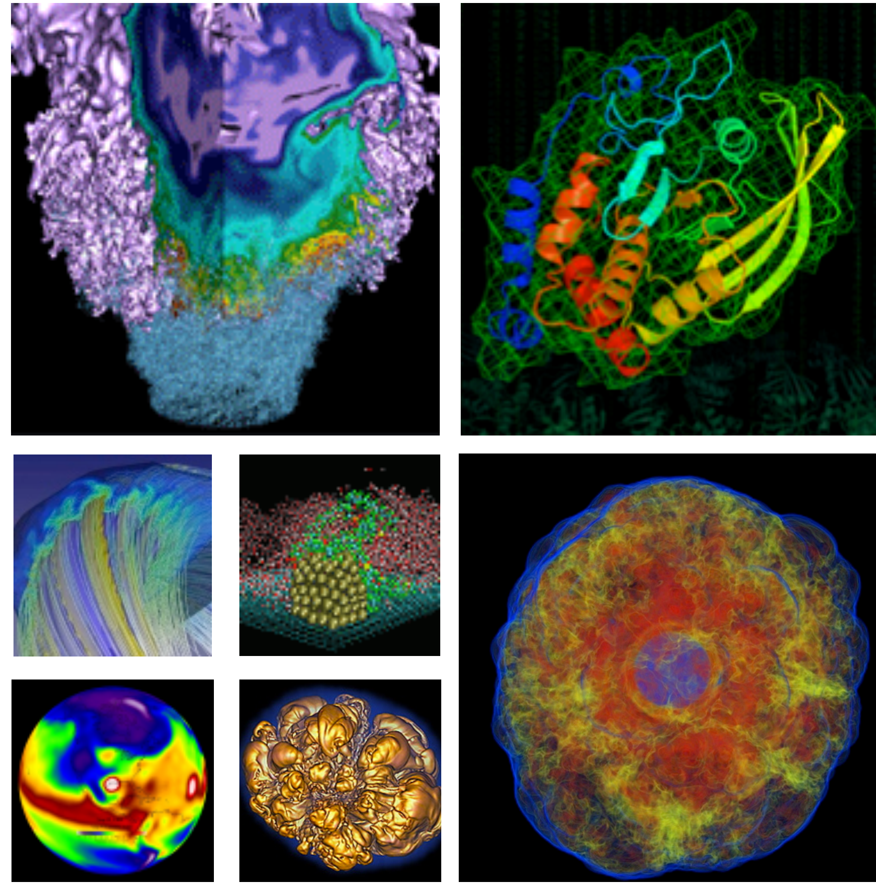
Debugging and Performance Analysis Tools at NERSC



2013 BOUT++ Workshop

Woo-Sun Yang

NERSC User Services Group

September 5, 2013

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Overview

- **Survey of selected debugging and profiling tools at NERSC**
  – To provide a quick start

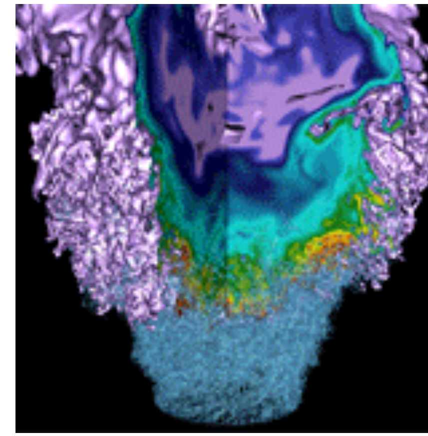
- **Some examples presented are using 'elm-pb'**
  – Build scripts (and batch scripts for some cases, too) available in a NERSC training directory

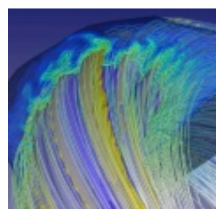
    % module load training

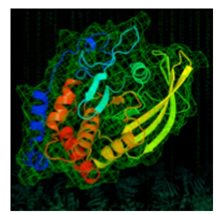
    % ls $EXAMPLES   (/project/projectdirs/training/2013/BOUT++/examples)

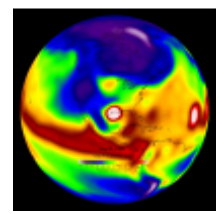
  – Note: Example results in the presentation were obtained with BOUT-1.0, but the scripts are updated for BOUT-2.0

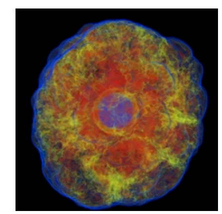
Debugging Tools

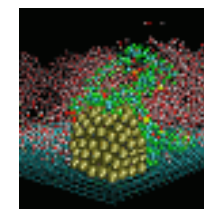












Debugging tools

- **Let us control the pace of running your code by**
  - Advancing a line or lines of your program at a time (‘next’, ‘step’, ‘continue’, …)
  - Stopping execution at certain locations in your program
    - Set a “breakpoint” where you want execution to stop
    - Set a “watchpoint” for a varialbe or an expression to make the program stop when its value changes
- **Let us examine execution flow or check variables to see if it is running as expected**

Debuggers on NERSC machines

- **Parallel debuggers with a graphical user interface**
  - DDT (Distributed Debugging Tool)
    - Can run for up to 8192 tasks
  - TotalView
    - Can run for up to 512 tasks
- **Cray Debugger Support Tools**
  - STAT (Stack Trace Analysis Tool)
  - ATP (Abnormal Termination Processing)
    - A system that monitors user applications and replaces the core dump with a more comprehensive stack backtrace and analysis
  - lgdb
    - A modified gdb for parallel programs that interfaces with aprun

For information

- **DDT**
  - $ALLINEA_TOOLS_DOCDIR/userguide.pdf (after loading 'allineatools' module)
  - http://www.nersc.gov/users/software/debugging-and-profiling/ddt/
- **TotalView**
  - http://www.roguewave.com/products/totalview
  - http://www.nersc.gov/users/software/debugging-and-profiling/totalview/

Running DDT

```
%./configure --with-debug
% make
% cd examples/elm-pb
% make
```

To compile with -g to have debugging symbols

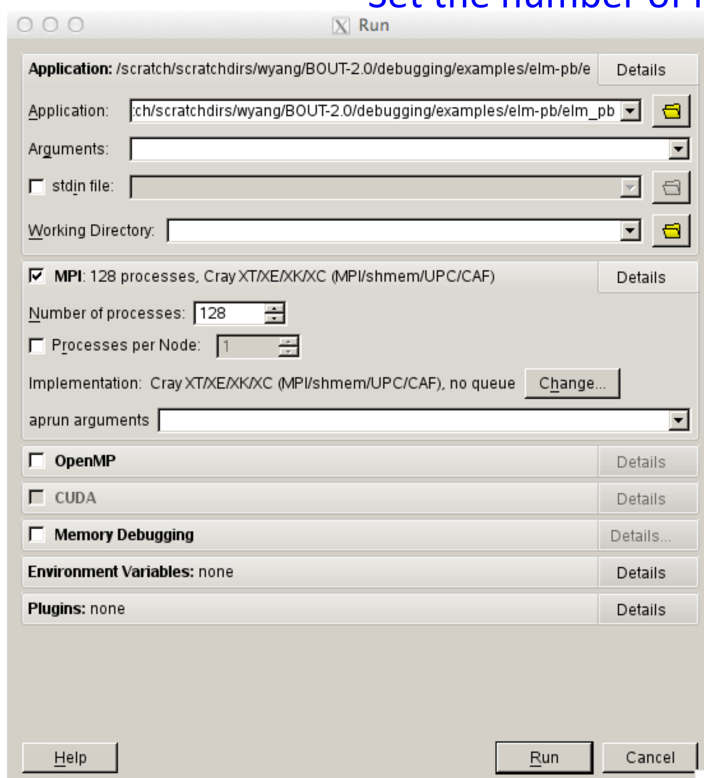
Let's try this example

```
% qsub -IV -lmppwidth=144,advres=bout.10 -q regular
% cd $PBS_O_WORKDIR
% module load allineatools
% ddt ./elm_pb
```

Start an interactive batch session

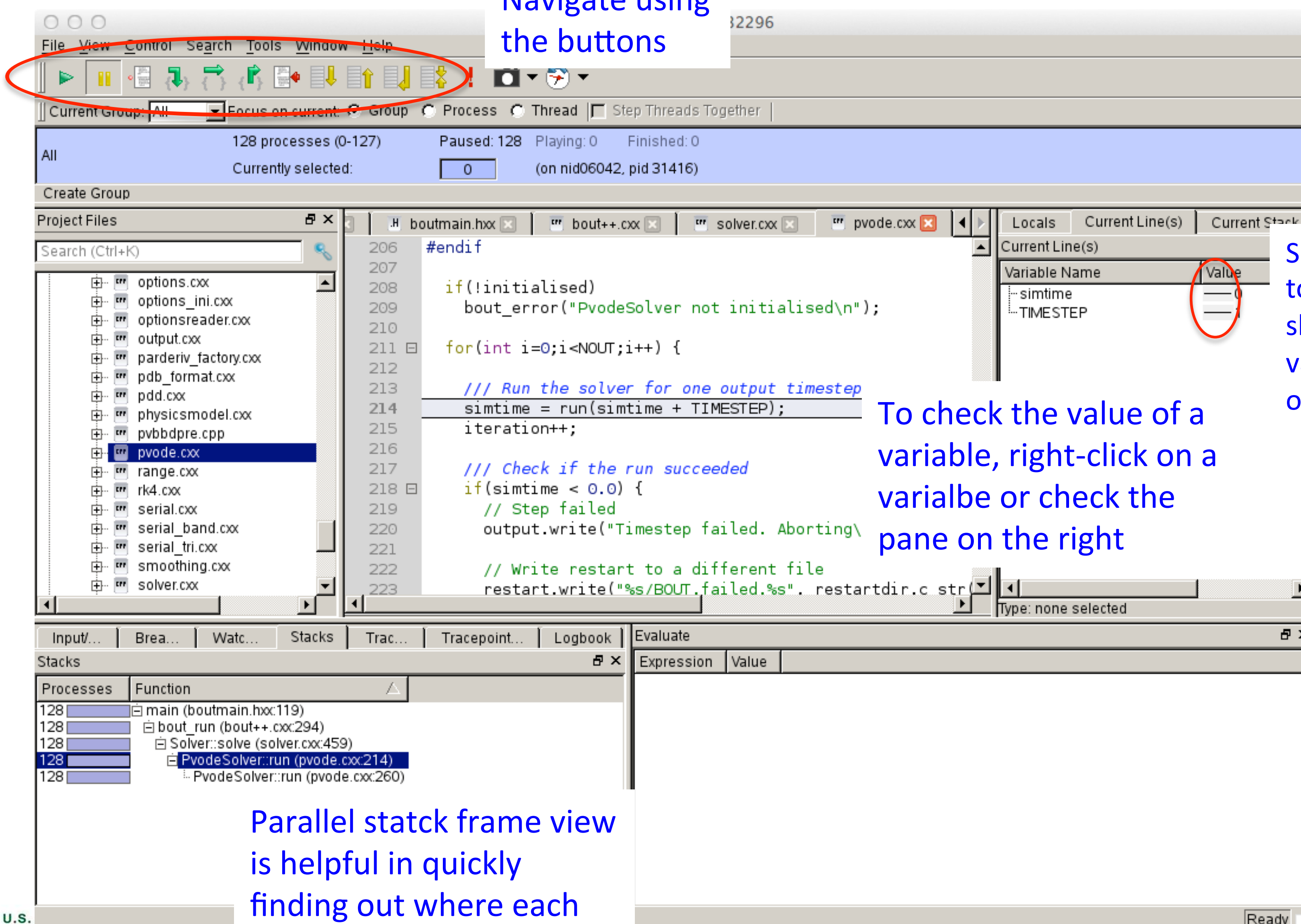
Use DDT

Set the number of MPI tasks to 128



Running DDT (cont'd)

Navigate using the buttons



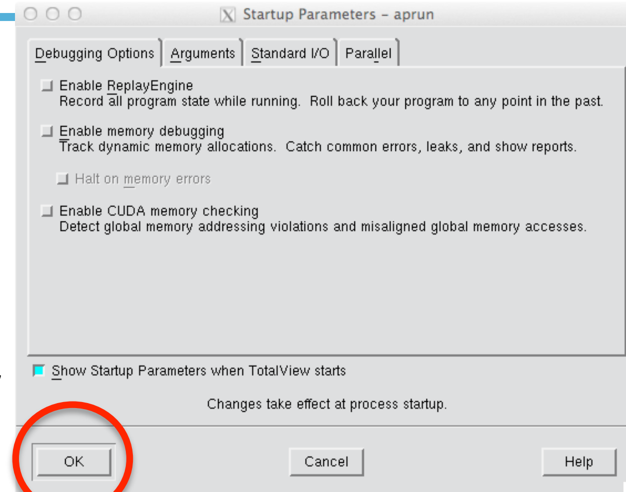
Sparklines to quickly show variation over tasks

To check the value of a variable, right-click on a varialbe or check the pane on the right

Parallel statck frame view is helpful in quickly finding out where each process is executing

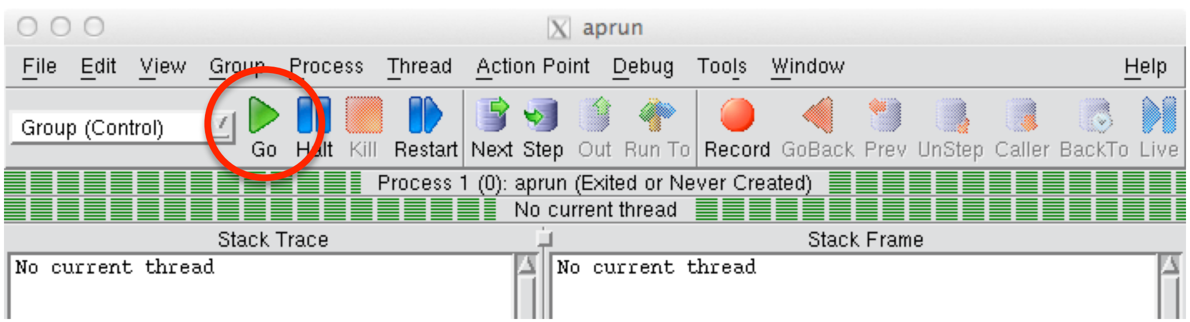
Running TotalView

```
% qsub -IV -lmppwidth=144
% cd $PBS_O_WORKDIR
% module load totalview
% totalview aprun -a -n 128 ./elm_pb
```

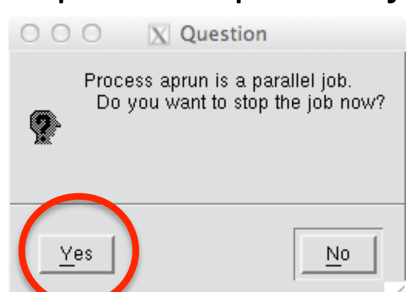


Then,

- Click OK in the 'Startup Parameters - aprun' window
- Click 'Go' button in the main window



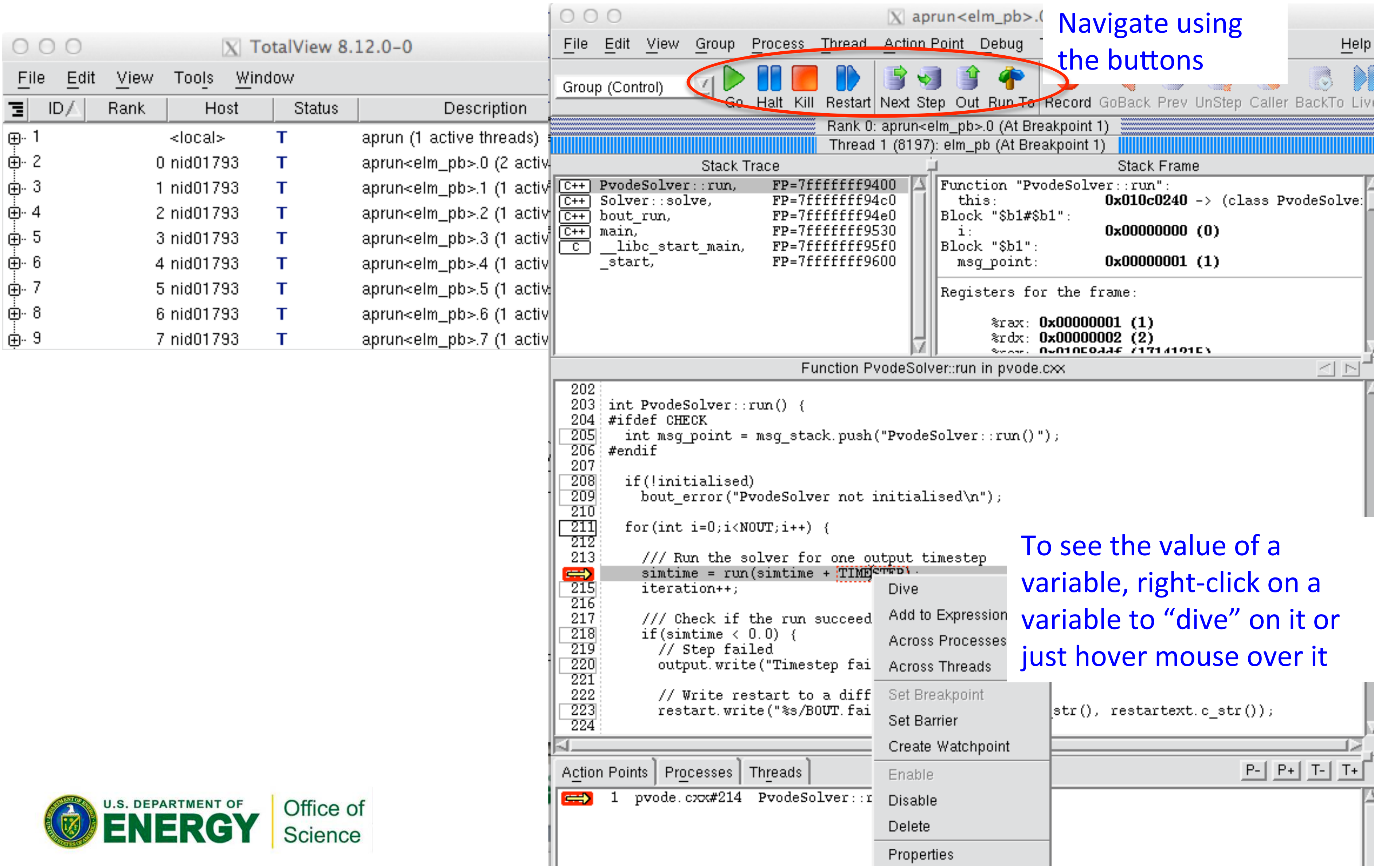
- Click 'Yes' to the question 'Process aprun is a parallel job. Do you want to stop the job now?'



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Running TotalView (cont'd)



Navigate using the buttons

To see the value of a variable, right-click on a variable to "dive" on it or just hover mouse over it

STAT (Stack Trace Analysis Tool)

- **Gathers stack backtraces (the function calling sequences) from all processes of a running application and merges them into a single file (*.dot)**
  - The output shows the location in the code that each process is executing
  - Can be used for debugging a hung application
- **STAT commands (after loading the 'stat' module)**
  - stat (STAT or stat-cl): invokes STAT to gather stack traces
  - statview (STATview or stat-view): a GUI to view STAT results
  - statgui (STATGUI or stat-gui): a GUI to run STAT or view results
- **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/

Gathering backtraces for a hung application using STAT

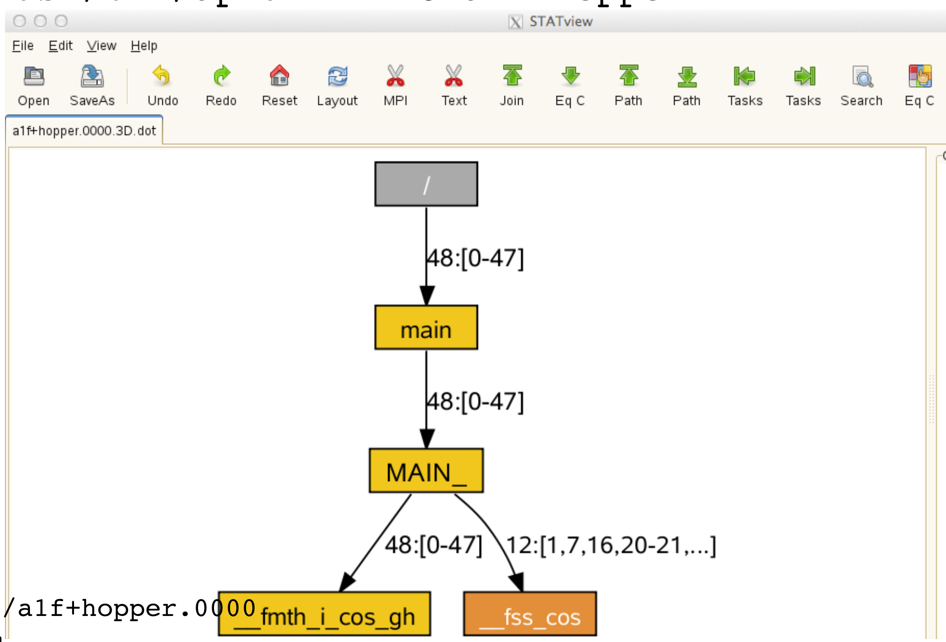
Find the MOM node that launched the app.

```
% qstat -f 6397413
...
 login_node_id = nid05621
```

Log into the MOM node

Find pid

```
% ssh -XY nid05621
% ps -aux | grep your_login_name
…
wyang 9246 0.0 0.0 27320 2100 ?  S 18:41 /usr/bin/aprun -n 48 a1f+hopper
…
% module load stat
% stat 9246
…
Attaching to application...
Attached!
Application already paused... ignoring request to pause
Sampling traces...
Traces sampled!
…
Resuming the application...
Resumed!
Merging traces...
Traces merged!
Detaching from application...
Detached!

Results written to /scratch/scratchdirs/wyang/stat_results/a1f+hopper.0000
% ls -l stat_results/a1f+hopper.0000/*.dot
-rw------- 1 wyang wyang 665 2013-08-27 18:45 stat_results/a1f+hopper.0000/a1f+hopper.0000.3D.dot
% statview stat_results/a1f+hopper.0000/a1f+hopper.0000.3D.dot
```



Attaching to an application using STAT

```
% qstat -f 6398933
...
 login_node_id = nid05620
```

Find the MOM node that launched the app.

```
% ssh -XY nid05620
…
```

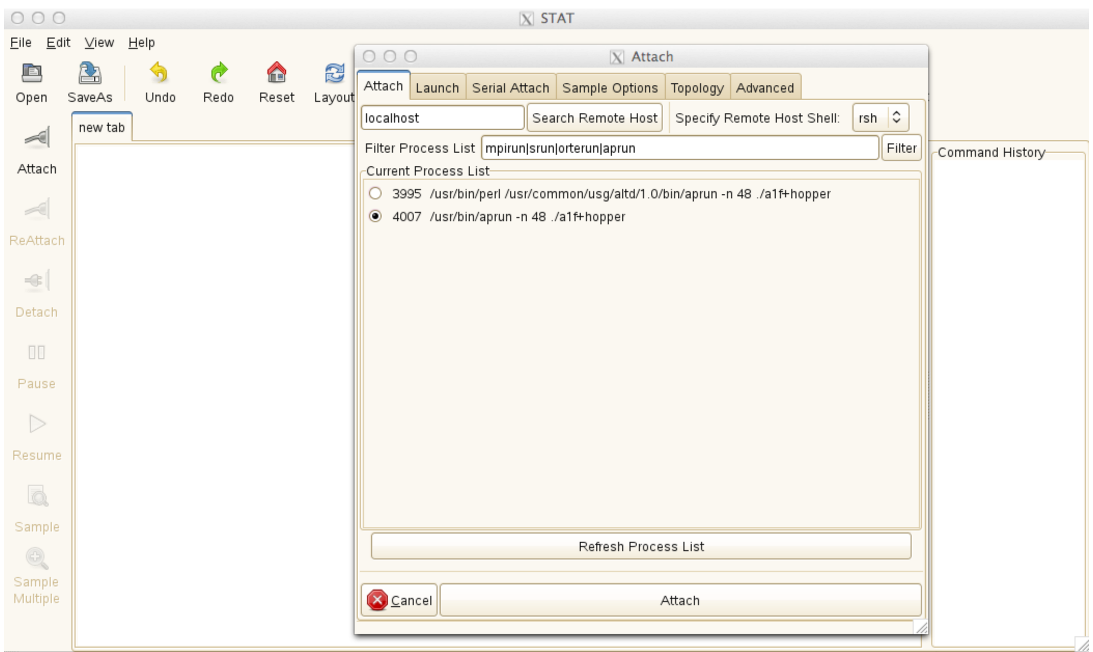
Log into the MOM node

```
% ps -aux | grep your_login_name
…
wyang 4007 0.0 0.0 26920 2100 ?  S 09:00 /usr/bin/aprun -n 48 a1f+hopper
…
```

Find pid

```
% module load stat
% statgui 4007
```

Attach to the app.



U.S. DEPARTMENT OF ENERGY

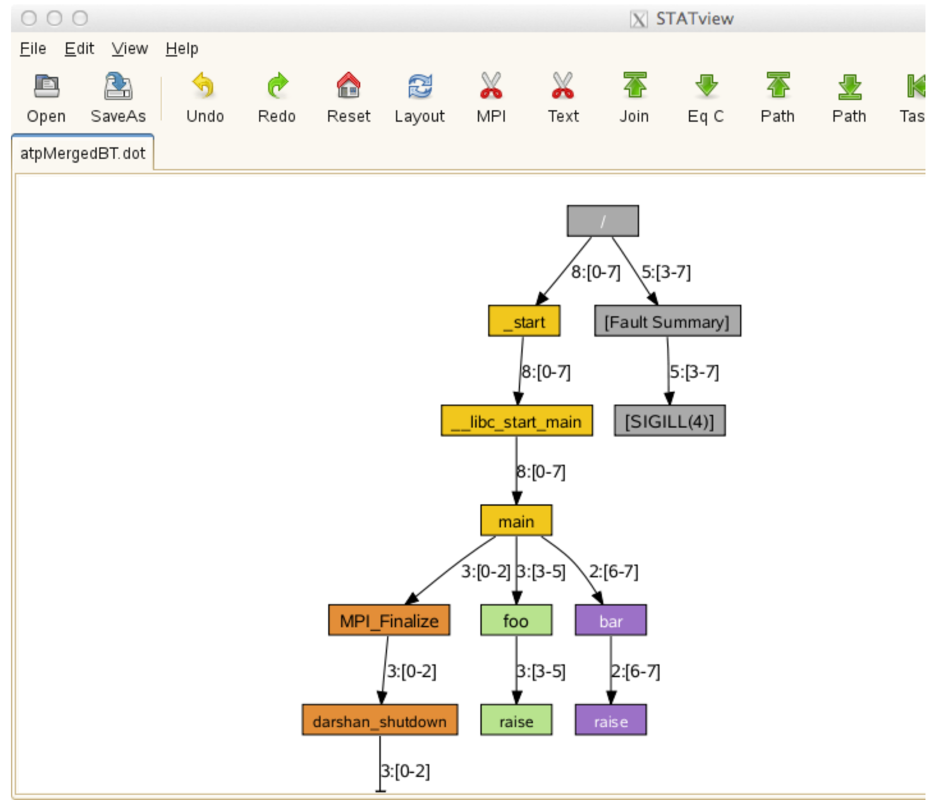
BERKELEY LAB

ATP (Abnormal Termination Processing)

- **When enabled, ATP gathers stack backtraces from all processes of a failing application**
  – Output saved in atpMergedBT.dot and atpMergedBT_line.dot (which shows source code line numbers)
  – They are viewed with statview
- **By default, the atp module is loaded on Hopper and Edison, but ATP is not enabled**
- **Can make core dumps (core.atp.*apid*.*rank*), too, by setting coredumpsize unlimited, but the location of failure can be inaccurate**
  – unlimit coredumpsize # for csh/tcsh
  – ulimit -c unlimited # for sh/bash/ksh
- **For more info**
  – 'intro_atp' man page
  – http://www.nersc.gov/users/software/debugging-and-profiling/gdb-and-atp/

Running an application with ATP

```
% cp $ATP_HOME/demos/testMPIApp.c .
% cc -o testMPIApp testMPIApp.c
% cat runit
#!/bin/csh
#PBS -l mppwidth=24
#PBS -l walltime=5:00
#PBS -q debug
#PBS -j oe

 cd $PBS_O_WORKDIR
 setenv ATP_ENABLED 1   # Enable ATP
#unlimit coredumpsize   # For core dumps
 aprun -n 8 ./testMPIApp 1 4
% qsub runit
6399762.hopque01

% cat runit.o6399762
…
Application 20111543 is crashing. ATP analysis proceeding...
…
Process died with signal 4: 'Illegal instruction'
View application merged backtrace tree with: statview atpMergedBT.dot
…
% module load stat
% statview atpMergedBT.dot   # or statview atpMergedBT_line.dot
```



Hung application with ATP

- **Force to generate a BT from a hung application**
- **For the following to work, the batch job should have ATP enabled in the batch script**

Find apid

```
% apstat
…
 Apid ResId     User    PEs Nodes     Age State            Command
…
20112743  3768    wyang    48      2    0h02m   run             a1f+hopper
…
% apkill 20112743
% cat runit.o6399999
…
Application 20112743 is crashing. ATP analysis proceeding...
…
Process died with signal 15: 'Terminated'
View application merged backtrace tree with: statview atpMergedBT.dot
…
% module load stat
% statview atpMergedBT.dot   # or statview atpMergedBT_line.dot
```

Kill the hung application

lgdb

- **Line (not GUI) mode parallel debugger by Cray**
  - Just like GDB but for MPI applications; threading not supported
  - Many GDB commands inherited
- **Use for**
  - Launching an application
  - Attaching to a running application
    - Useful for debugging a hung application
  - Comparative debugging
- **Some entities used by lgdb**
  - Process Set
    - Set of MPI processes used
    - Denote it by a scalar or, in case of MPI applications, array variable
    - Use this variable to refer to a group of the processes
  - Decomposition descriptor for distributed arrays
  - Assertion scripts: commands used for comparing variable values during comparative debugging

Launching an application with lgdb

```
% cc -g -o hello_mpi_c hello_mpi.c
% qsub -IV -lmppwidth=24 -q debug
…
% cd $PBS_O_WORKDIR
% module rm altd
% module load cray-lgdb
% lgdb
…
dbg all> launch $pset{8} hello_mpi_c

dbg all> break hello_mpi.c:21
dbg all> continue

dbg all> print $pset::myRank
pset[0]: 0
…
pset[7]: 7
dbg all> print $pset{3}::myRank
pset[3]: 3
```

Launching 'hello_mpi_c' with 8 MPI tasks; I am going to call this process set '$pset'

Setting a breakpoint at line 21 of hello_mpi.c

Check the value of 'myRank' for all the processes in $pset

Print the value of 'myRank' for process $pset[3] only

See the usage example in the man page which uses the example code in $CRAY_LGDB_DIR/demos/mpi_example

Attaching lgdb to an application

```
% qstat -f 6398933
...
 login_node_id = nid05620

% apstat
...
   Apid   ResId  User PEs Nodes   Age  State Command
...
200108035 1516  wyang  8     1 0h02m   run   a.out

% ssh nid05620
…
% module load cray-lgdb
% lgdb
…
dbg all> attach $pset 200108035
Attaching to alps applications, please wait…
…
Attach complete
dbg all> backtrace
…
dbg all> print $pset::myRank
pset[0]: 0
…
pset[7]: 7
```

Find the MOM node that launched the app.

Find apid

Log into the MOM node

Attach to the app.; I am going to call the process set '$pset'; $pset is an array variable whose size is determined automatically

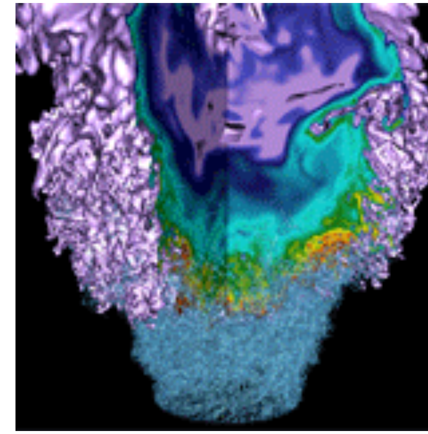
Check the backtraces

Check the value of 'myRank' for all the processes in $pset

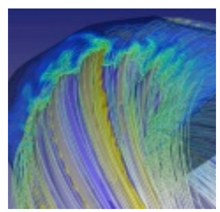
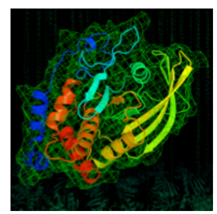
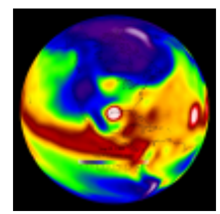
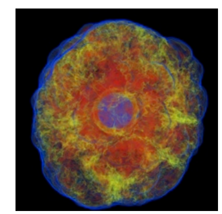
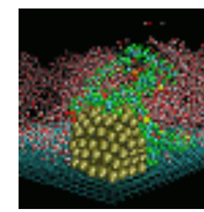
Comparative debugging with lgdb

- **To find a bug introduced in a version by running the new and old versions side by side and comparing the results**
- **Preliminary but can be still useful**
  - Comparative debugging will be formally introduced with the future release of CCDB (Cray Comparative Debugger)
- **A detailed walk-through in '*Using the lgdb Comparative Debugging Feature*' (http://docs.cray.com/books/S-0042-20/S-0042-20.pdf) using the example codes in $CRAY_LGDB_DIR/demos/hpcc_demo**

Performance Analysis Tools



NERSC

Performance analysis tools

- **Measure code performance in order to identify performance bottlenecks and improve them**
- **Two types of measurement**
  - Sampling
    - Sample where the program is executing (i.e., 'program counter') at regular time intervals (or certain events)
    - Low overhead
  - Tracing
    - Count some event such as the number of times certain library functions or user functions are executed
    - Need to specify a list of the functions to be traced
- **Some tools available at NERSC**
  - CrayPat: for sampling or tracing
  - IPM: for sampling
  - MAP: for sampling

CrayPat (Cray Performance Measurement and Analysis Tools)

NERSC

- **Steps**
  1. '**module load perftools**' before starting to build your code
  2. Instrument your program using 'pat_build'
     - Build your code; *.o must be kept as well as *.a, if any
     - **pat_build [options] a.out**   # to create an instrumented binary, a.out+pat
  3. Execute your instrumented program
     - **aprun … ./a.out+pat**
     - Performance data saved in a.out+pat+#####-####*e*.xf (*e*: s for sampling or t for tracing)
  4. Analyze the resulting data
     - **pat_report a.out+pat+#####-####*e*.xf**

- **Instrumentation types (and their pat_build options)**
  - For sampling
  - For tracing – Specify a list of the functions to be traced
    - User functions: using pat_build's -T,-t, -u (-u for all; can increase run time significantly)
    - Preset trace groups for popular functions: using pat_build's -g
      - mpi, heap, io, omp, blas, lapack, …

Sampling with CrayPat

```
% module rm darshan
% module load perftools
% ./configure
% make
% cd examples/elm-pb
% make
% pat_build -f elm_pb
% cat runit
…
 aprun -n 128 ./elm_pb+pat
% qsub runit
6416027.hopque01
% pat_report elm_pb+pat+20194488_4680s.xf > my.rpt
% more my.rpt
% app2 elm_pb+pat+20194488_4680s.ap2
% rm elm_pb+pat+20194488+4680s.xf
```

- `module rm darshan`: Unload darshan as it will interfere with perftools
- `pat_build -f elm_pb`: Build an instr. binary; -f to overwrite if there is one already
- `aprun -n 128 ./elm_pb+pat`: Use the instr. binary
- `pat_report ... > my.rpt`: ASCII text report captured in my.rpt
- `more my.rpt`: See the report
- `app2 ...`: Visualization of the results using a GUI tool, app2
- `rm ... .xf`: Not needed as you now have a .ap2 file; *.ap2 is self-contained and portable while .xf is not; text report can be generated from .ap2, too

Tracing with CrayPat (one way - using Automatic Program Analysis)

```
% module rm darshan
% module load perftools
% ./configure
% make
% cd examples/elm-pb
% make
% pat_build -f -O apa elm_pb
% cat runit
…
 aprun -n 128 ./elm_pb+pat
% qsub runit
6415765.hopque01
% pat_report elm_pb+pat+20192069_1225s.xf > mys.rpt
% more elm_pb+pat+20192069_1255s.apa

% pat_build -f -O elm_pb+pat+20192069_1225s.apa
% cat runit
…
 aprun -n 128 ./elm_pb+apa
% qsub runit
6415831.hopque01
% pat_report elm_pb+apa+20193028-1791t.xf > myt.rpt
% more myt.rpt
% app2 elm_pb+apa+20193028-1791t.ap2
% rm *.xf
```

Unload darshan as it will interfere with perftools

Special option for building an instr. binary for sampling

Sampling run

See what functions/groups are suggested for tracing, and edit if you want

Build a new instr. binary for tracing, guided by the sampling results

Use the new instr. binary for tracing

ASCII text report in myt.rpt

If you want…

Not needed as you now have .ap2 files

CrayPat-lite

- **A simplified version of CrayPat**
  - No need for you to manually build an instrumented binary
  - *.ap2, *.rpt (text report) files are generated for you

```
% module rm darshan
% module load perftools-lite
% ./configure
% make
% cd examples/elm-pb
% make
% cat runit
…
#setenv CRAY_LITE sample_profile
 setenv CRAY_ROOTFS DSL
 aprun -n 128 ./elm_pb
% qsub runit
6416899.hopque01
% more runit.o6416899
% more elm_pb+20199902-445s.rpt
% app2 elm_pb+20199902_445s.ap2
% rm elm_pb+20199902_445s.xf
```

Unload darshan as it will interfere with perftools

'sample_profile' for sampling; 'event_profile' for tracing

You need this line (because of pat_report)

Performance summary included in stdout file

Same text report saved in elm_pb+*.rpt

If you want…

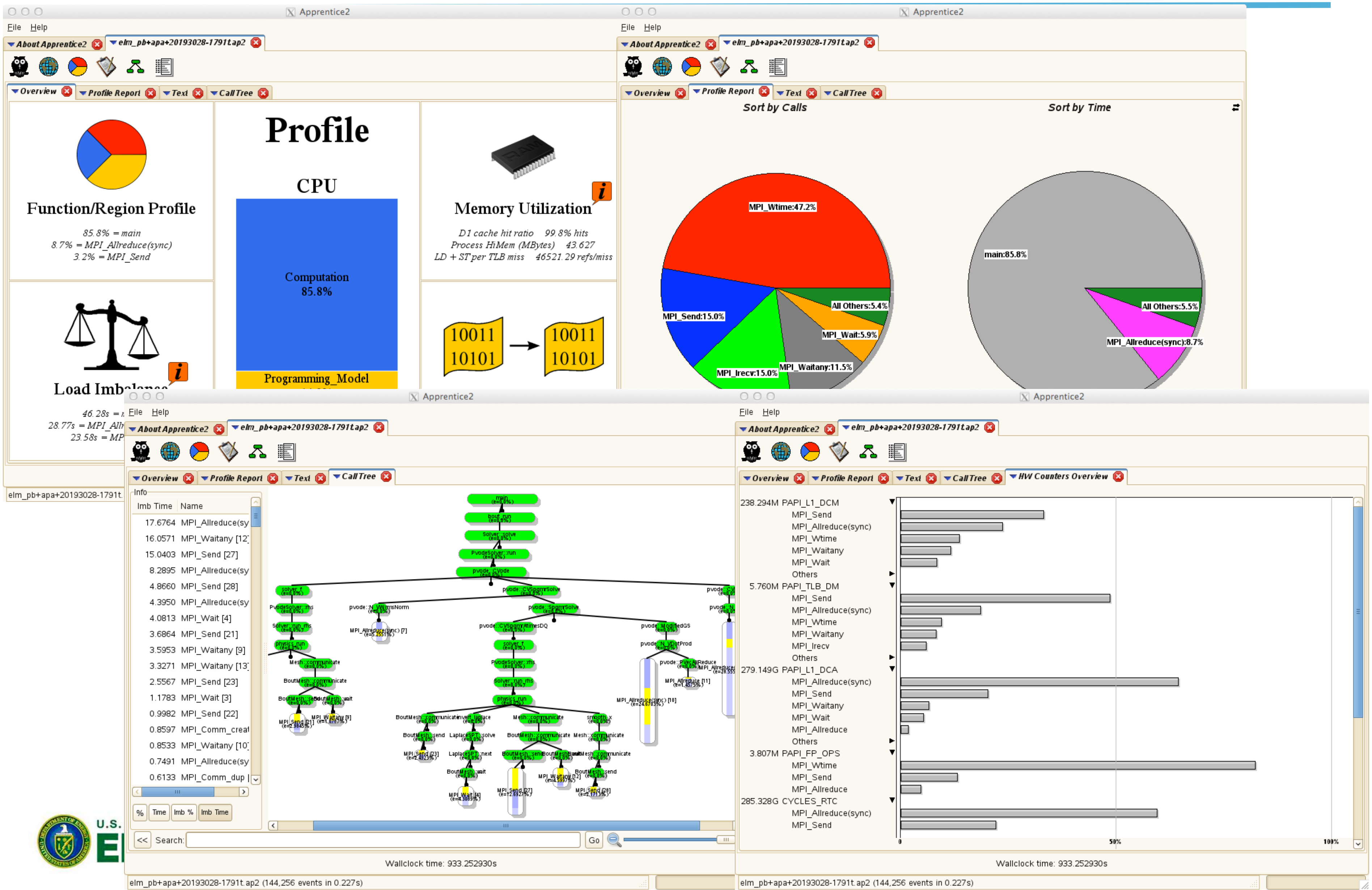
Not needed as you have a .ap2 file

U.S. DEPARTMENT OF ENERGY | Office of Science

CrayPat results in the text report

```
% more myt.rpt
...
Table 1:  Profile by Function Group and Function

 Time%   |        Time |      Imb. |  Imb.   |     Calls   |Group
         |             |      Time |  Time%  |             | Function
         |             |           |         |             |  PE=HIDE

 100.0% | 925.231187 |        -- |      -- | 3316802.0 |Total
|------------------------------------------------------------------------
|  85.8% | 793.976022 |        -- |      -- |        2.0 |USER
||-----------------------------------------------------------------------
|  85.8% | 793.976006 | 46.283832 |    5.6% |        1.0 | main
||=======================================================================
|   8.7% |  80.708496 | 28.767508 |   35.6% |    88812.0 |MPI_SYNC
||-----------------------------------------------------------------------
|   8.7% |  80.708496 | 28.767508 |   35.6% |    88812.0 | MPI_Allreduce(sync)
||=======================================================================
|   5.5% |  50.546518 |        -- |      -- |  3227787.0 |MPI
||-----------------------------------------------------------------------
||   3.2% |  29.195955 | 23.581497 |   45.0% |   498535.0 |MPI_Send
||   1.0% |   8.821983 | 21.931692 |   71.9% |   381225.0 |MPI_Waitany
|========================================================================
...
```

CrayPat results displayed with app2



More things to do with CrayPat…

- **Automatic Rank Order Analysis**
  - Suggests a better MPI rank placement
- **CrayPat API**
  - Instrument and get tracing results only for selected regions of your code
- **Monitor a selected group of hardware counters (floating point operations, cache usage, etc.) or network performance counters**
- **Reveal**
  - A new tool that combines run-time performance stats and program source code visualization with compile-time optimization feedback for optimization
- **For info:**
  - Man pages: 'intro_craypat', 'craypat-lite', 'pat_build', 'hwpc', 'nwpc', 'pat_report', 'pat_help', 'grid_order', 'reveal'
  - Pat_help online help systems

    % pat_help
  - 'Using Cray Performance Measurement and Analysis Tools' (S-2376-610, http://docs.cray.com/books/S-2376-610/S-2376-610.pdf)
  - http://www.nersc.gov/users/software/debugging-and-profiling/craypat/

IPM (Integrated Performance Monitoring)

- **Profiling tool with a low overhead that reports**
  - Floating point operations
  - Memory usage
  - MPI function timings
  - Hardware counters data
  - Load imbalance
  - …
- **For info:**
  - http://ipm-hpc.sourceforge.net/
  - http://www.nersc.gov/users/software/debugging-and-profiling/ipm/

Profiling using IPM

```
% module rm darshan
% module load ipm
% ./configure EXTRA_LIBS="${IPM_GNU}"
% make
% cd examples/elm-pb
% make
% cat runit
…
#setenv IPM_REPORT terse
 setenv IPM_REPORT full
 setenv IPM_HPM PAPI_FP_OPS,PAPI_TOT_INS,PAPI_L1_DCM,PAPI_L1_DCA
 aprun -n 128 ./elm_pb
% qsub runit
6418950.hopque01
% more runit.o6418950
% ipm_parse -html wyang.1378041829.ipm.xml
…
% tar -cvf ipmrpt.tar elm_pb_128_wyang.1378041829.ipm.xml_ipm_6418950.hopque01
```

Unload darshan as it will interfere

Add the IPM link flags to EXTRA_LIBS

Choose either type; 'terse' is default

Set hardware counters if you want; only PAPI_FP_OPS is set by def.

See the text report at the end of the stdout file

Create html files out of xml

Do the following on your local desktop where a web browser exists; get the file, untar it and open index.html in the created directory using a web browser

```
localmachine % scp myloginid@hopper.nersc.gov:/my/directory/ipmrpt.tar .
localmachine % tar -xvf ipmrpt.tar
```

IPM results (1)

```
% more runit.o6418950
...
##IPM2v0.xx#####################################################
#
# command   : ./elm_pb
# start     : Sun Sep 01 13:23:49 2013   host      : nid00779
# stop      : Sun Sep 01 13:38:47 2013   wallclock : 897.22
# mpi_tasks : 128 on 6 nodes             %comm     : 11.93
# mem [GB]  : 3.38                       gflop/sec : 29.24
#
#           :       [total]        <avg>         min          max
# wallclock :     114836.90       897.16      897.11       897.22
# MPI       :      13698.07       107.02       77.43       145.18
# %wall     :
#   MPI     :                      11.93        8.63        16.18
# #calls    :
#   MPI     :     212841728      1662826     1183837      1731253
# mem [GB]  :          3.38         0.03        0.02         0.03
#
#                               [time]        [count]        <%wall>
# MPI_Allreduce                9126.85       11367936           7.95
# MPI_Send                     3093.66       63812480           2.69
# MPI_Waitany                   804.80       48796800           0.70
# MPI_Wait                      644.26       25025280           0.56
```

IPM results (2)

6418950.hopque01

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

command: ./elm_pb			
codename:	unknown	state:	unknown
username:	wyang	group:	
host:	nid00779 (x86_64_Linux)	mpi_tasks:	128 on 6 hosts
start:	09/01/13/06:23:49	wallclock:	8.97548e+02 sec
stop:	09/01/13/06:38:47	%comm:	11.9231699516349
total memory:	3.44727039999999 gbytes	total gflop/sec:	0.0347140253223225
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop
PAPI_FP_OPS	26230877880085	*
PAPI_L1_DCA	127868505433094	*
PAPI_L1_DCM	186381889684	*
PAPI_TOT_INS	242606121165071	*

Communication

% of MPI Time

HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_FP_OPS	*	204928733438.16	183408469176 (67)	218388257083 (112)
PAPI_L1_DCA	*	998972698696.05	930195500088 (117)	1069000558265 (75)
PAPI_L1_DCM	*	1456108513.16	1147365493 (121)	2912526906 (19)
PAPI_TOT_INS	*	1895360321602.12	1770354532482 (117)	2016061278284 (75)

Communication Event Statistics (0.00% detail, 1.3698e+04 error)

Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall

Load balance by task: HPM counters

MAP

- **New parallel profiling tool with GUI by Allinea Software**
- **Can run MAP for up to 512 tasks**
  - Shared by other users
- **Need to build two small libraries for sampling, MAP sampler and MPI wrapper libraries**
  - make-map-static-libraries: for static linking
  - make-map-cray-libraries: for dynamic linking
- **Need to follow a certain linking order - see the user manual**
- **For info:**
  - $ALLINEA_TOOLS_DOCDIR/userguide.pdf (after loading the allineatools module)
  - https://www.nersc.gov/users/software/debugging-and-profiling/MAP/

Using MAP

```
% module load allineatools
% ./configure CXXFLAGS="-g" LDFLAGS="-v"
```

Add '-g' to CXXFLAGS to get debugging symbols; add '-v' (verbose) to LDFLAGS to get the detailed link line printed to terminal

```
% sed -i 's/@$(LD)/$(LD)/' make.config
```

Change make.config to echo the link command

```
% make
% make-map-static-cray-libraries lib
```

Build the libs in 'lib' directory that MAP needs

```
% cd examples/elm-pb
% set cmd=`make |& grep "collect2" | \
  sed -e 's/collect2/collect2 --eh-frame-hdr/' \
      -e 's/\(-lmpichcxx_gnu_46\)/-L..\/..\/lib -lmap-sampler-pmpi \
 -undefined=allinea_init_sampler_now -lmap-sampler \1/'`
```

Build and capture the link line; modify it to include the libs that MAP needs

```
% eval "$cmd"
```

Run the modified link command

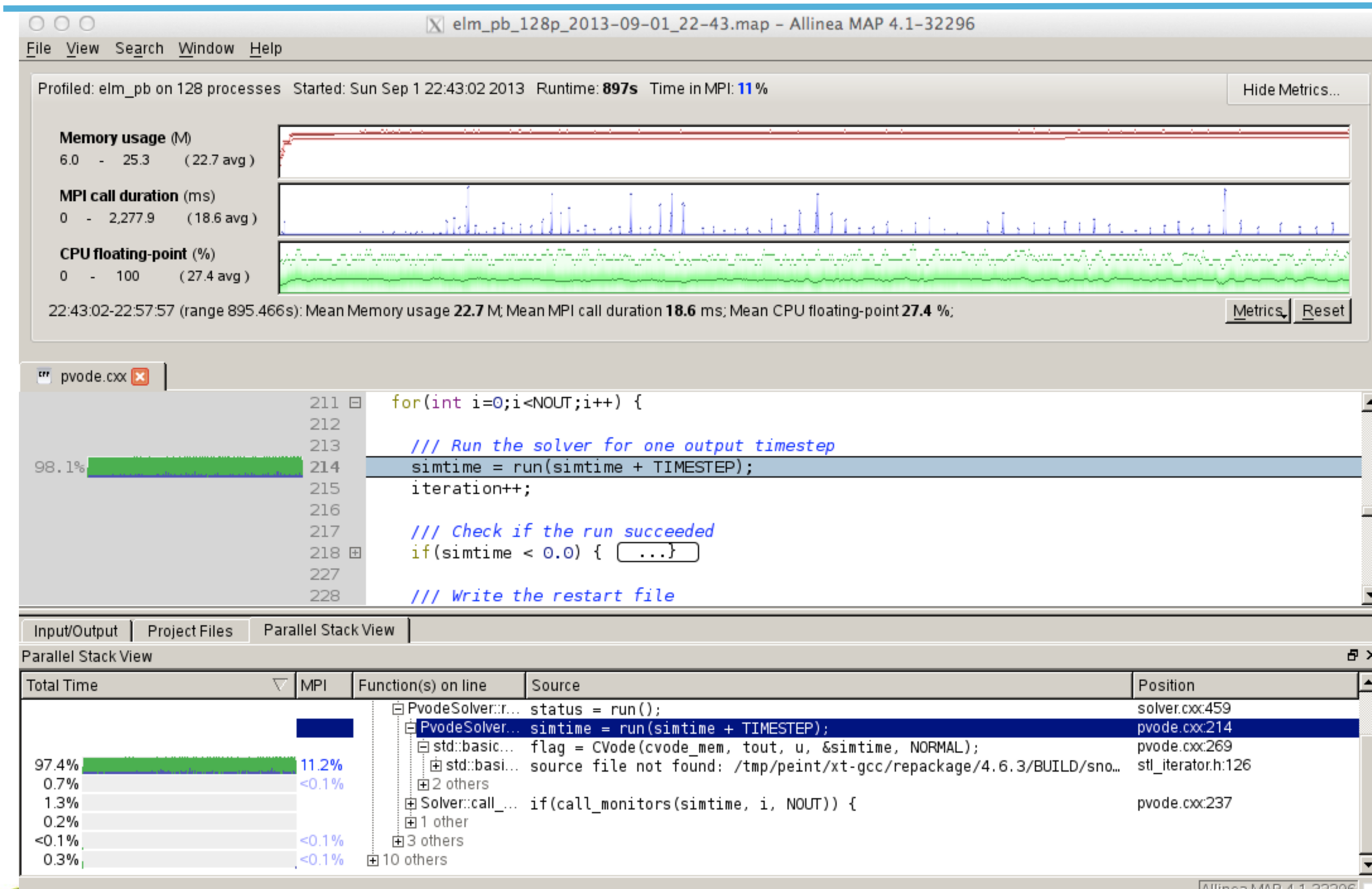
```
% ls -lrt
…
-rwx------  1 wyang wyang 28842273 2013-09-01 22:30 elm_pb

% qsub -IV -lmppwidth=144 …
% cd $PBS_O_WORKDIR
% map ./elm_pb
```

Run with MAP; select 128 tasks in the Run window

```
% ls -lrt
…
-rw-------  1 wyang wyang  7889009 2013-09-01 22:59 elm_pb_128p_2013-09-01_22-43.map
```

Profiling results saved in a file

MAP results


elm_pb_128p_2013-09-01_22-43.map - Allinea MAP 4.1-32296
File View Search Window Help
Profiled: elm_pb on 128 processes Started: Sun Sep 1 22:43:02 2013 Runtime: 897s Time in MPI: 11 %
Hide Metrics...
Memory usage (M)
6.0 - 25.3 (22.7 avg)
MPI call duration (ms)
0 - 2,277.9 (18.6 avg)
CPU floating-point (%)
0 - 100 (27.4 avg)
22:43:02-22:57:57 (range 895.466s): Mean Memory usage 22.7 M; Mean MPI call duration 18.6 ms; Mean CPU floating-point 27.4 %;
Metrics Reset
pvode.cxx
211 for(int i=0;i<NOUT;i++) {
212
213 /// Run the solver for one output timestep
98.1% 214 simtime = run(simtime + TIMESTEP);
215 iteration++;
216
217 /// Check if the run succeeded
218 if(simtime < 0.0) { ...}
227
228 /// Write the restart file
Input/Output Project Files Parallel Stack View
Parallel Stack View
Total Time MPI Function(s) on line Source Position
PvodeSolver::r... status = run(); solver.cxx:459
PvodeSolver... simtime = run(simtime + TIMESTEP); pvode.cxx:214
97.4% 11.2% std::basic... flag = CVode(cvode_mem, tout, u, &simtime, NORMAL); pvode.cxx:269
0.7% <0.1% std::basi... source file not found: /tmp/peint/xt-gcc/repackage/4.6.3/BUILD/sno... stl_iterator.h:126
1.3% 2 others
0.2% Solver::call_... if(call_monitors(simtime, i, NOUT)) { pvode.cxx:237
<0.1% <0.1% 1 other
0.3% <0.1% 3 others
10 others
Allinea MAP 4.1-32296

National Energy Research Scientific Computing Center