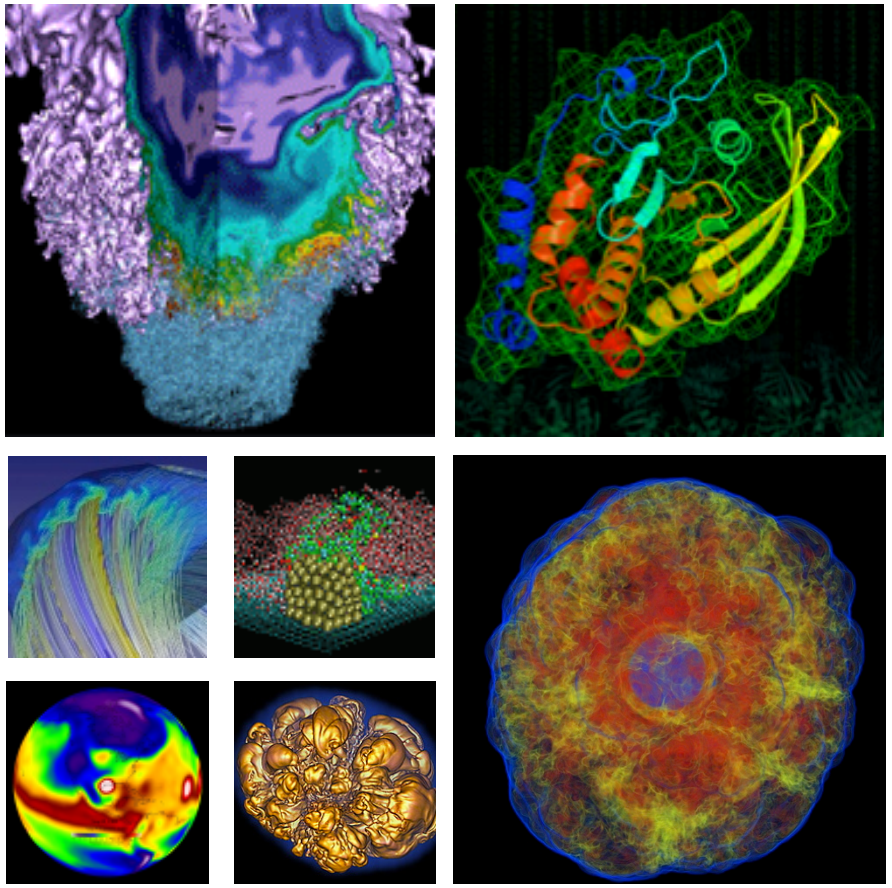
Other Debugging Tools on Cori and Edison

Debugging and Profiling with Allinea (ARM) Tools and Others



Woo-Sun Yang

User Engagement Group, NERSC

April 24, 2018

Parallel debuggers on Cori and Edison

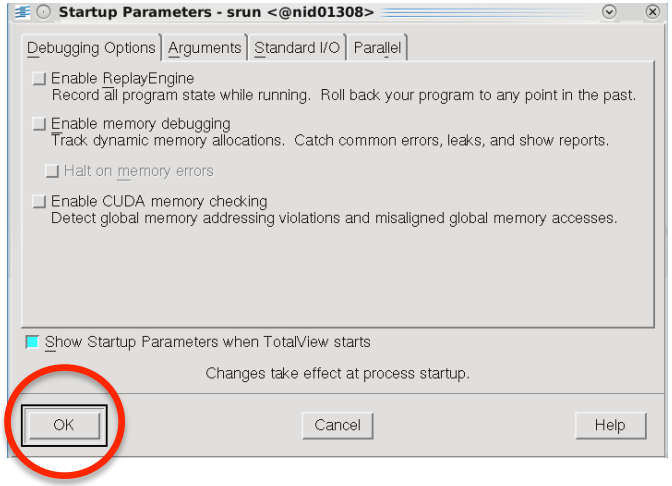
- **Parallel debuggers with a graphical user interface**
  - DDT (Distributed Debugging Tool)
  - TotalView
- **Specialized debuggers on Cori and Edison**
  - STAT (Stack Trace Analysis Tool)
    - Collect stack backtraces from all (MPI) tasks
  - Cray ATP (Abnormal Termination Processing)
    - Collect stack backtraces from all (MPI) tasks when an application fails
- **Valgrind**
  - Suite of debugging and profiling tools
  - Best known for its detailed memory debugging (memcheck)
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/valgrind/
- **Intel Inspector**
  - Thread and memory debugging
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/inspector/
- **Cray debuggers for comparative debugging**
  - CCDB
  - lgdb

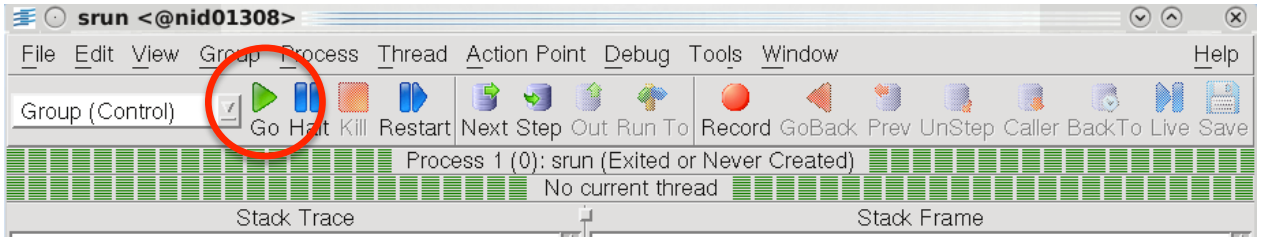
TotalView

```
$ salloc -N 1 -t 30:00 -q debug -C knl
$ module load totalview
$ export OMP_NUM_THREADS=6
$ totalview srun -a -n 4 ./jacobi_mpiomp
```

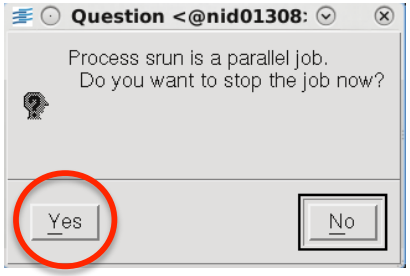
Then,

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



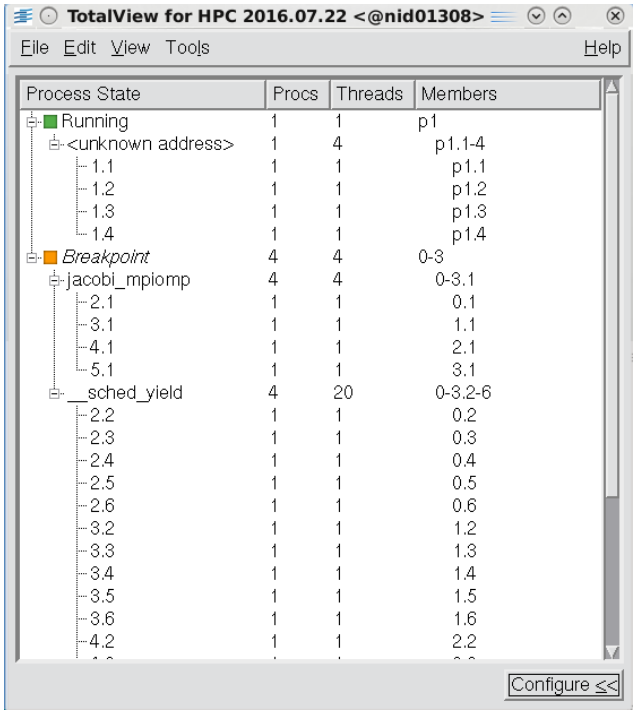


- Click 'Yes' to the question 'Process srun is a parallel job. Do you want to stop the job now?'



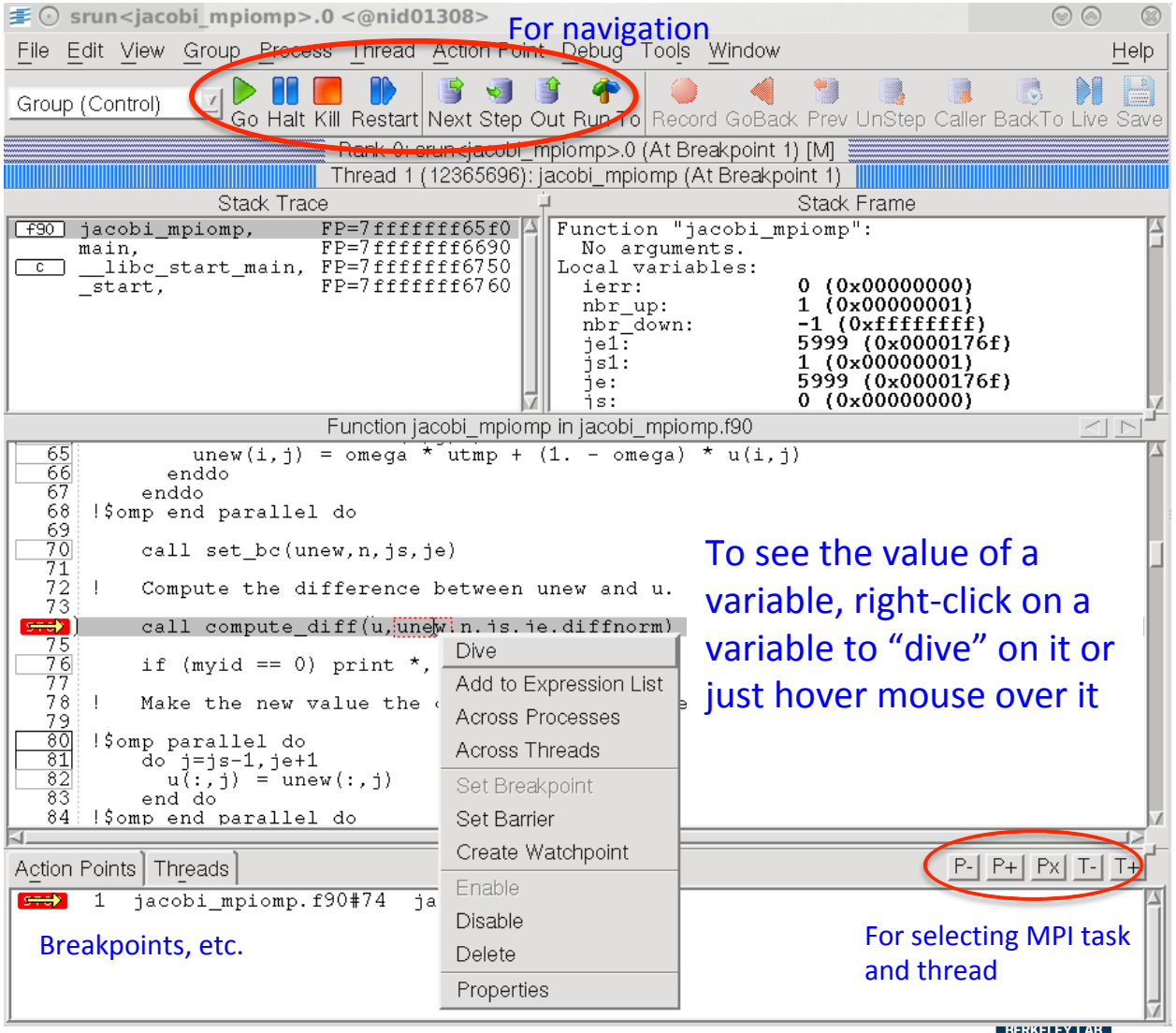
TotalView (cont'd)

Root window



State of MPI tasks and threads; members denoted roughly as '*rank.thread*'

Process window



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

Breakpoints, etc.

For selecting MPI task and thread

Memory debugging with MemoryScape

- **MemoryScape integrated into TotalView for memory debugging**
  - Memory leaks
  - Memory usage
  - Memory corruption
  - …

- **A statically-linked executable**

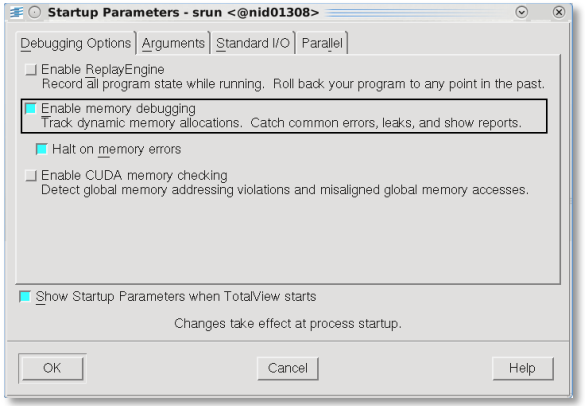
```
$ module load totalview
$ CC -g -O0 -o memry_leaks memory_leaks.o ${TVMEMDEBUG_POST_OPTS}
```

- **A dynamically-linked executable, build as usual**

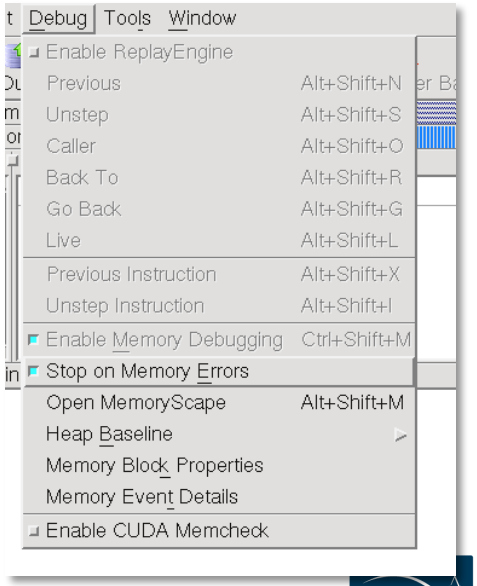
```
$ CC -dynamic -g -O0 -o memry_leaks memory_leaks.o
```

Memory debugging with MemoryScape

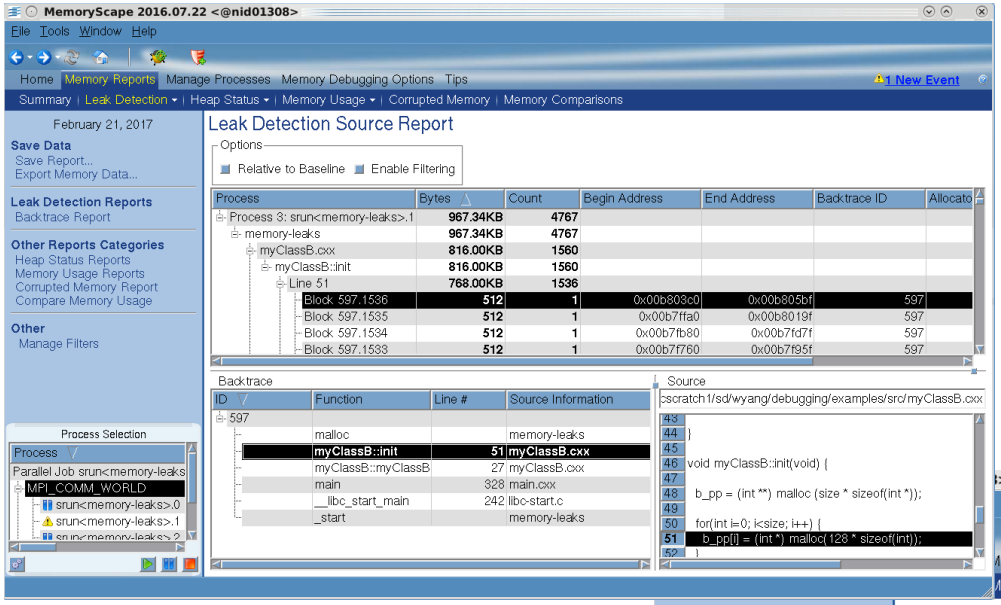
- **Start TotalView and enable memory debugging in the 'Startup Parameters' window**
- **Proceed to use TotalView as usual**

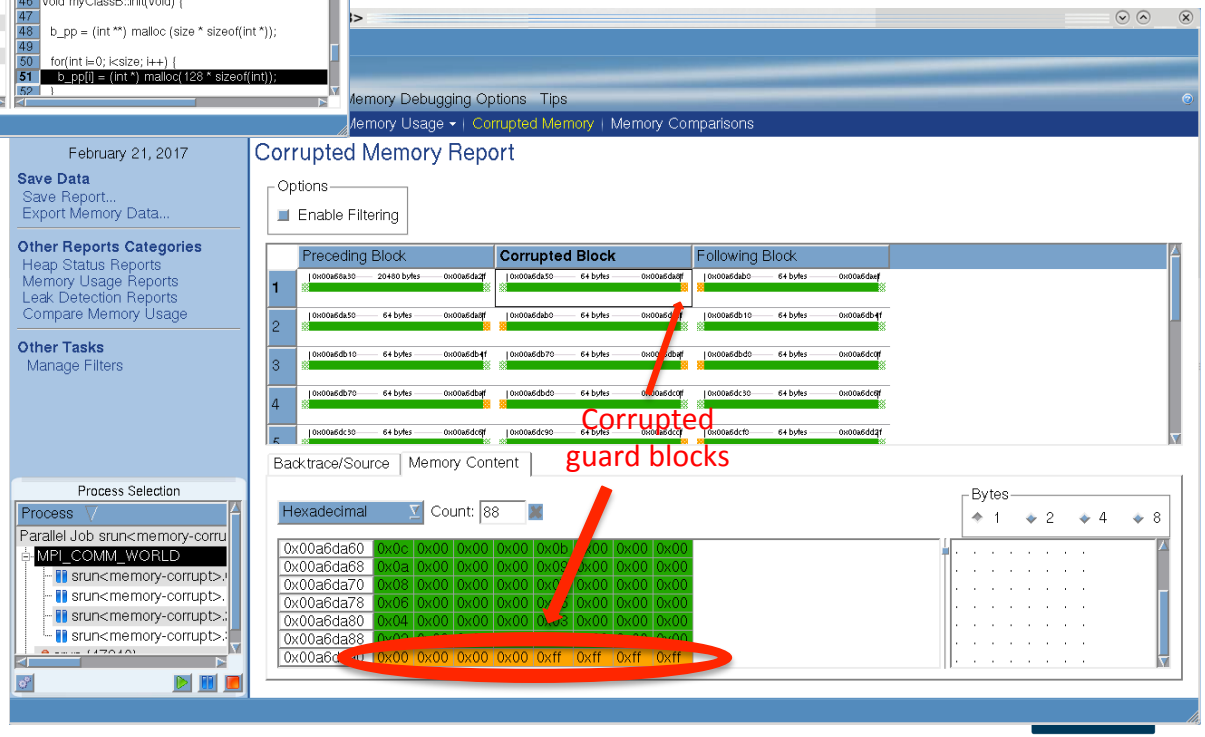


- **For memory-related issues, open MemoryScape from the Debug pull-down menu**



Memory debugging examples





U.S. DEPARTMENT OF ENERGY | Office of Science

STAT (Stack Trace Analysis Tool)

- **Gathers stack backtraces (sequence of function calls leading up to the current function) for all (MPI) processes**
  - Merge them into a single file (*.dot)
  - Results displayed as a single call tree for all processes
  - 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, OpenMP**

STAT (Cont'd)

- **STAT commands (after loading the 'stat' module)**
  - stat-cl: invokes STAT to gather stack backtraces
  - STATview: a GUI to view the results
  - STATGUI: a GUI to run STAT or view results
- **For more info:**
  - 'intro_stat', 'STAT', 'STATview' and 'STATGUI' man pages
  - /opt/cray/pe/stat/default/doc/stat_userguide.pdf
  - http://www.nersc.gov/users/software/performance-and-debugging-tools/stat-and-atp/

Debug a hung application with STAT

- If your code hangs in a consistent manner, you can use STAT to see where some MPI ranks are stuck.
- One way to use STAT is as follows.

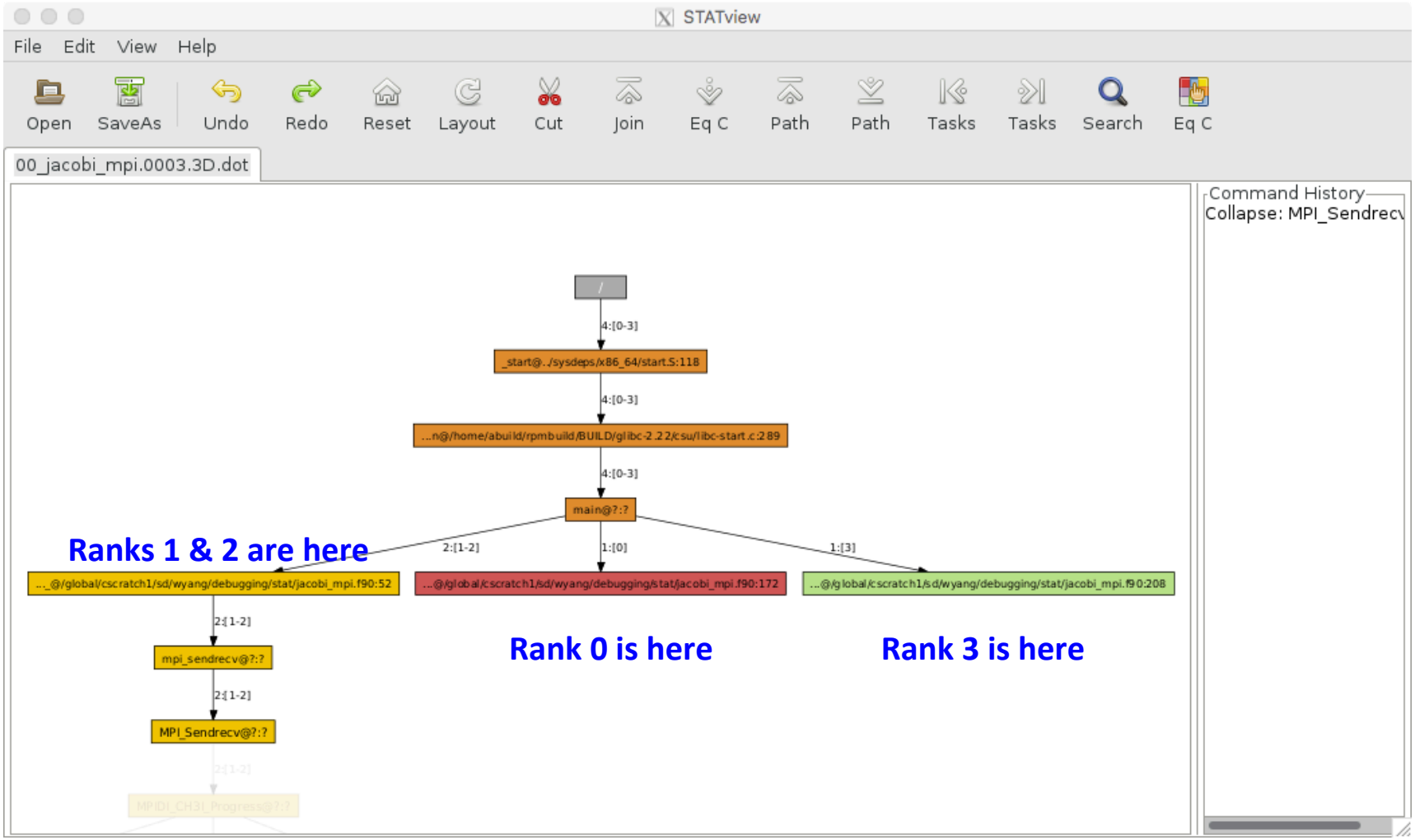
```
$ ftn -g -o jacobi_mpi jacobi_mpi.f90
$ salloc -N 2 -t 30:00 -q debug -C knl
...
$ srun -n 4 ... ./jacobi_mpi &
[1] 142927
$ module load stat
$ stat-cl -i 142927
…
Attaching to application...
Attached!
Application already paused... ignoring request to pause
Sampling traces...
Traces sampled!
…
Resuming the application...
Resumed!
Merging traces...
Traces merged!
Detaching from application...
Detached!

Results written to /global/cscratch1/sd/wyang/debugging/stat/stat_results/jacobi_mpi.0003
$ ls -l stat_results/jacobi_mpi.0003/*.dot
-rw-r--r-- 1 wyang wyang 2724 Apr 23 17:34 stat_results/jacobi_mpi.0003/00_jacobi_mpi.0003.3D.dot
$ STATview stat_results/jacobi_mpi.0003/00_jacobi_mpi.0003.3D.dot
```

with usual optimization flags, if any

-i to get source line numbers

STAT samples stack backtraces a few times

Debug a hung application with STAT (Cont'd)



Cray ATP (Abnormal Termination Processing)

- **ATP gathers stack backtraces from all processes *when an application fails***
  - Invokes STAT underneath
  - Output in atpMergedBT.dot and atpMergedBT_line.dot (which shows source code line numbers), which are to be viewed with STATview
- **The atp module is loaded on Cori and Edison by default, but ATP is *not* enabled; to enable:**

```
export ATP_ENABLED=1    # sh/bash/ksh
setenv ATP_ENABLED 1    # csh/tcsh
```

- **Can get core dumps (core.atp.*jobid*.*rank*), too, by setting coredumpsize unlimited:**

```
ulimit -c unlimited              # sh/bash/ksh
unlimit coredumpsize             # csh/tcsh
```

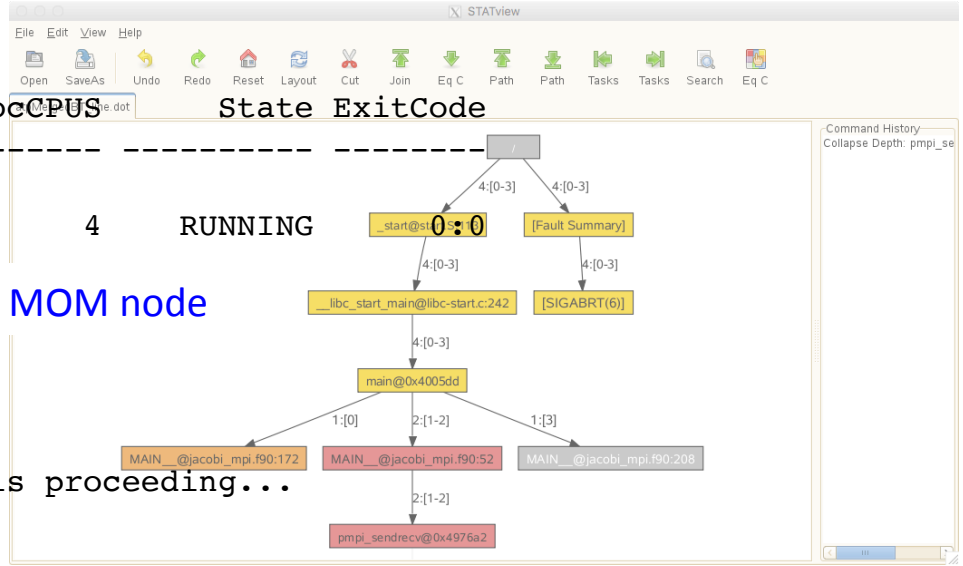
  but they do *not* represent the exact same moment in time (therefore the location of a failure can be inaccurate)

- **For more info**
  - 'intro_atp' man page
  - http://www.nersc.gov/users/software/debugging-and-profiling/stat-and-atp/

Use ATP for hung application

- **Abort a hung application to generate backtraces**
- **For the following to work, must have used**
  - 'export ATP_ENABLED=1' in batch script
  - 'export FOR_IGNORE_EXCEPTIONS=true' in batch script for Intel Fortran
  - '-f no-backtrace' at compile/link time for GNU Fortran

```
$ sacct -j 11091158        Find the job step ID
 JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------ ---------- ---------- ---------- ---------- ---------- --------
...
11091158.0   jacobi_mp+           nstaff          4    RUNNING      0:0
...
$ ssh cmom02               Kill the application on a MOM node
$ scancel -s ABRT 11091158.0
$ exit
$ cat slurm-11091158.out
Application 11091158 is crashing. ATP analysis proceeding...
...
Process died with signal 6: 'Aborted'
View application merged backtrace tree with: stat-view atpMergedBT.dot
...
$ module load stat
$ STATview atpMergedBT.dot    # or statview atpMergedBT_line.dot
```



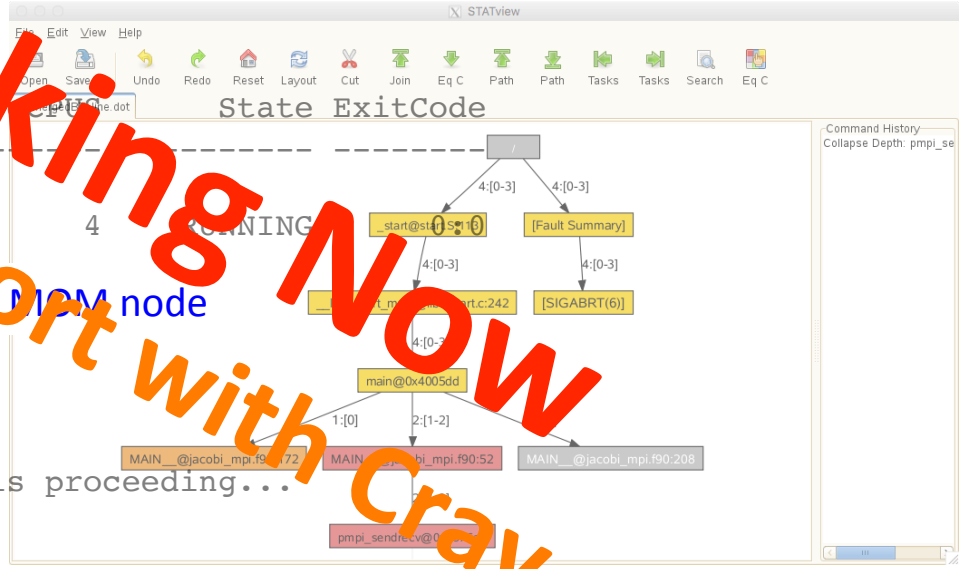
Use ATP for hung application (Cont'd)

- Force to generate backtraces from a hung application
- For the following to work, must have used
  - 'export ATP_ENABLED=1' in batch script
  - 'export FOR_IGNORE_EXCEPTIONS=true' in batch script for Intel Fortran
  - '-fno-backtrace' at compile/link time for GNU Fortran

Not Working Now

Filed a bug report with Cray

```
$ sacct -j 11091158                                    Find the job step ID
 JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------ ---------- ---------- ---------- ---------- ---------- --------
...
11091158.0   jacobi_mp+              nstaff          4    RUNNING      0:0
...
$ ssh cmom02                                  Kill the application on a MOM node
$ scancel -s ABRT 11091158.0
$ exit
$ cat slurm-11091158.out
Application 11091158 is crashing. ATP analysis proceeding...
...
Process died with signal 6: 'Aborted'
View application merged backtrace tree with: stat-view atpMergedBT.dot
...
$ module load stat
$ STATview atpMergedBT.dot    # or statview atpMergedBT_line.dot
```



Anything can be done in the meantime?

- **A script provided in stat_userguide (p.40)**
  - Launches the application in background
  - If the app runs much longer than expected (that is, it's possibly hanging), then run stat-cl on it
- **A practically same script below**
- **Be careful as run times can fluctuate**

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -t 10
#SBATCH -q debug
#SBATCH -C knl

stat_wait_time_minutes=5  # exp. run time in minutes
application_exited=0

#run the application and get the launcher PID
export OMP_NUM_THREADS=1
#…
srun -n 4 -c 128 ./a.out &
pid=$!
```

```
# periodically check for application exit
for i in `seq ${stat_wait_time_minutes}`
do
    sleep 60
    ps -p ${pid}
    if test $? -eq 1; then
        # the application exited, so we're done!
        application_exited=1
        break
    fi
done

# if the application is still running then invoke STAT
if test ${application_exited} -eq 0; then
    module load stat
    stat-cl -i ${pid}
    # kill -TERM ${pid}
fi
```

Anything can be done in the meantime? (Cont'd)


```
$ sbatch runit
Submitted batch job 11806533

$ cat slurm-11806533.out
...
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 /global/cscratch1/sd/wyang/debugging/stat/stat_results/jacobi_mpi.0005

srun: forcing job termination
srun: Job step aborted: Waiting up to 32 seconds for job step to finish.
slurmstepd: error: *** STEP 11806594.0 ON nid02517 CANCELLED AT 2018-04-23T20:27:32 ***

$ ls -l stat_results/jacobi_mpi.0005/00_jacobi_mpi.0005.3D.dot
-rw------- 1 wyang wyang 2684 Apr 23 20:27 stat_results/jacobi_mpi.0005/00_jacobi_mpi.0005.3D.dot

$ module load stat
$ STATview stat_results/jacobi_mpi.0005/00_jacobi_mpi.0005.3D.dot
```

Valgrind

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

Valgrind's memcheck

```
$ module load valgrind
$ ftn -dynamic -g -O0 memory_leaks.f $VALGRIND_MPI_LINK
$ salloc -N 1 -t 30:00 -q debug -C knl
$ srun -n 2 valgrind --leak-check=full --log-file=%p ./a.out
$ ls -l
...
-rw-r--r-- 1 wyang wyang   9131 Apr 23 21:09 157465
-rw-r--r-- 1 wyang wyang   7526 Apr 23 21:09 194650
```

The code from NERSC DDT webpage

Could have explicitly added '--tool=memcheck'

- **Let's look at the report for process 157465**

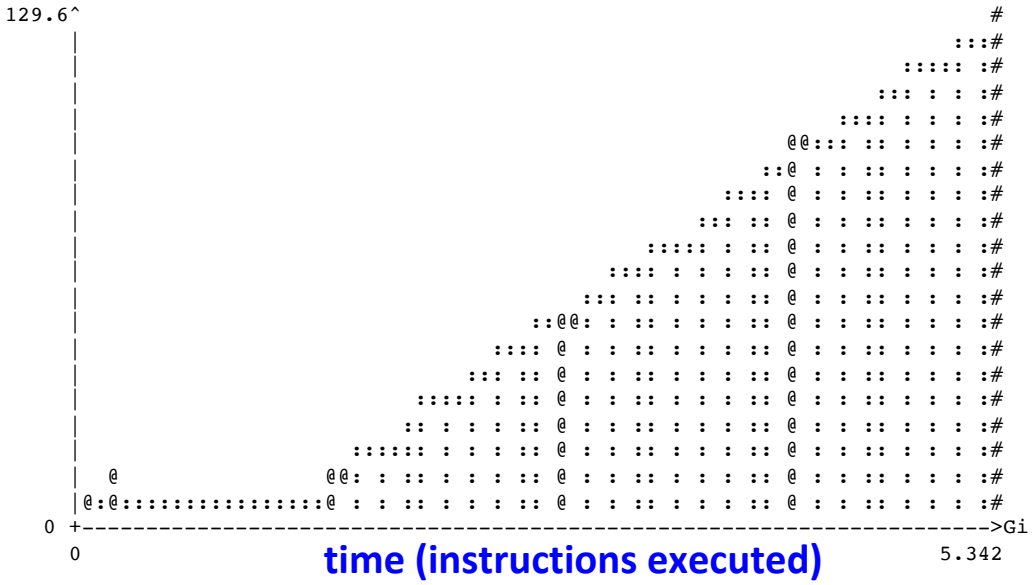
```
$ more 157465
...
==157465== LEAK SUMMARY:
==157465==    definitely lost: 109,052,984 bytes in 29 blocks
==157465==    indirectly lost: 0 bytes in 0 blocks
==157465==      possibly lost: 16,777,376 bytes in 4 blocks
==157465==    still reachable: 15,358 bytes in 35 blocks
==157465==         suppressed: 0 bytes in 0 blocks
...
```

- **Can suppress spurious error messages ("false positives") by using a suppression file (--suppressions=/path/to/directory/file)**
  - http://valgrind.org/docs/manual/manual-core.html#manual-core.suppress
  - Default suppression file: $VALGRIND_DIR/lib/valgrind/default.supp

Valgrind's massif

- **For profiling heap memory usage**

```
$ ftn -g -O2 memory_leaks.f
$ srun -n 2 -c 128 valgrind --tool=massif ./a.out
$ ls -lrt
…
-rw------- 1 wyang wyang  72612 Apr 23 21:25 massif.out.158228
-rw------- 1 wyang wyang 125809 Apr 23 21:25 massif.out.195354
$ ms_print massif.out.195354
...
```




```
Number of snapshots: 96
 Detailed snapshots: [3, 8, 16, 28, 35, 38, 43, 44, 46, 56, 66, 76, 86 (peak)]
...
```

':': normal snapshot; basic info provided

'@': detailed snapshot where detailed info is provided

'#': peak snapshot where the peak heap usage is

This example strongly suggests memory leaks

Valgrind's massif (Cont'd)

```
...
--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
 82    531,809,757       96,862,856       96,761,707       101,149            0
...
 86  5,730,387,202      135,866,296      135,694,102       172,194            0
99.87% (135,694,102B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->92.61% (125,830,320B) 0x50C4BDA: _mm_malloc (in /opt/intel/compilers_and_libraries_2018.
1.163/linux/compiler/lib/intel64_lin/libintlc.so.5)
| ->92.61% (125,830,320B) 0x40EE77: for_allocate (in /global/cscratch1/sd/wyang/debugging/
valgrind/a.out)
| | ->30.87% (41,943,440B) 0x408369: sub_bad_ (memory_leaks.f:41)
| | | ->30.87% (41,943,440B) 0x407DB4: MAIN__ (memory_leaks.f:18)
| | |   ->30.87% (41,943,440B) 0x407D1C: main (in /global/cscratch1/sd/wyang/debugging/val
grind/a.out)
| | |
| | ->30.87% (41,943,440B) 0x4085ED: sub_badx2_ (memory_leaks.f:51)
| | | ->30.87% (41,943,440B) 0x407DE7: MAIN__ (memory_leaks.f:21)
| | |   ->30.87% (41,943,440B) 0x407D1C: main (in /global/cscratch1/sd/wyang/debugging/val
grind/a.out)
| | |
| | ->30.87% (41,943,440B) 0x408850: sub_badx2_ (memory_leaks.f:54)
| |   ->30.87% (41,943,440B) 0x407DE7: MAIN__ (memory_leaks.f:21)
| |     ->30.87% (41,943,440B) 0x407D1C: main (in /global/cscratch1/sd/wyang/debugging/val
grind/a.out)
| |
| ->00.00% (0B) in 1+ places, all below ms_print's threshold (01.00%)
...
--------------------------------------------------------------------------------
  n        time(i)         total(B)   useful-heap(B) extra-heap(B)    stacks(B)
--------------------------------------------------------------------------------
 95  5,736,281,106      125,967,864      125,847,342       120,522            0
```

National Energy Research Scientific Computing Center