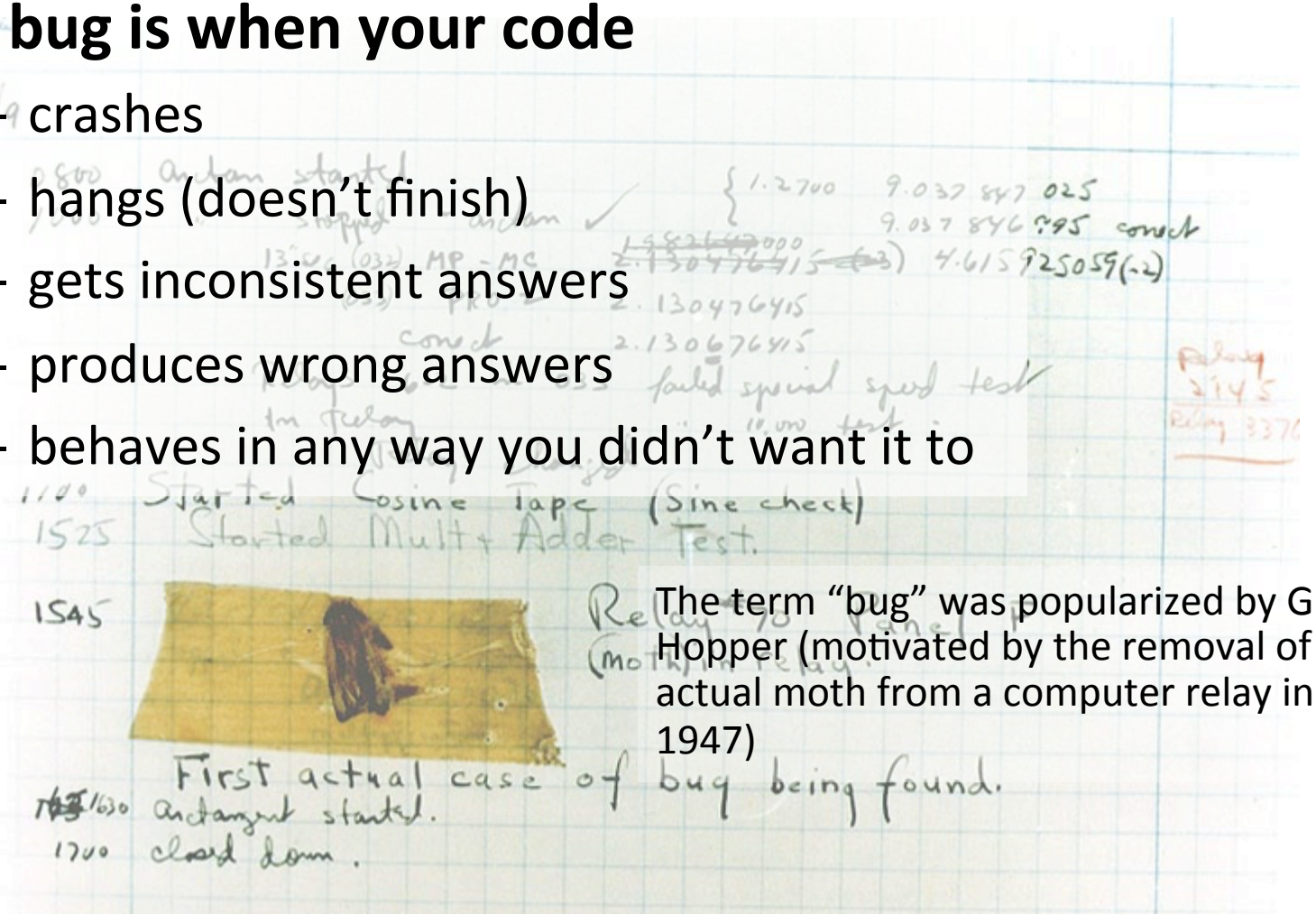






What is a Bug?

- **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

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/

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);

}
```

NERSC NX – Accelerate Your X Connection

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.

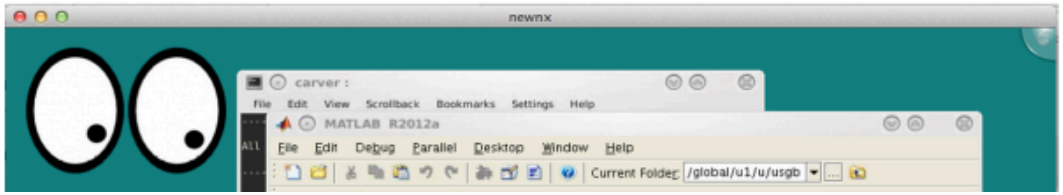


TABLE OF CONTENTS

Related Information

Download Tested NX Player

Download NX Configuration File

NX FAQ

Compile & Start DDT

Compile for debugging

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

Set up the parallel run environment

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

Start the DDT debugger

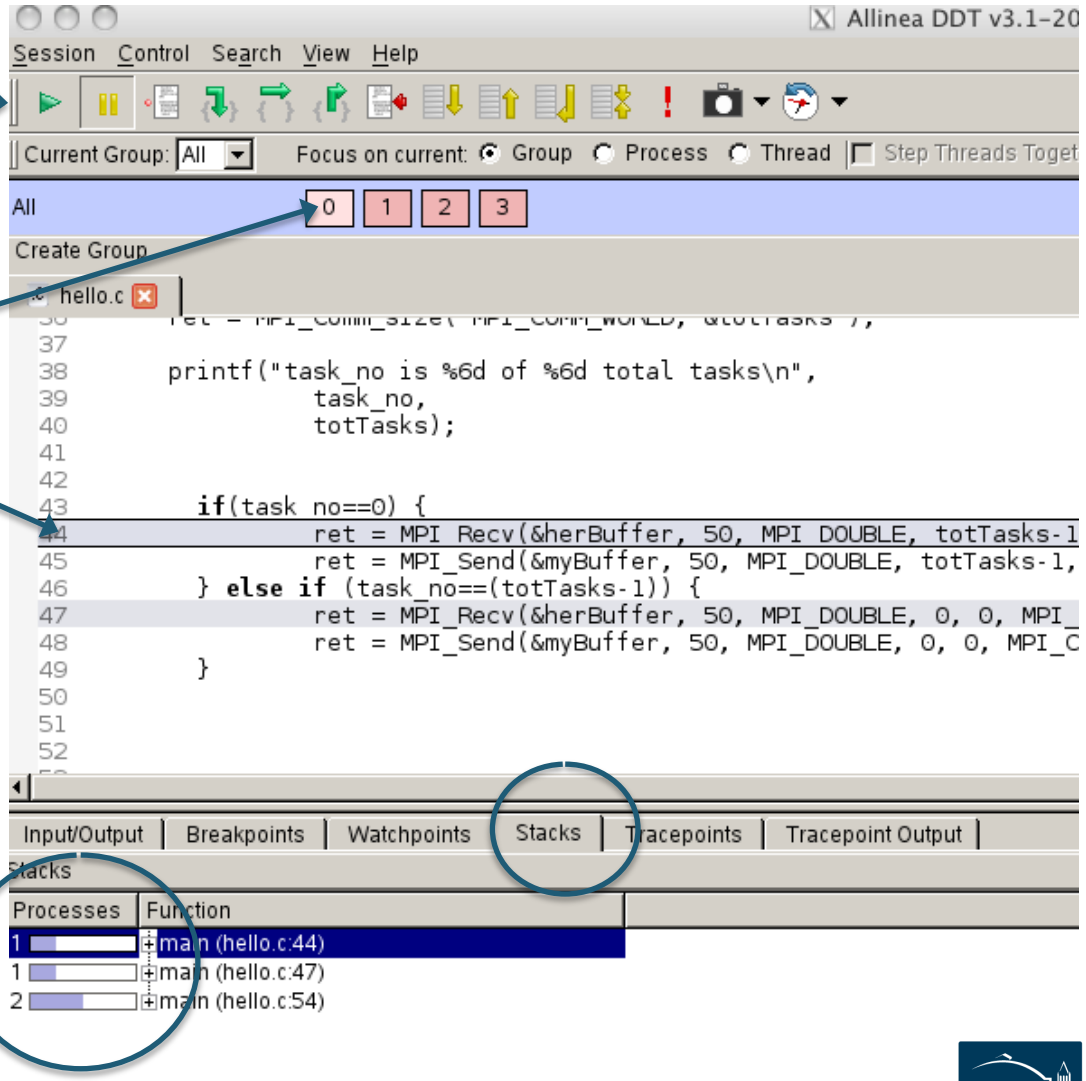
```
edison% module load ddt
edison% ddt ./hello
```

DDT Screen Shot

Press Go and then Pause when code appears hung.

Task 0 is at line 44

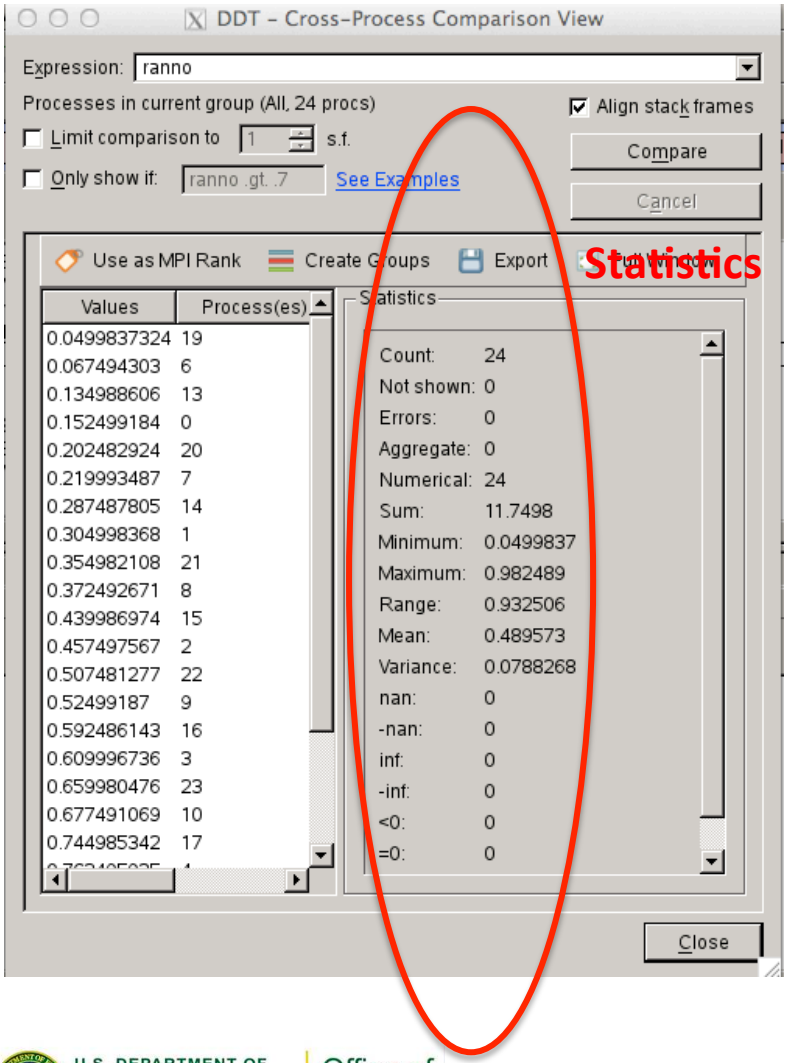
At hang, tasks are in 3 different places.

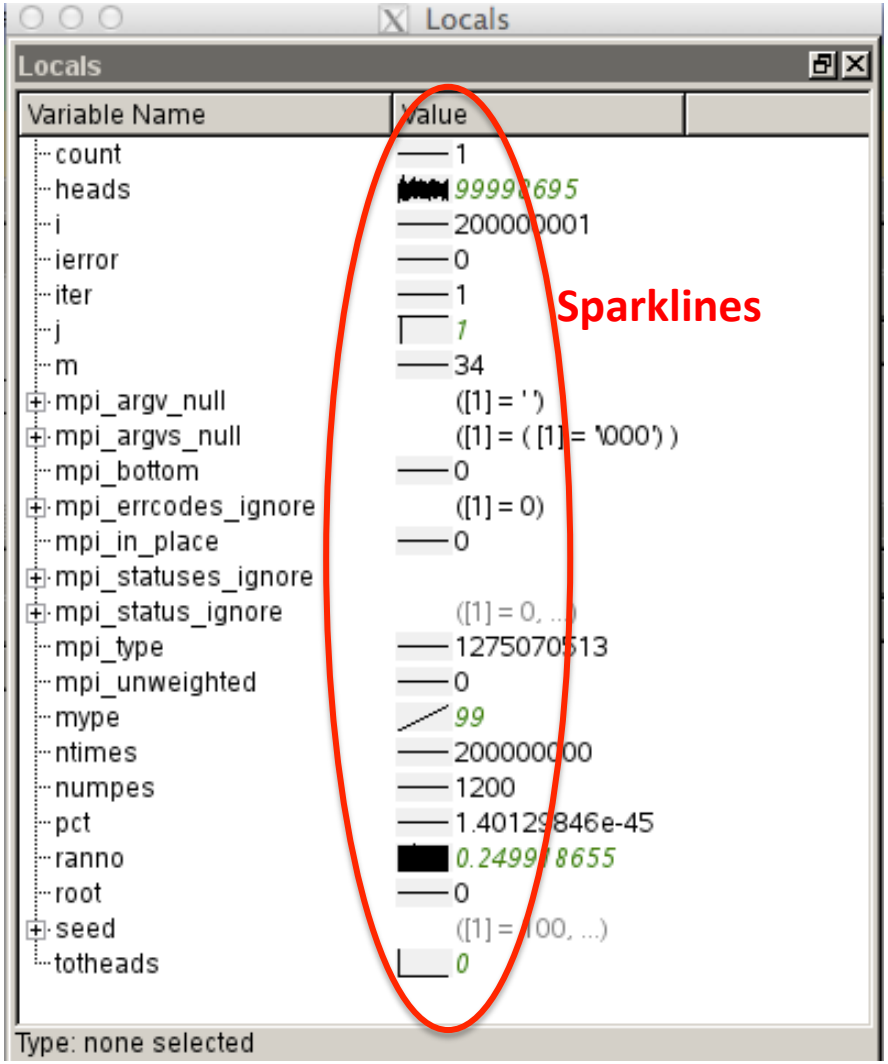


What About Massive Parallelism?

- **With 10K+ tasks/threads/streams it's impossible to examine every parallel instance in detail**
- **Make use 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

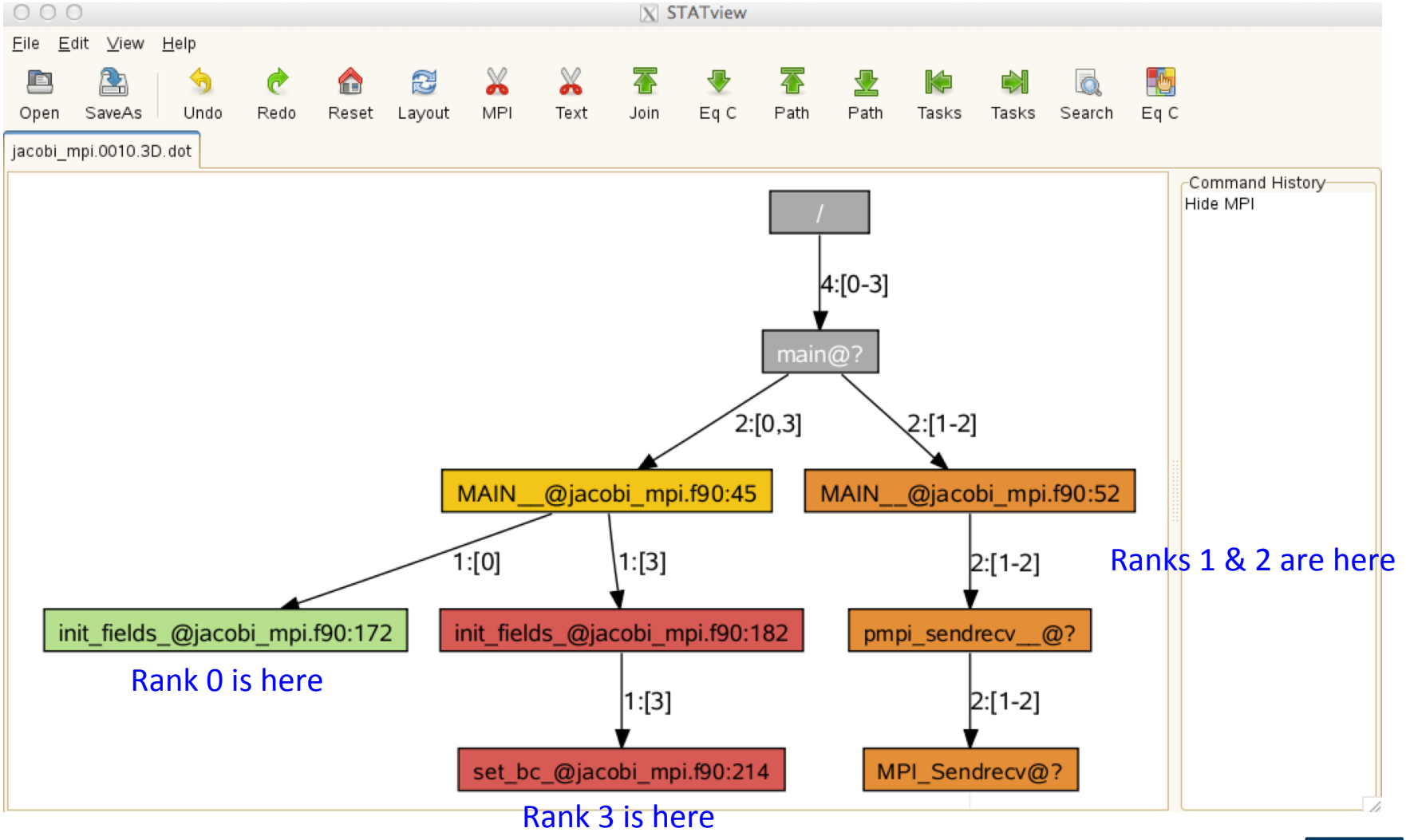
- **Parallel debuggers with a graphical user interface**
  - DDT (Distributed Debugging Tool)
  - TotalView
- **Specialized debuggers on Hopper and Edison**
  - STAT (Stack Trace Analysis Tool)
    - Collect stack backtraces from all (MPI) tasks
  - ATP (Abnormal Termination Processing)
    - Collect 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
  - https://computing.llnl.gov/code/STAT/stat_userguide.pdf
  - http://www.nersc.gov/users/software/debugging-and-profiling/stat-2/

Hung application with STAT (Cont'd)



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
  - http://www.nersc.gov/users/software/debugging-and-profiling/gdb-and-atp/

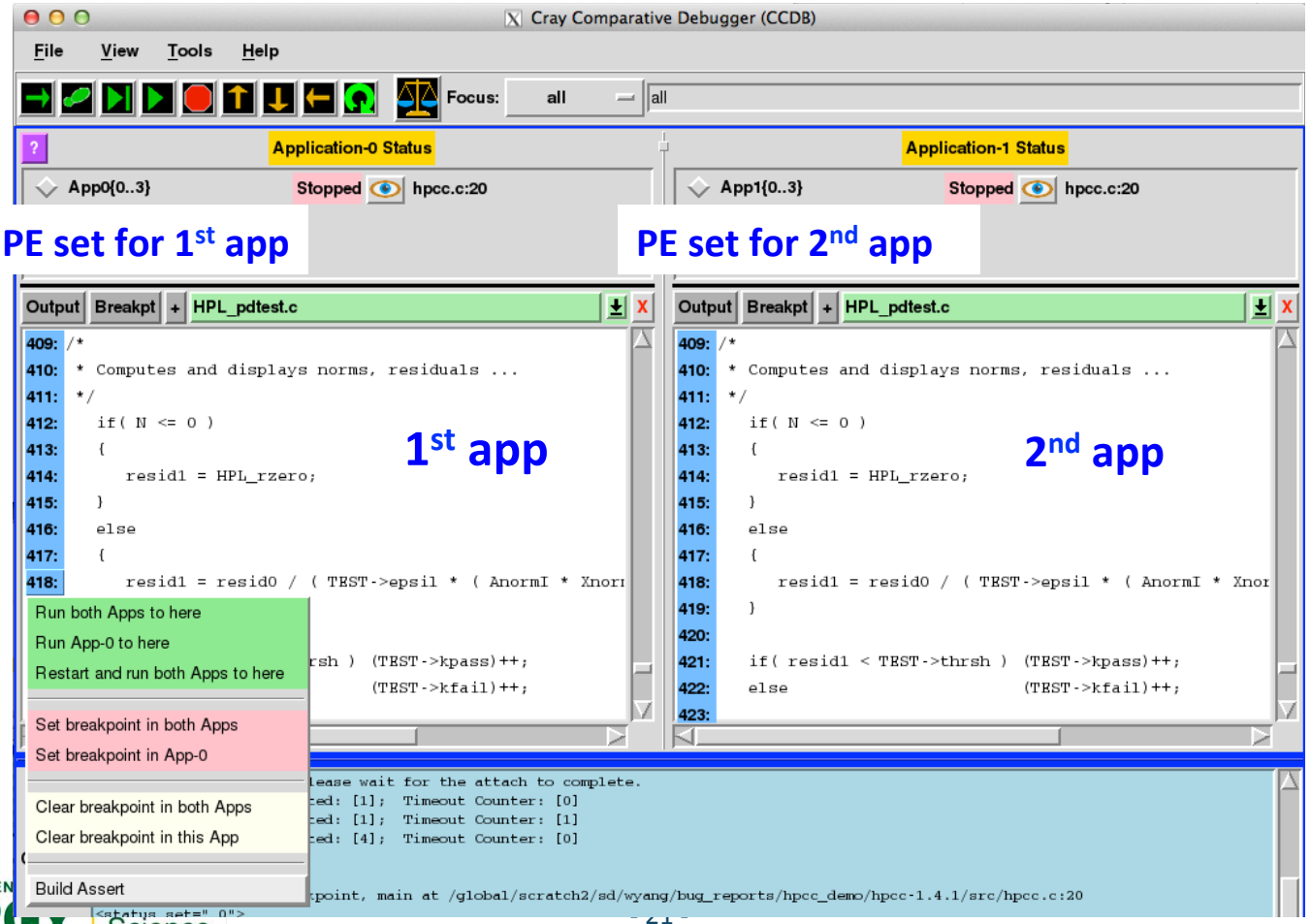
CCDB (Cray Comparative Debugger)

- **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
  - '*Using the lgdb Comparative Debugging Feature*', http://docs.cray.com/books/S-0042-22/S-0042-22.pdf
  - http://www.nersc.gov/users/software/debugging-and-profiling/ccdb-lgdb/ (work in progress)

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

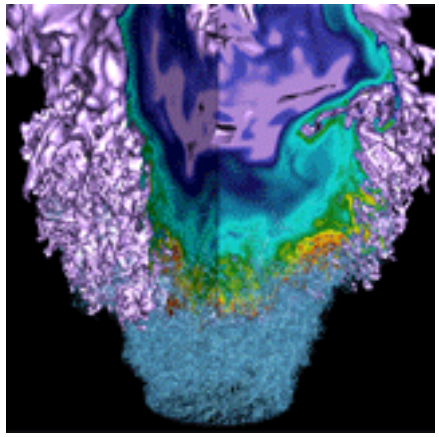


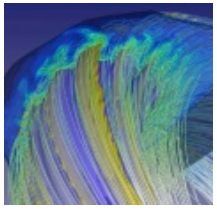
Slide Courtesy of Woo-Sun Yang

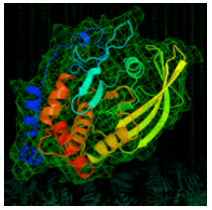
Valgrind

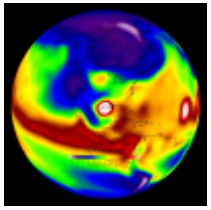
- **Suite of debugging and profiler tools**
- **Tools include**
  - **memcheck**: memory error and memory leaks detection
  - **cachegrind**: a cache and branch-prediction profiler
  - **callgrind**: a call-graph generating cache and branch prediction profiler
  - **massif**, **dhat (exp-dhat)**: heap profilers
  - **helgrind**, **drd**: pthreads error detectors
- **For info:**
  - http://valgrind.org/docs/manual/manual.html

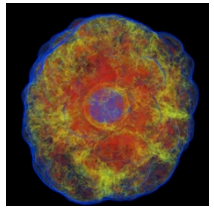
Performance / Optimization

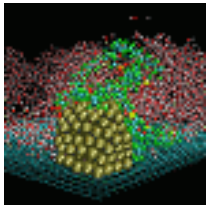












Performance Questions

- 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?

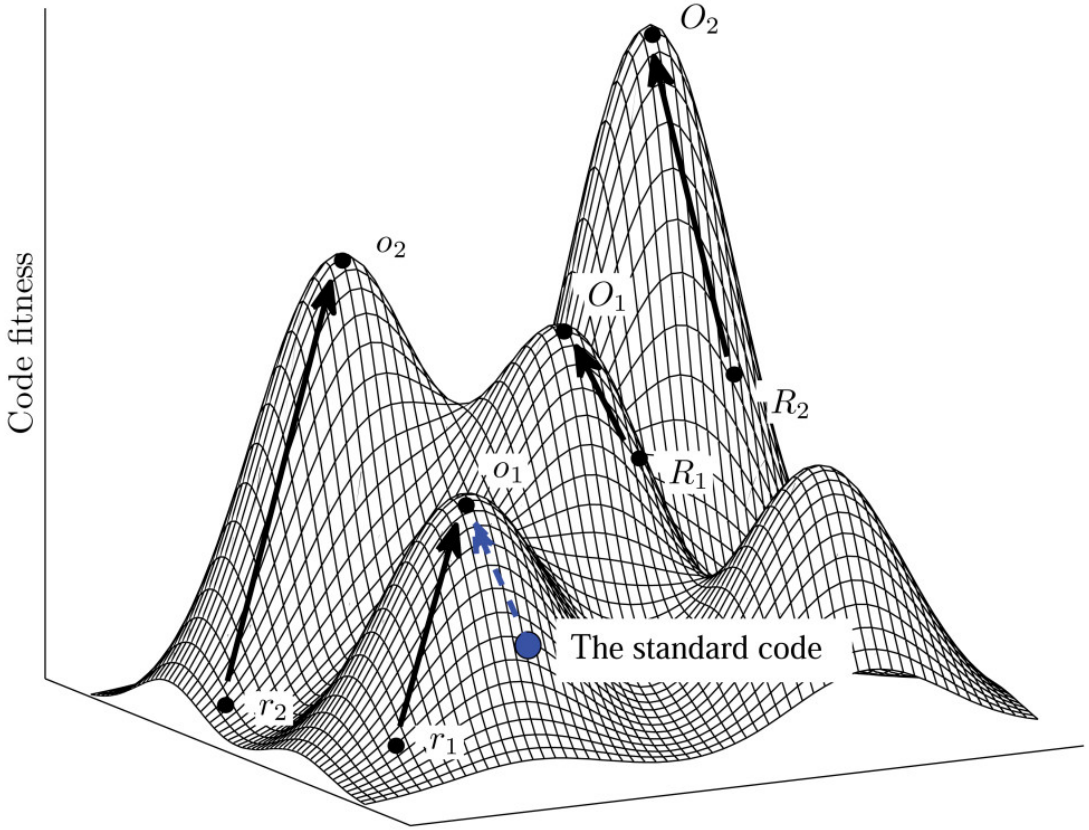
Is Your Code Performing Well?

- **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% of theoretical is pretty good
  - GPUs, Xeon Phi: >few% 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**
- **Take advantage of vector (SIMD) capabilities**
- **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
  - HPC Toolkit
  - Scalasca
- **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 that can be viewed with apprentice GUI application

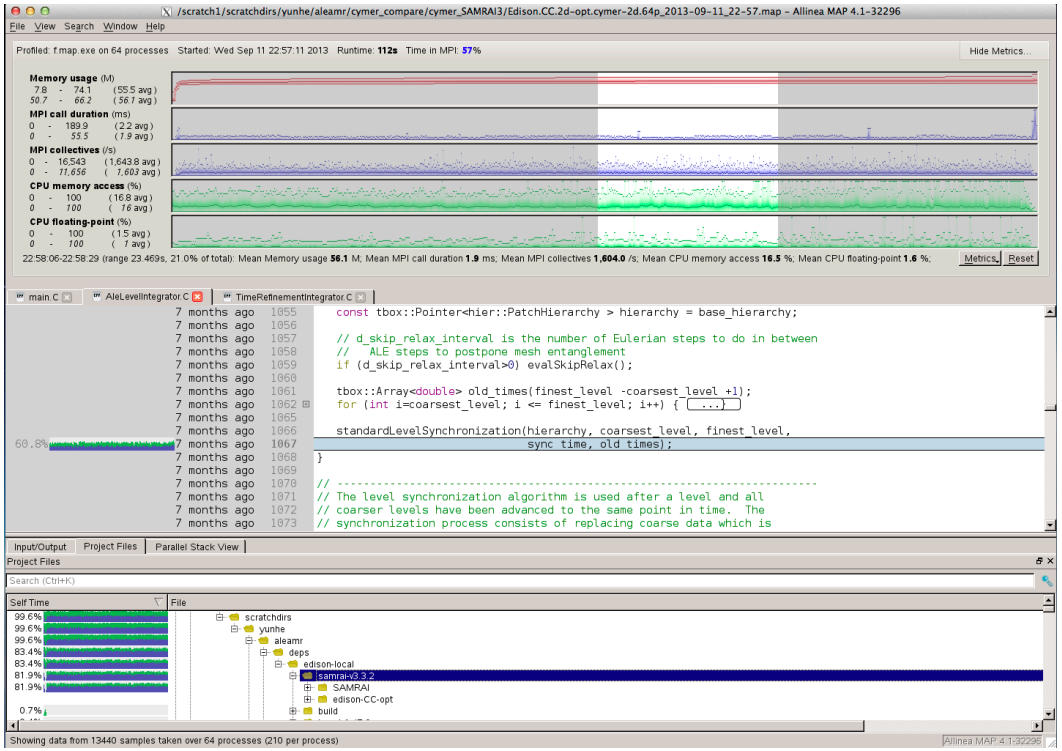
Cray perftools and perftools-lite

- **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:**
  - https://www.nersc.gov/users/software/debugging-and-profiling/craypat/
  - http://www.nersc.gov/assets/Uploads/UsingCrayPat.pdf
  - http://www.nersc.gov/assets/Training-Materials/UsingApprentice2012.pdf
  - http://www.nersc.gov/assets/Uploads/Perftools-lite-2013.pdf

```
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:    290.822940 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_
...
||==========================================================
|  29.6% |  8432.1 |    -- |     -- |MPI
||----------------------------------------------------------
|| 9.0% |  2571.0 | 129.0 |   4.8% |MPI_GATHERV
```

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.**



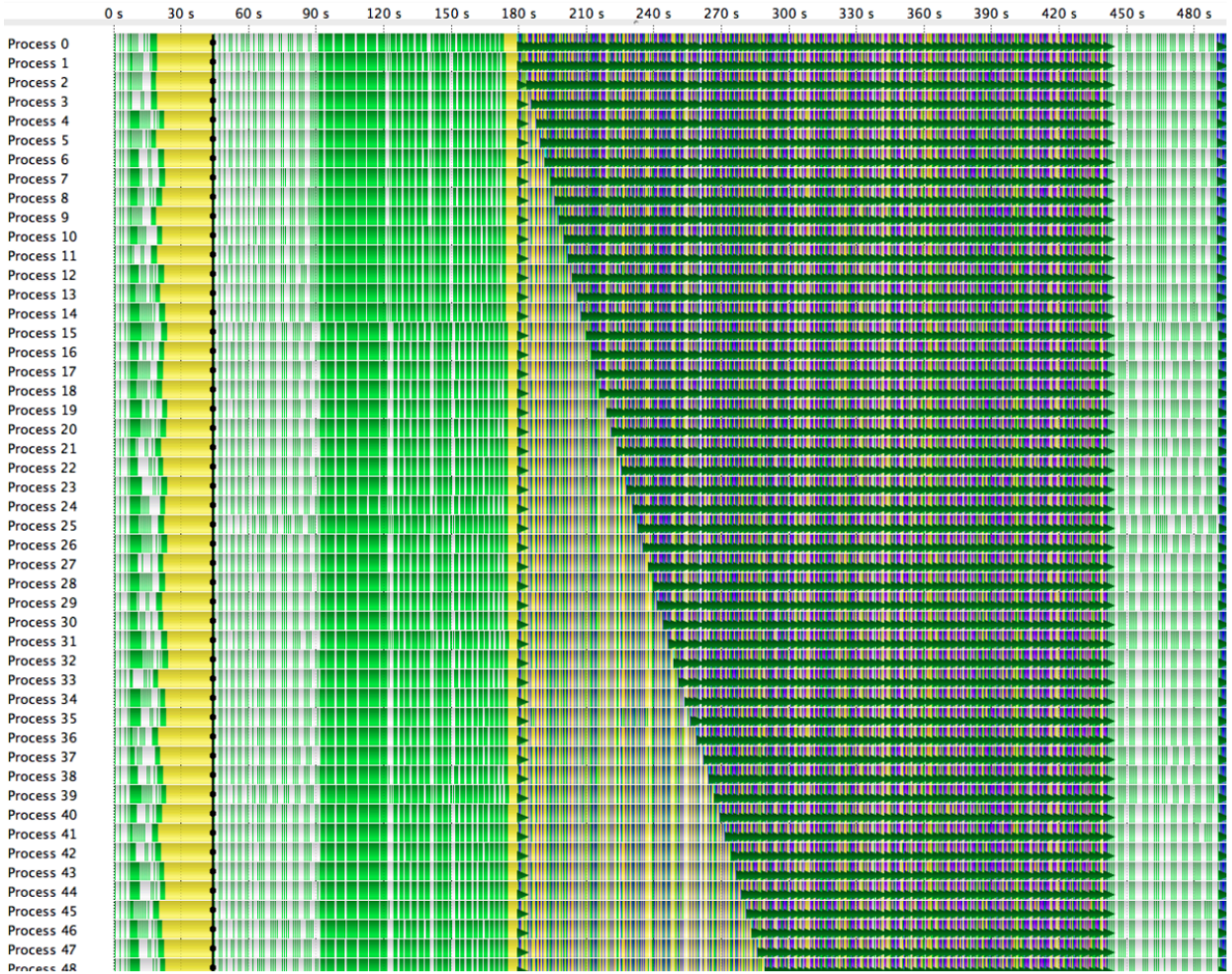
Documentation:

http://www.nersc.gov/users/software/debugging-and-profiling/MAP/

http://www.allinea.com/products/map/

Slide Courtesy of Helen He

Vampir and Vampirtrace



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

Show 50 entries Search:

#	Host	JobID	Job Name	User	Nds	Complete	Wall hrs
0	edison	782557	my_job	hdixit	11	02/17/14 13:14	0.160
1	edison	782431	cp_ref_big	hsinyu	13	02/17/14 13:14	0.849
2	edison	782591	tvsoi	wangyu	2	02/17/14 13:14	0.005
3	edison	782560	my_job	hdixit	11	02/17/14 13:13	0.131
4	edison	719618	nacl_big	carnevav	16	02/17/14 13:12	11.453
5	edison	782403	slab-Ni-pa... [Full Name]	hdixit	11	02/17/14 13:12	1.006
6	edison	782590	NL_test	khkim	86	02/17/14 13:12	0.008

APID	Command	Nodes	Tasks	Threads per Task	Tasks per Node	Max Task Mem (MB)	Run Time (secs)	St
3140812	gribmeanp.x	3	64	1	24	235	5	
	Command line:	/opt/cray/alps/default/bin/aprun -n 64 -N 24 /scratch1/scratchdirs/whitaker/edison/newstuff/bin/gribme /scratch1/scratchdirs/whitaker/gfsenkf_t126_1999iau_biasc						
	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 : 29.9751 sec
# stop    : 11/30/04/14:36:00              %comm     : 27.72
# gbytes  : 6.65863e-01 total              gflop/sec : 2.3347 e+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
```

IPM Data 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	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	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	69	68	68	68	68	68	68	68	69	68	68	68	69	68	68	69	68	68	68	68	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	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	69	68	68	68	68	68	68	68	68	68	68	68
320	68	68	68	68	68	68	69	68	68	69	68	68	69	68	68	68	68	68	68	69	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	68	67	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	67	67	67	67	67	67	67	67	67	67	67	66	67	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	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
512	100	74	72	71	70	69	69	69	69	68	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	67	67	67	67	67	67	67	67	67	67	67	67	67
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	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
640	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	69	70	69	69	70	69	69	70	70	69	70	69	69	70	69	70	69	69	70	69	70	69	69	69	70	69	69	70	70	70	70	69	70	69	70	70	69	70	69	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	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	67	67	67	67	67	67	67	67	67	67	67	67	67	67
832	70	70	70	69	70	69	70	69	69	70	69	69	70	69	70	69	69	69	69	69	70	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	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	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
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	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	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	67	67	67	67	67	67	67	67	67	67	67	67	67
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	69	69	68	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
1152	69	69	69	69	69	69	69	69	69	69	69	69	69	70	70	69	70	69	70	69	69	70	69	70	69	70	70	69	70	69	69	70	69	70	69	70	69	70	69	70	70	70	70	70	70	69	70	70	69	70	70	70	70	70	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	69	70	70	70	70	70	70	69	70	69	70	70	69	70	69	70	69	69	70	70	70	69	70	70	70	70	70	70	70
1280	69	69	70	69	69	70	70	69	70	70	70	69	70	69	70	70	69	70	69	70	69	70	69	70	70	69	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	69	70	70	70	70	70	69	70	70	70	70	69	70	70	69	70	69	69	70	70	69
1344	70	70	70	70	70	70	70	69	69	70	69	69	70	69	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	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	68	68	67	68	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
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	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	65	65	66	66	66	66	66	66	66	66	66	66	66	66
1600	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	67	66	66	67	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	67	67	67	67	67	67	67	67	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	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68	68
1728	69	68	69	68	69	68	68	68	68	69	68	69	68	68	68	68	69	68	68	68	68	68	69	68	68	68	68	69	69	68	69	68	68	68	69	68	68	69	68	68	68	68	69	68	68	68	69	68	68	68	69	68	68	68	69	68	68	68	69	68	68	68	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	69	68	69	68	68	69	68	69	68	69	68	69	68	69	68	69	68	68	69	68	69	69	69	69	69	69	69	69	69	68	69	69	69	69	69	68	69	69
1856	69	69	69	69	69	68	69	69	69	69	68	69	69	69	69	68	69	69	69	68	69	69	68	69	69	68	69	68	68	69	68	69	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	68	68	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	68	68	67	68	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	67	67
1984	66	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	66	67	67	67	67	67	67	66	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	67	68	68	68	68	69	69	70	71	74	99

gflops vs. MPI Rank

IPM Examples

NERSC job details

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

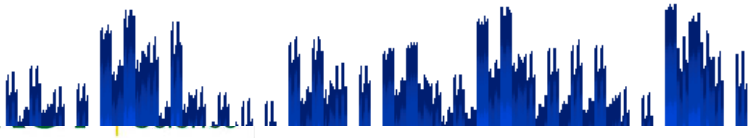
Time spent by each task in *MPI_Recv* as a percentage of the maximum value

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*



Cori – NERSC's next supercomputer system

- NERSC will be installing a 30-petaflop/s Intel KNL-based Cray system in the 2016 time frame, named after American Biochemist Gerty Cori
- Over 9,300 single-socket nodes in the system with each node > 3TeraFLOPS/s theoretical peak performance
- Cray Aries high speed "dragonfly" topology interconnect
- Cabinets, Liquid cooling
- Lustre filesystem with > 430 GB/sec I/O bandwidth and 28 PB of disk capacity; 1-2 TB/sec Burst Buffer bandwidth

Gerty Cori

Intel Xeon Phi Knights Landing Processor overview

- Single socket node with NUMA
- Greater than 60 cores per node with up to 4 hardware threads each
- AVX512 Vector pipelines with a hardware vector length of 512 bits (eight double-precision elements)
- Better performance per watt than previous generation Xeon Phi™ systems and 3X single-thread performance
- 64-128 GB of DRAM memory per node
- On-package, high-bandwidth memory, up to 16GB capacity with bandwidth projected to be 5X that of DDR4

NESAP Tools Partners

- HPCToolkit
  - http://hpctoolkit.org/
- TAU
  - http://www.cs.uoregon.edu/research/tau/home.php
- Open|Speedshop
  - http://www.openspeedshop.org/wp/
- PAPI
  - http://icl.cs.utk.edu/papi/
- MAP (Allinea)
  - http://www.allinea.com/products/map
- Vampir
  - https://www.vampir.eu/
- PerfExpert
  - https://www.tacc.utexas.edu/research-development/tacc-projects/perfexpert
- Scalasca
  - http://www.scalasca.org/
- VTUNE
  - https://software.intel.com/en-us/intel-vtune-amplifier-xe
- TotalView
  - http://www.roguewave.com/products-services/totalview
- DDT (Allinea)
  - http://www.allinea.com/products/ddt

Challenges with debugging and profiling on Cori

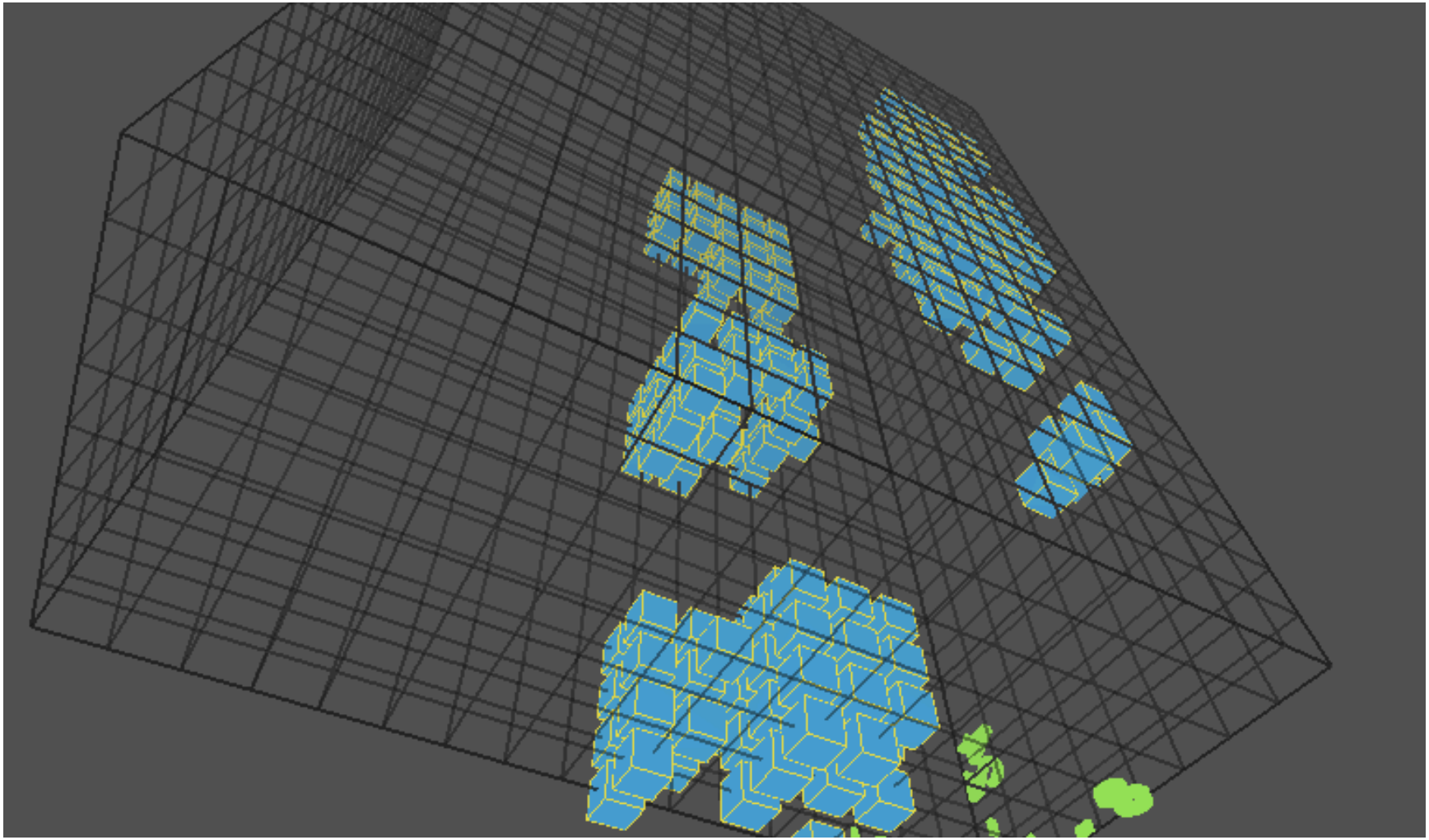
- **Debugging and performance tools need to scale up to thousands (or more?) of threads and tasks**
- **NESAP participants will want to know**
  - Hotspots
  - Memory usage, bandwidth and errors with on-package memory
  - Cache utilization
  - MPI communication performance - same interconnect fabrics for more powerful nodes
  - Threading performance
  - I/O statistics
  - …

Summary

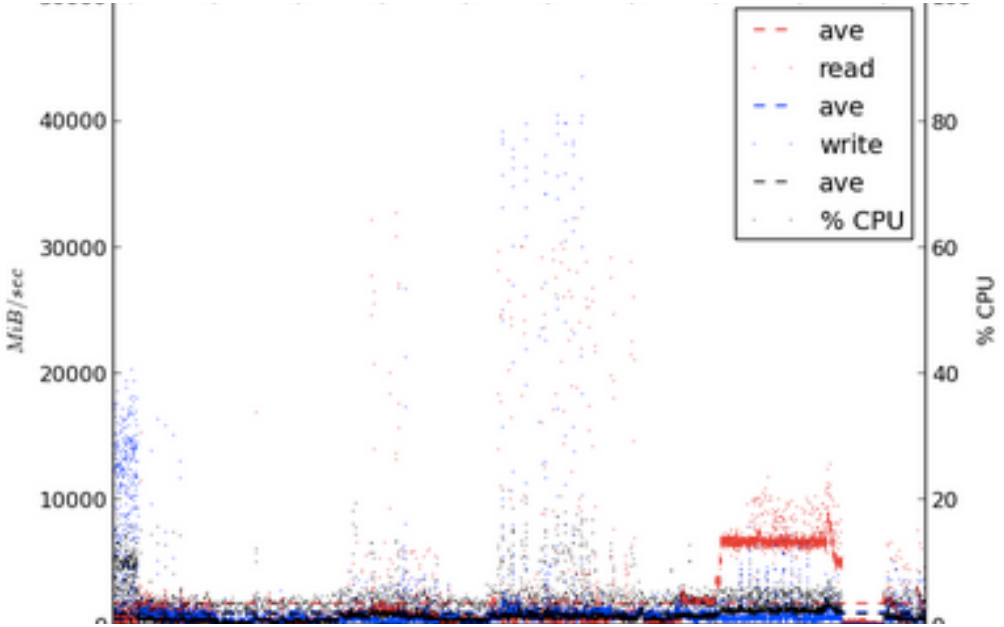
- **Debugging and Parallel Code Optimization can be hard**
- **Tools can help**
  - See NERSC web pages for recommendations
  - Use the ones that work for you
- **Be aware of some of the more common errors and best practices**
- **Look for outliers in parallel programs**
- **Refer to NERSC web pages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

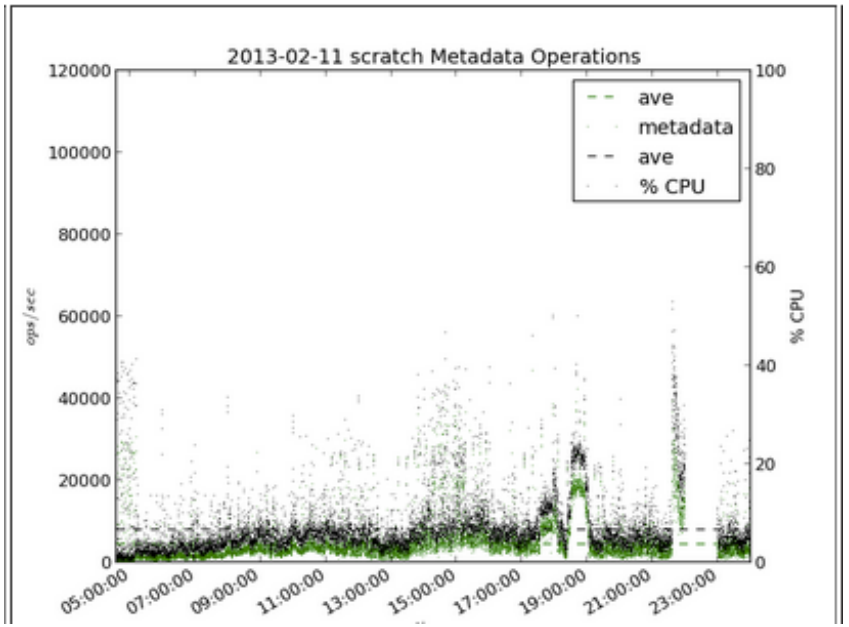
National Energy Research Scientific Computing Center

Job Physical Topology



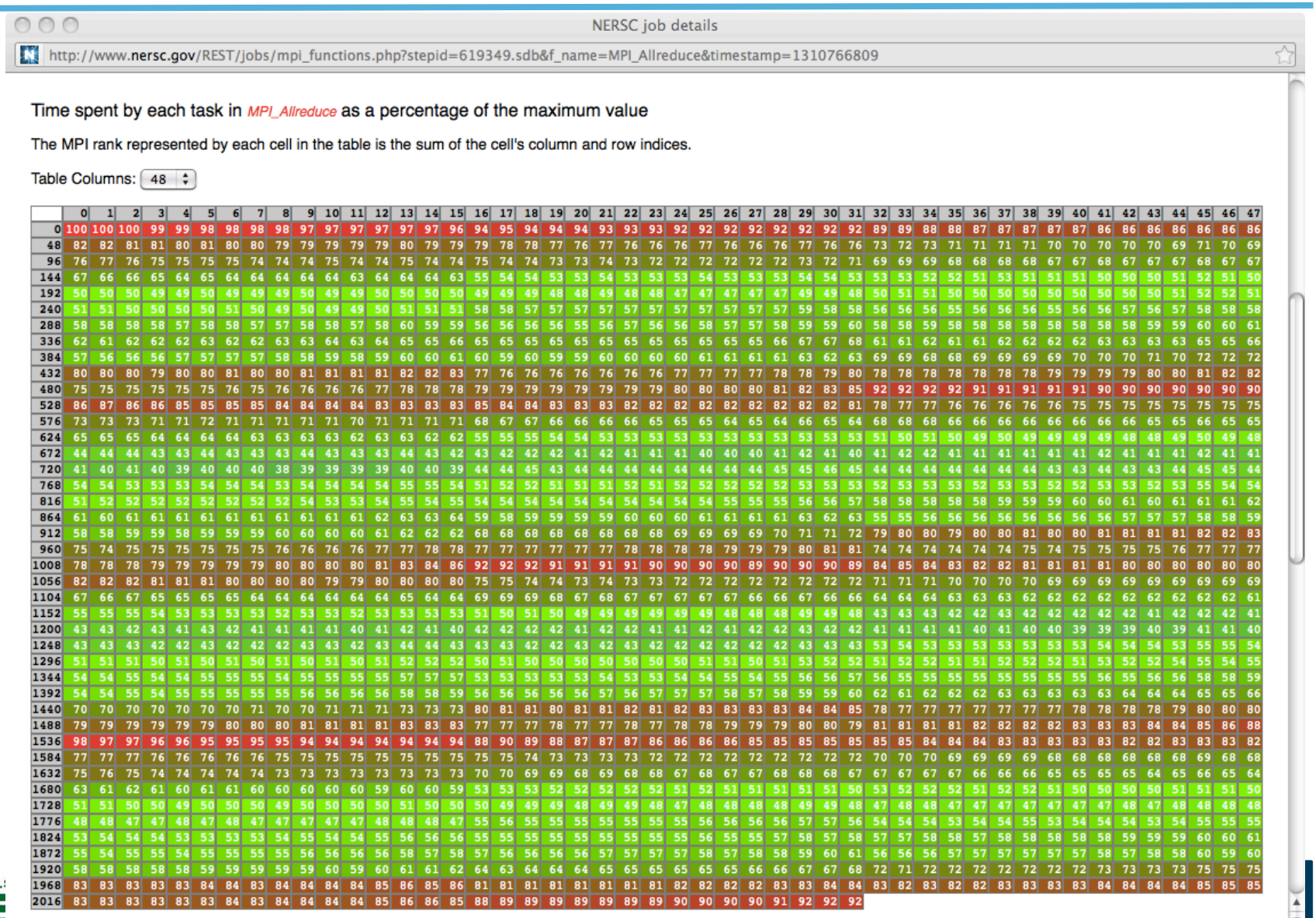
System-Level I/O Monitoring



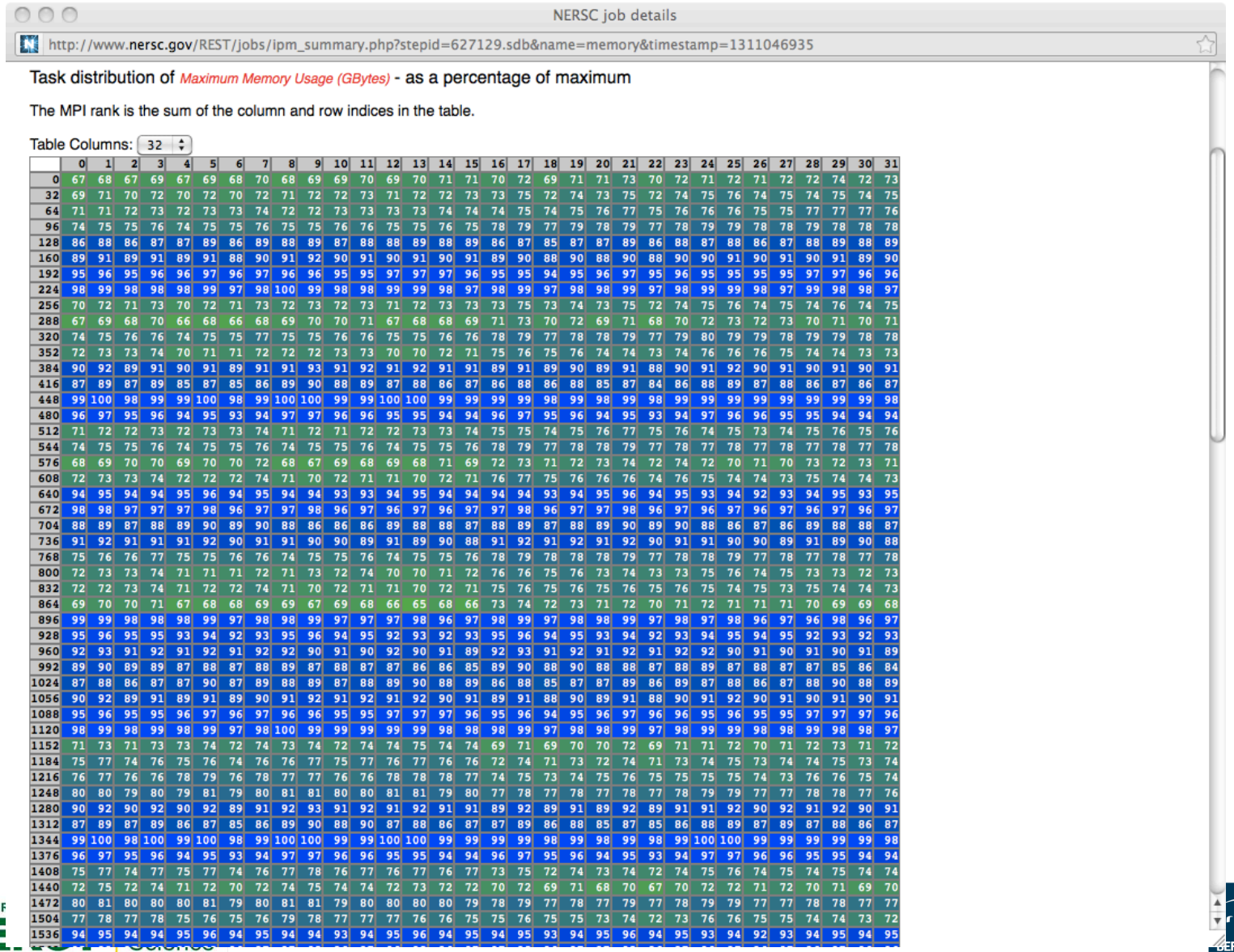


Users can see the system-wide I/O activity while their job ran to look for contention.

IPM Examples



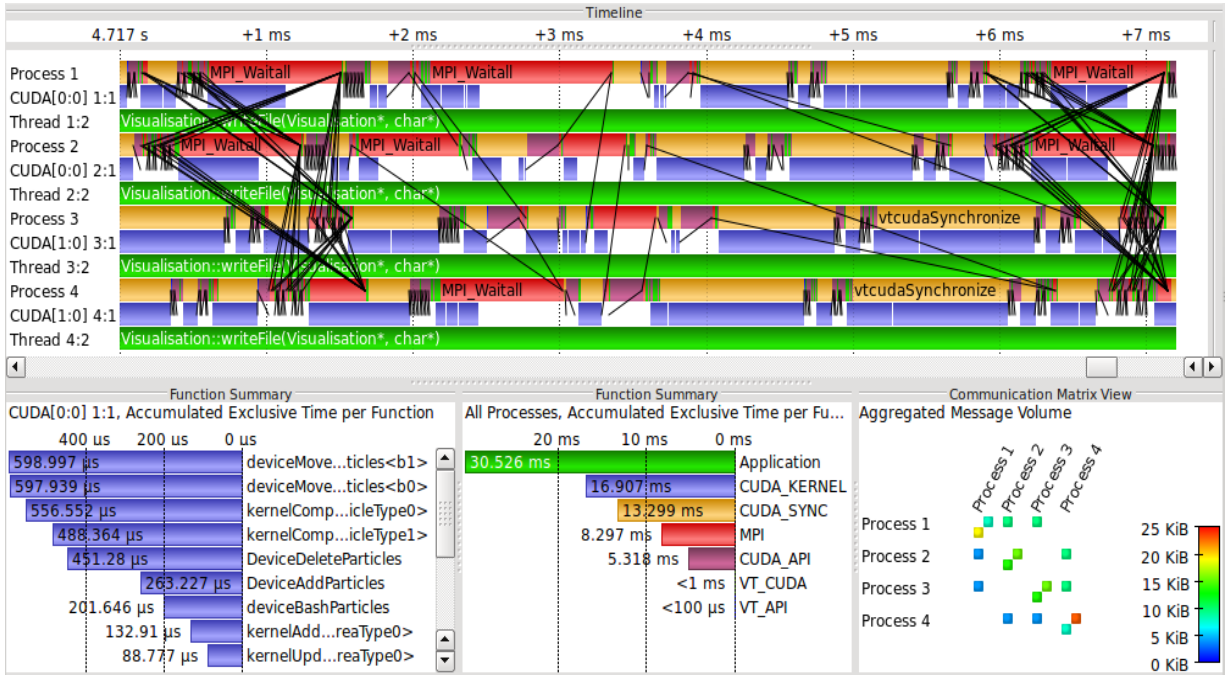
IPM Examples



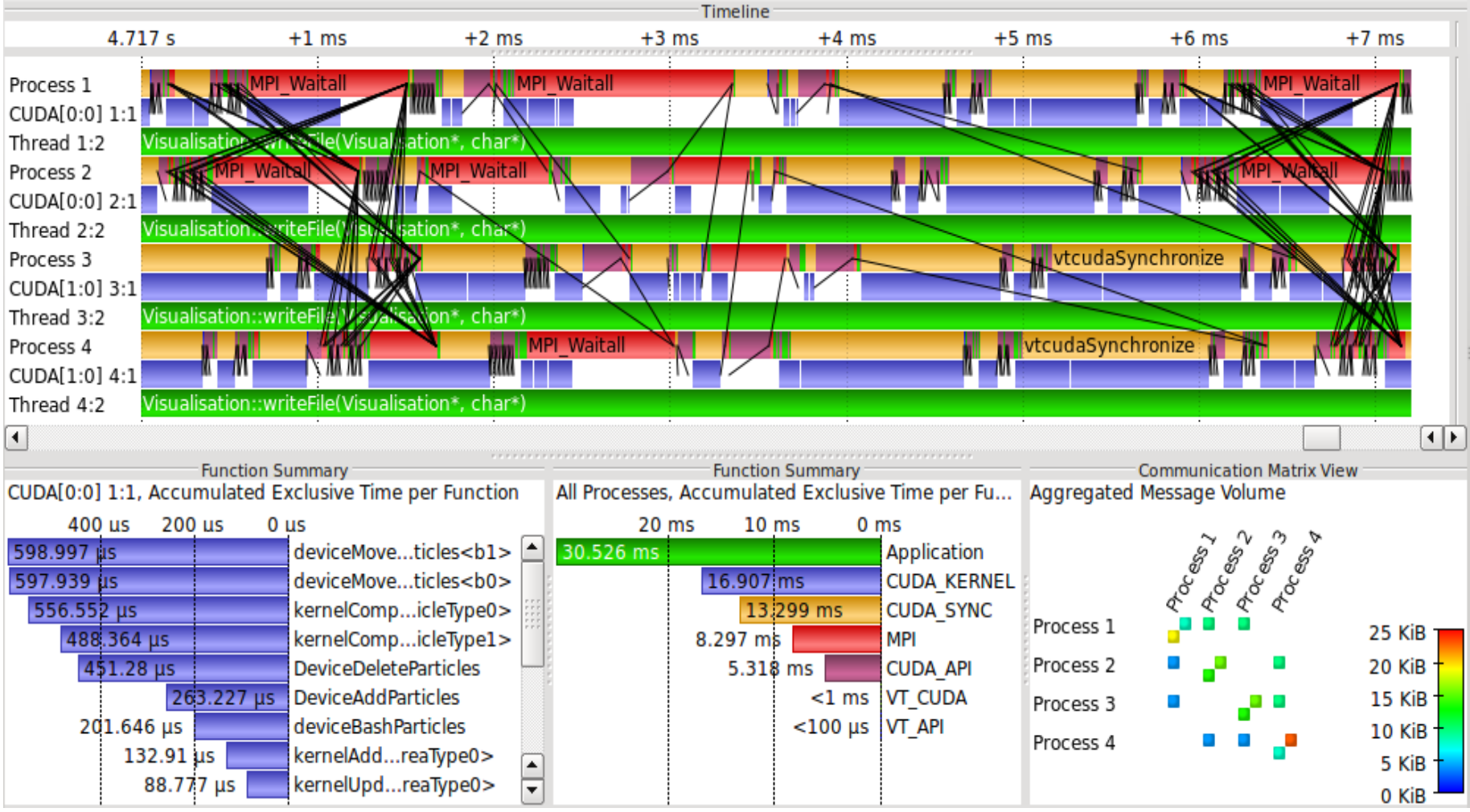
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	67	68	67	69	67	69	68	70	68	69	69	70	69	70	71	71	70	72	69	71	71	73	70	72	71	72	71	72	72	74	72	73
32	69	71	70	72	70	72	70	72	71	72	72	73	71	72	72	73	73	75	72	74	73	75	72	74	75	76	74	75	74	75	74	75
64	71	71	72	73	72	73	73	74	72	72	73	73	73	73	74	74	74	75	74	75	76	77	75	76	76	76	75	75	77	77	77	76
96	74	75	75	76	74	75	75	76	75	75	76	76	75	75	76	75	78	79	77	79	78	79	77	78	79	79	78	78	79	78	78	78
128	86	88	86	87	87	89	86	89	88	89	87	88	88	89	88	89	86	87	85	87	87	89	86	88	87	88	86	87	88	89	88	89
160	89	91	89	91	89	91	88	90	91	92	90	91	90	91	90	91	89	90	88	90	88	90	88	90	90	91	90	91	90	91	89	90
192	95	96	95	96	96	97	96	97	96	96	95	95	97	97	97	96	95	95	94	95	96	97	95	96	95	95	95	95	97	97	96	96
224	98	99	98	98	98	99	97	98	100	99	98	98	99	99	98	97	98	99	97	98	98	99	97	98	99	99	98	97	99	98	98	97
256	70	72	71	73	70	72	71	73	72	73	72	73	71	72	73	73	73	75	73	74	73	75	72	74	75	76	74	75	74	76	74	75
288	67	69	68	70	66	68	66	68	69	70	70	71	67	68	68	69	71	73	70	72	69	71	68	70	72	73	72	73	70	71	70	71
320	74	75	76	76	74	75	75	77	75	75	76	76	75	75	76	76	78	79	77	78	78	79	77	79	80	79	79	78	79	79	78	78
352	72	73	73	74	70	71	71	72	72	72	73	73	70	70	72	71	75	76	75	76	74	74	73	74	76	76	76	75	74	74	73	73
384	90	92	89	91	90	91	89	91	91	93	91	92	91	92	91	91	89	91	89	90	89	91	88	90	91	92	90	91	90	91	90	91
416	87	89	87	89	85	87	85	86	89	90	88	89	87	88	86	87	86	88	86	88	85	87	84	86	88	89	87	88	86	87	86	87
448	99	100	98	99	99	100	98	99	100	100	99	99	100	100	99	99	99	99	98	99	98	99	98	99	99	99	99	99	99	99	99	98
480	96	97	95	96	94	95	93	94	97	97	96	96	95	95	94	94	96	97	95	96	94	95	93	94	97	96	96	95	95	94	94	94
512	71	72	72	73	72	73	73	74	71	72	71	72	72	73	73	74	75	75	74	75	76	77	75	76	74	75	73	74	75	76	75	76
544	74	75	75	76	74	75	75	76	74	75	75	76	74	75	75	76	78	79	77	78	78	79	77	78	77	78	77	78	77	78	77	78
576	68	69	70	70	69	70	70	72	68	67	69	68	69	68	71	69	72	73	71	72	73	74	72	74	72	70	71	70	73	72	73	71
608	72	73	73	74	72	72	72	74	71	70	72	71	71	70	72	71	76	77	75	76	76	76	74	76	75	74	74	73	75	74	74	73
640	94	95	94	94	95	96	94	95	94	94	93	93	94	95	94	94	94	94	93	94	95	96	94	95	93	94	92	93	94	95	93	95
672	98	98	97	97	97	98	96	97	97	98	96	97	96	97	96	97	97	98	96	97	97	98	96	97	96	97	96	97	96	97	96	97
704	88	89	87	88	89	90	89	90	88	86	86	86	89	88	88	87	88	89	87	88	89	90	89	90	88	86	87	86	89	88	88	87
736	91	92	91	91	91	92	90	91	91	90	90	89	91	89	90	88	91	92	91	92	91	92	90	91	91	90	90	89	91	89	90	88
768	75	76	76	77	75	75	76	76	74	75	75	76	74	75	75	76	78	79	78	78	78	79	77	78	78	79	77	78	77	78	77	78
800	72	73	73	74	71	71	71	72	71	73	72	74	70	70	71	72	76	76	75	76	73	74	73	73	75	76	74	75	73	73	72	73
832	72	72	73	74	71	72	72	74	71	70	72	71	71	70	72	71	75	76	75	76	75	76	75	76	75	74	75	73	75	74	74	73
864	69	70	70	71	67	68	68	69	69	67	69	68	66	65	68	66	73	74	72	73	71	72	70	71	72	71	71	71	70	69	69	68
896	99	99	98	98	98	99	97	98	98	99	97	97	97	98	96	97	98	99	97	98	98	99	97	98	97	98	96	97	96	98	96	97
928	95	96	95	95	93	94	92	93	95	96	94	95	92	93	92	93	95	96	94	95	93	94	92	93	94	95	94	95	92	93	92	93
960	92	93	91	92	91	92	91	92	92	90	91	90	92	90	91	89	92	93	91	92	91	92	91	92	92	90	91	90	91	90	91	89
992	89	90	89	89	87	88	87	88	89	87	88	87	87	86	86	85	89	90	88	90	88	88	87	88	89	87	88	87	87	85	86	84
1024	87	88	86	87	87	90	87	89	88	89	87	88	89	90	88	89	86	88	85	87	87	89	86	89	87	88	86	87	88	90	88	89
1056	90	92	89	91	89	91	89	90	91	92	91	92	91	92	90	91	89	91	88	90	89	91	88	90	91	92	90	91	90	91	90	91
1088	95	96	95	95	96	97	96	97	96	96	95	95	97	97	97	96	95	96	94	95	96	97	96	96	95	96	95	95	97	97	97	96
1120	98	99	98	99	98	99	97	98	100	99	99	99	99	99	98	98	98	99	97	98	98	99	97	98	99	99	98	98	99	98	98	97
1152	71	73	71	73	73	74	72	74	73	74	72	74	74	75	74	74	69	71	69	70	70	72	69	71	71	72	70	71	72	73	71	72
1184	75	77	74	76	75	76	74	76	76	77	75	77	76	77	76	76	72	74	71	73	72	74	71	73	74	75	73	74	74	75	73	74
1216	76	77	76	76	78	79	76	78	77	77	76	76	78	78	78	77	74	75	73	74	75	76	75	75	75	75	74	73	76	76	75	74
1248	80	80	79	80	79	81	79	80	81	81	80	80	81	81	79	80	77	78	77	78	77	78	77	78	79	79	77	77	78	78	77	76
1280	90	92	90	92	90	92	89	91	92	93	91	92	91	92	91	91	89	92	89	91	89	92	89	91	91	92	90	92	91	92	90	91
1312	87	89	87	89	86	87	85	86	89	90	88	90	87	88	86	87	87	89	86	88	85	87	85	86	88	89	87	89	87	88	86	87
1344	99	100	98	100	99	100	98	99	100	100	99	99	100	100	99	99	99	99	98	99	98	99	98	99	100	100	99	99	99	99	99	98
1376	96	97	95	96	94	95	93	94	97	97	96	96	95	95	94	94	96	97	95	96	94	95	93	94	97	97	96	96	95	95	94	94
1408	75	77	74	77	75	77	74	76	77	78	76	77	76	77	76	77	73	75	72	74	73	74	72	74	75	76	74	75	74	75	74	74
1440	72	75	72	74	71	72	70	72	74	75	74	74	72	73	72	72	70	72	69	71	68	70	67	70	72	72	71	72	70	71	69	70
1472	80	81	80	80	80	81	79	80	81	81	79	80	80	80	80	79	78	79	77	78	77	79	77	78	79	79	77	77	78	78	77	77
1504	77	78	77	78	75	76	75	76	79	78	77	77	77	76	76	75	75	76	75	75	73	74	72	73	76	76	75	75	74	74	73	72
1536	94	95	94	94	95	96	94	95	94	94	93	94	95	96	94	95	94	95	93	94	95	96	94	95	93	94	92	93	94	95	94	95

Parallel Tools for the Masses

- **Using even the best tools can be tedious**
  - "Follow these 10 steps to perform the basic analysis of your program"



Vampir w/ CUDA



Compiler runtime bounds checking

Out of bounds reference in source code for program "flip"

```
…

allocate(put_seed(random_size)
)
…

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

Intel compiler:

```
ftn  -c -g -Ktrap=fp –check bounds  flip.f90
ftn  -c -g -Ktrap=fp –check bounds printit.f90
ftn  -o flip flip.o printit.o -g


% qsub –I –qdebug –lmppwidth=16
% cd $PBS_O_WORKDIR
%
% aprun –n 16./flip
forrtl: severe (408): fort: (2): Subscript #1
of the array SEED has value 3 which is greater
than the upper bound of 2
```

Compiler Documentation

- **For a list of compiler options, see the man pages for the individual compilers**
  - man pgcc | pgCC | pgf90
  - man icc | icpc| ifort
  - man gcc | g++ | gfortran
- **Use your favorite search engine to find vendor manuals on line**

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)**

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

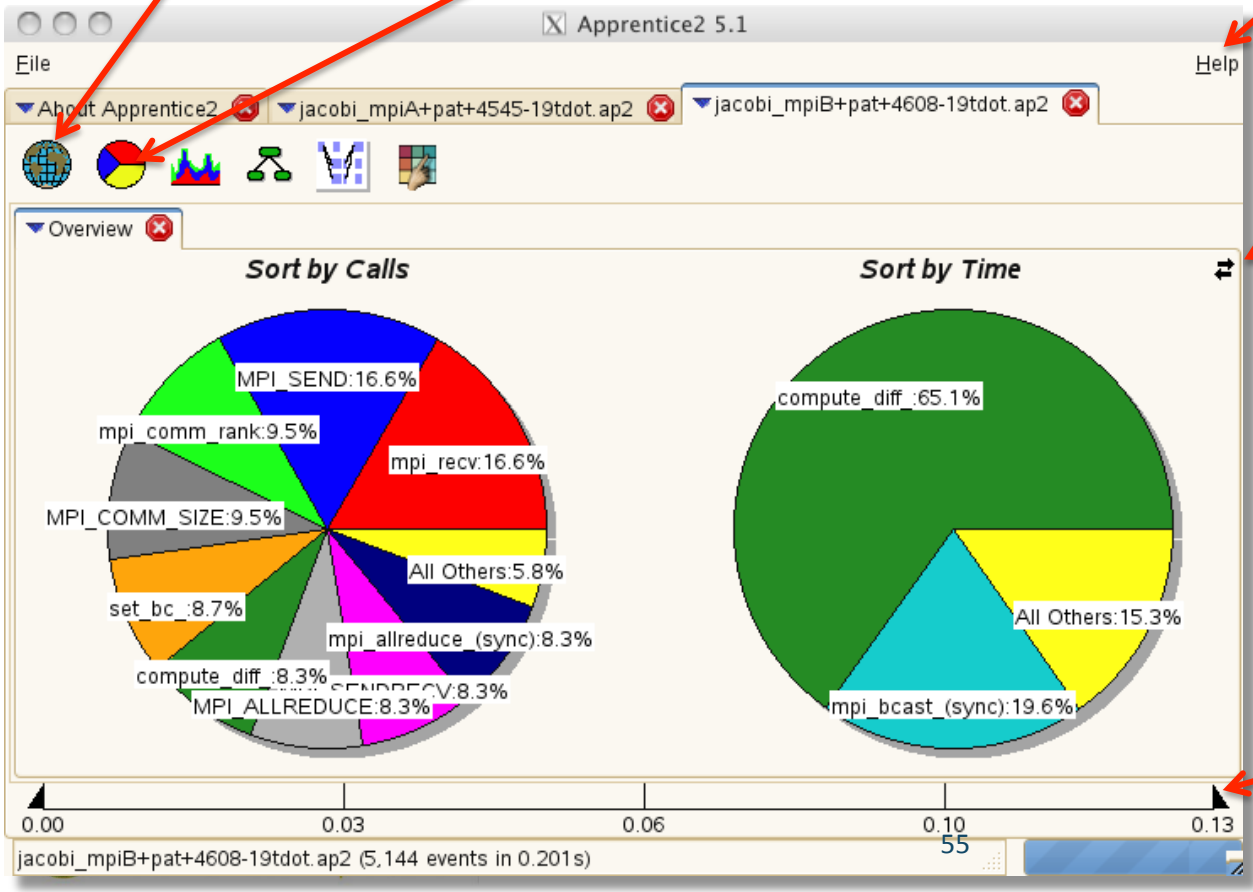
Apprentice Basic View

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

TAU Assignment

- **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**
  - http://acts.nersc.gov/events/Workshop2011/Talks/TAU.pdf

Users Want (Need?) 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

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?

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 & ubiquitous**
- **Simple, supplement existing complex tools**
  - Point the way for a “deeper dive” in problem areas

IPM

- Integrated Performance Monitoring
- Developed by David Skinner at NERSC
- MPI profiling, hardware counter metrics, IO profiling (?)
- IPM requires no code modification & no instrumented binary
- IPM uses hooks already in the MPI library to intercept your MPI calls and wrap them with timers and counters

IPM

- **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)

IPM Examples

Click on the metric you are want.

NERSC job details

http://www.nersc.gov/REST/jobs/job_details.php?stepid=732423.sdb&timestamp=1313679078&completion=1313679081

IPM Summary

Executable					./wrf.exe
Number of tasks	512	Aggregate GFlop/sec	0.1482	Average GFlop/sec/task	0.0003
Average wall secs	8.861e+01	Aggregate memory (GB)	32.2626	Average memory/task (GB)	0.0630
Average MPI secs/task	7.898e+01	MPI time %	89.14	Aggregate MPI calls made	7.027e+07

IPM Summary Statistics - 512 tasks

Metric	Sum over all tasks	Average (per task)	Task CV (%)	Task Minimum	Task Maxium
Aggregate Floating Point Operations (Flop x 10**9)	1.313e+01	2.565e-02	6.10	1.713e-02	2.758e-02
GFlop/sec	1.482e-01	2.895e-04	6.10	1.934e-04	3.114e-04
Maximum Memory Usage (GBytes)	3.226e+01	6.301e-02	10.12	5.701e-02	1.947e-01
Time Spent in MPI Routines (sec)	4.044e+04	7.898e+01	4.05	9.801e+00	8.359e+01
Wallclock Time (sec)	4.537e+04	8.861e+01	0.10	8.848e+01	8.895e+01

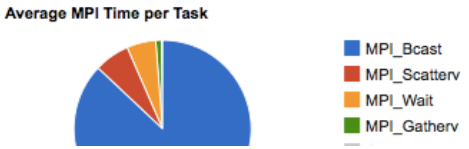
Memory in units of gigabytes; time in seconds.

Hardware counter statistics - 512 tasks

Counter Name	Sum over all tasks	Average (per task)	Task CV (%)	Task Minimum	Task Maxium
PAPI_FP_OPS	1.161799e+12	2.269139e+09	6.09	1.515023e+09	2.439529e+09

MPI Time Statistics - 512 tasks

Call	Sum over all tasks	Average (per task)	Task CV (%)	Task Minimum	Task Maxium	% of MPI	% of wall
MPI_Bcast	3.517e+04	6.869e+01	4.48	4.342e-01	7.269e+01	86.969	77.520
MPI_Scatterv	2.589e+03	5.057e+00	5.79	1.059e+00	5.540e+00	6.403	5.707
MPI_Wait	2.176e+03	4.249e+00	17.82	1.250e+00	4.968e+00	5.380	4.795
MPI_Gatherv	4.312e+02	8.422e-01	36.44	3.552e-03	2.271e+00	1.066	0.950
MPI_Isend	5.250e+01	1.025e-01	11.96	7.182e-02	1.259e-01	0.130	0.116
MPI_Irecv	1.033e+01	2.017e-02	10.21	1.217e-02	2.613e-02	0.026	0.023
MPI_Gather	1.021e+01	1.995e-02	502.07	1.391e-03	1.434e+00	0.025	0.023
MPI_Comm_rank	4.563e-01	8.913e-04	4.74	7.799e-04	1.404e-03	0.001	0.001
MPI_Comm_size	9.629e-02	1.881e-04	10.65	1.462e-04	4.859e-04	0.000	0.000
MPI_Init	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000	0.000	
MPI_Finalize	0.000e+00	0.000e+00	0.000e+00	0.000e+00	0.000	0.000	



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!)

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`

```
app2 [--limit_per_pe tags] XXX.ap2
```

PAPI

- **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
- **http://acts.nersc.gov/tau/index.html**

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