Debugging on Blue Waters

Debugging tools and techniques for Blue Waters are described here with example sessions, output, and pointers to small test codes. For tutorial purposes, this material will work best when applied via pair programming.

One person should have a copy of this presentation displayed and be the advisor/observer/idea-person. The other person should be logged into the system with a couple of ssh sessions (one for editing, one for running an interactive job).

Example commands and output are in `monospaced typewriter font`. Source code names are in blue where possible. Questions are in light green?

Test codes shown in this tutorial/how-to are available on the system in the directory shown. Ssh to the system with X11 forwarding enabled in order to work through the examples.

~arnoldg/bwdebug/

```
mkdir mydebug ; cp ~arnoldg/bwdebug/*  mydebug/ ; cd mydebug

xterm &

     # within xterm: qsub -I -V -lnodes=2:ppn=16:xk,walltime=00:55:00
```

Abnormal Termination Processing : ATP *(not adenosine triphosphate)*

This should be enabled for all of your applications. The module should be in your environment by default.

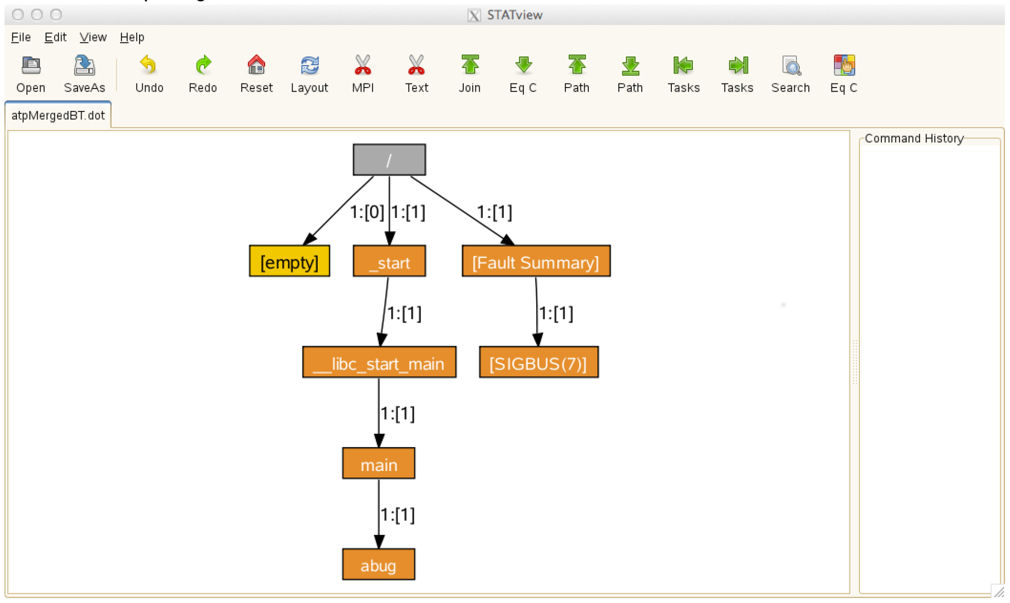
1. Make sure atp is shown in "module list"
2. rebuild the application if it wasn't build with the atp module
    a. `cc -g -o bugcG0 bugc.c`
    b. export ATP_ENABLED=1 # for your batch script or from within your interactive batch job
3. run in a batch job as usual (bugc.c) , if you don't already have an interactive job, obtain a short time job with:
    a. `qsub -I -V -lnodes=1:ppn=2,walltime=00:15:00`

```
jyc.ncsa.illinois.edu-[IN_JOB]arnoldg@nid00010:~/debug> aprun -n 2 bugcG0
Hello world! I'm 0 of 2
Hello world! I'm 1 of 2
Application 206080 is crashing. ATP analysis proceeding...

Stack walkback for Rank 1 starting:
  _start@start.S:113
  __libc_start_main@0x2aaab1b59c35
  main@bugc.c:40
  abug@bugc.c:62
Stack walkback for Rank 1 done
Process died with signal 7: 'Bus error'
Forcing core dumps of ranks 1, 0
View application merged backtrace tree with: statview atpMergedBT.dot
You may need to: module load stat

_pmiu_daemon(SIGCHLD): [NID 00028] [c0-0c0s1n2] [Mon Nov 25 13:57:21 2013] PE RANK 1 exit
signal Bus error
[NID 00028] 2013-11-25 13:57:21 Apid 206080: initiated application termination
Application 206080 exit codes: 135
Application 206080 resources: utime ~0s, stime ~0s, Rss ~8052, inblocks ~650, outblocks ~709
```

The code bugc.c resembles:

```
                total += abug(rank);
                total += abug(rank);
                MPI_Barrier(MPI_COMM_WORLD);
                total += abug(rank);
                total += abug(rank);
        }
        printf ("Hello world! I'm %d of %d with total=%ld\n", rank, size, total);
        MPI_Finalize();
        exit(0);
}

double abug(int rank)
{
        double a[1];
        if (rank == 1)
        {
           a[1]=3.5;
           sum += a[1];
           a[100]=4.0;
           sum += a[100];
           a[1000]=4.0;
           sum += a[1000];
           a[10000]=4.0;
```

4. From a login node, after the job has run, statview will show a graphic with the stack trace. Launch statview from a login node
   a. `module load stat`
   b. `statview atpMergedBT.dot &`



Example 2. thread_model_detect.c (`cc -G0 -o thread_model_detectG0 thread_model_detect.c`) with the MPI_Bcast() uncommented and the MPI_Irecv() and MPI_Wait() commented.

```
arnoldg@nid25432:~/debug> aprun -n 8 ./thread_model_detectG0
MPI_THREAD_SERIALIZED support detected
rank 0 of 8 on nid19884 core 0
rank 1 of 8 on nid19884 core 1
rank 3 of 8 on nid19884 core 3
rank 2 of 8 on nid19884 core 2
rank 5 of 8 on nid19884 core 5
rank 4 of 8 on nid19884 core 4
rank 6 of 8 on nid19884 core 6
rank 7 of 8 on nid19884 core 7
Rank 3 [Tue Nov 26 09:45:05 2013] [c13-7c0s6n2] Fatal error in PMPI_Bcast: Other MPI error, error stack:
PMPI_Bcast(1534)......: MPI_Bcast(buf=0x7fffffff9760, count=10, MPI_INT, root=3, MPI_COMM_WORLD) failed
MPIR_Bcast_impl(1372).:
MPIR_Bcast_intra(1163):
MPIR_SMP_Bcast(1080)..: Failure during collective
Application 2548001 is crashing. ATP analysis proceeding...

Stack walkback for Rank 3 starting:
  _start@start.S:113
  __libc_start_main@libc-start.c:226
  main@thread_model_detect.c:57
  PMPI_Bcast@0x2002a6d0
  MPIR_Err_return_comm@0x2005392c
  handleFatalError@0x20053759
  MPID_Abort@0x2005f249
  abort@abort.c:92
  raise@pt-raise.c:41
Stack walkback for Rank 3 done
Process died with signal 6: 'Aborted'
Forcing core dumps of ranks 3, 0
View application merged backtrace tree with: statview atpMergedBT.dot
You may need to: module load stat
_pmiu_daemon(SIGCHLD): [NID 19884] [c13-7c0s6n2] [Tue Nov 26 09:45:10 2013] PE RANK 3 exit signal Aborted
[NID 19884] 2013-11-26 09:45:10 Apid 2548001: initiated application termination
Application 2548001 exit codes: 134
Application 2548001 resources: utime ~0s, stime ~0s, Rss ~7160, inblocks ~9614, outblocks ~24397
arnoldg@nid25432:~/debug> cat -n thread_model_detect.c

    54          if (rank == 3)
    55          {
    56                  int array[10];
    57                  MPI_Bcast( array, 10, MPI_INT, rank, MPI_COMM_WORLD);
    58          }
    59          MPI_Finalize();
    60  }
```

Example 3. nan.f (`ftn -K trap=fp -G0 -o nan_trap nan.f`)

```
arnoldg@nid25356:~/debug> aprun -n 1 ./nan_trap
Application 2548029 is crashing. ATP analysis proceeding...

Stack walkback for Rank 0 starting:
  _start@start.S:113
  __libc_start_main@libc-start.c:226
  main@nan.f:11
Stack walkback for Rank 0 done
Process died with signal 8: 'Floating point exception'
Forcing core dump of rank 0
View application merged backtrace tree with: statview atpMergedBT.dot
You may need to: module load stat

_pmiu_daemon(SIGCHLD): [NID 03311] [c4-3c2s7n1] [Tue Nov 26 11:01:25 2013] PE RANK 0 exit signal Floating
point exception
Application 2548029 exit codes: 136
Application 2548029 resources: utime ~0s, stime ~0s, Rss ~7160, inblocks ~10221, outblocks ~26850
arnoldg@nid25356:~/debug>

# ftn -o nan nan.f

arnoldg@nid25356:~/debug> aprun -n 1 ./nan
 0.
Application 2548234 resources: utime ~0s, stime ~0s, Rss ~7160, inblocks ~10216, outblocks ~26828
arnoldg@nid25356:~/debug>

arnoldg@nid25356:~/debug> cat -n nan.f
     1          program main
     2          include 'mpif.h'
     3
     4          real xd
     5          real yd
     6          integer ierror
     7
     8          call MPI_INIT(ierror)
     9          xd = 0.0
    10          yd = 0.0
    11          print *, xd/yd
    12          call MPI_FINALIZE(ierror)
    13          end
```

Why might the optimized version of nan.f run without causing a floating point trap? What are the consequences to the potential algorithm of allowing this?

The bugc.c and nan.f codes run to completion when compiled with optimization (the default for the Cray compilers). Optimization level can change runtime behavior.

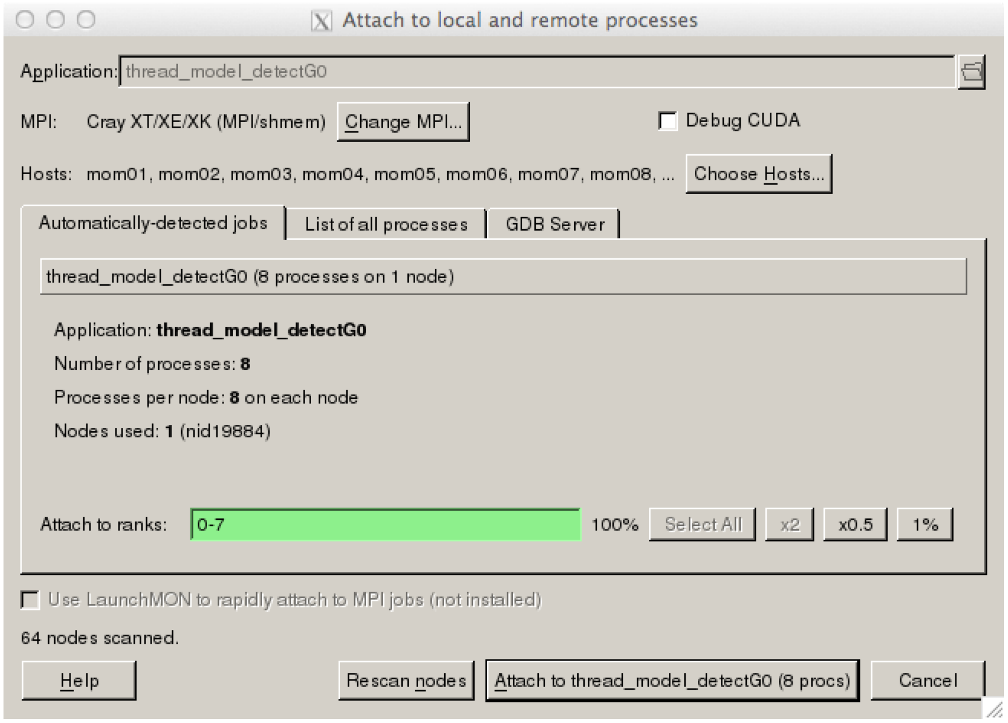
Allinea: DDT debugger

The DDT debugger is a scalable and runs with X11.

1. Build thread_model_detect.c code with -g or -G0 if you want to see source code display from DDT (though you can still debug otherwise). Comment out the MPI_Bcast() if you uncommented it for the ATP example and rebuild with MPI_Irecv() and MPI_Wait() enabled as shown.

```
        if (rank == 3)
        {
                int array[10];
// MPI_Bcast() this will causes a collective failure and abnormal exit
//              MPI_Bcast( array, 10, MPI_INT, rank, MPI_COMM_WORLD);

                MPI_Request myrequest;
                MPI_Status mystatus;
                MPI_Irecv( array, 10, MPI_INT,0 /*source*/,
                123 /*tag*/, MPI_COMM_WORLD, &myrequest);

                MPI_Wait(&myrequest, &mystatus);
        }
        MPI_Finalize();
```

2. run code in batch as usual (batch script or your interactive batch job)
3. from a fresh session on one of the login nodes
    a. `module load ddt`
    b. `ddt # choose Attach on the initial window`


Attach to local and remote processes
Change MPI...
Choose Hosts...
Automatically-detected jobs
List of all processes
GDB Server
Select All
x2
x0.5
1%
Help
Rescan nodes
Attach to thread_model_detectG0 (8 procs)
Cancel

Application: thread_model_detectG0

MPI: Cray XT/XE/XK (MPI/shmem)

Debug CUDA

Hosts: mom01, mom02, mom03, mom04, mom05, mom06, mom07, mom08, ...

thread_model_detectG0 (8 processes on 1 node)

Application: **thread_model_detectG0**

Number of processes: **8**

Processes per node: **8** on each node

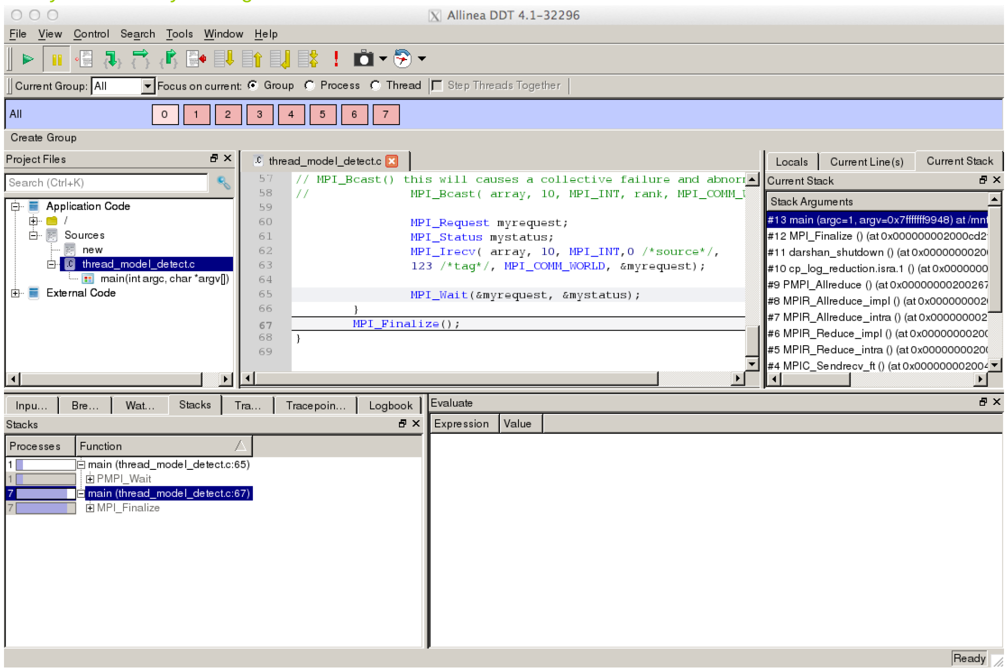
Nodes used: **1** (nid19884)

Attach to ranks: 0-7 100%

Use LaunchMON to rapidly attach to MPI jobs (not installed)

64 nodes scanned.

    c. Select the Attach to ... box if the information looks correct . The GUI should show the location of the various ranks. The Current Stack tab has been selected to show the progress of rank0. The remaining 7 ranks are all at the same location. What routine or library call are they waiting in?



Allinea DDT 4.1-32296

Current Group: All · Focus on current: Group (selected), Process, Thread · Step Threads Together

```
// MPI_Bcast() this will causes a collective failure and abnor
//              MPI_Bcast( array, 10, MPI_INT, rank, MPI_COMM_

                MPI_Request myrequest;
                MPI_Status mystatus;
                MPI_Irecv( array, 10, MPI_INT,0 /*source*/,
                123 /*tag*/, MPI_COMM_WORLD, &myrequest);

                MPI_Wait(&myrequest, &mystatus);
        }
        MPI_Finalize();
}
```

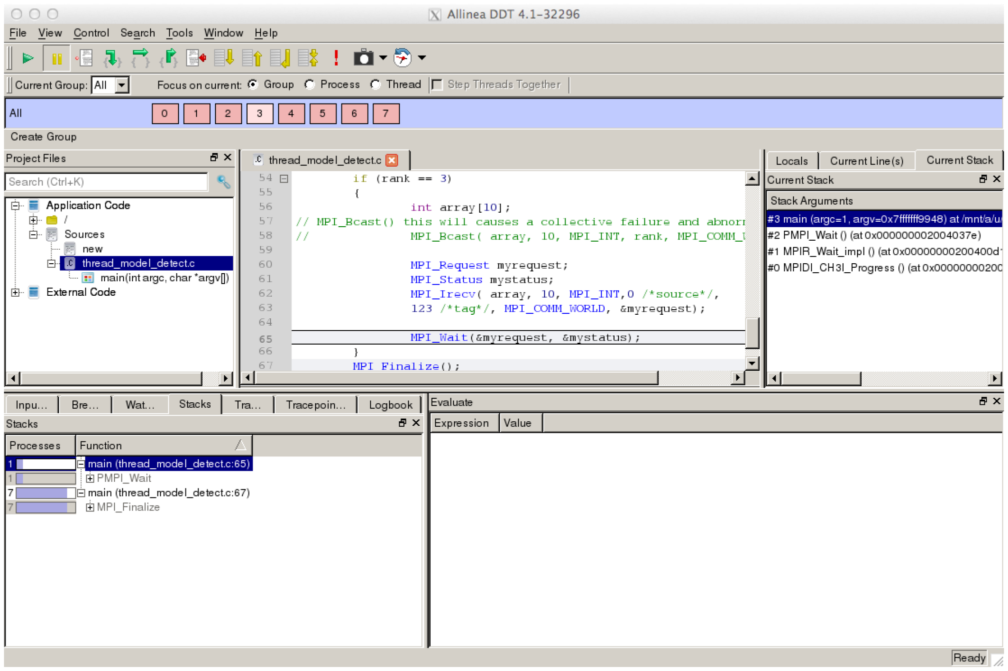
Current Stack:

```
#13 main (argc=1, argv=0x7fffffff9948) at /mnt
#12 MPI_Finalize () (at 0x000000002000cd2
#11 darshan_shutdown () (at 0x00000000020
#10 cp_log_reduction.isra.1 () (at 0x0000000
#9 PMPI_Allreduce () (at 0x00000000200267
#8 MPIR_Allreduce_impl () (at 0x00000000002
#7 MPIR_Allreduce_intra () (at 0x000000002
#6 MPIR_Reduce_impl () (at 0x00000000020
#5 MPIR_Reduce_intra () (at 0x00000000020
#4 MPIC_Sendrecv_ft () (at 0x0000000002004
```

Stacks:

Processes	Function
1	main (thread_model_detect.c:65)
1	PMPI_Wait
7	main (thread_model_detect.c:67)
7	MPI_Finalize

d. Rank 3 is at MPI_Wait() as expected, and the program is therefore hanging.



Allinea DDT 4.1-32296

Current Group: All · Focus on current: Group (selected), Process, Thread · Step Threads Together

```
        if (rank == 3)
        {
                int array[10];
// MPI_Bcast() this will causes a collective failure and abnor
//              MPI_Bcast( array, 10, MPI_INT, rank, MPI_COMM_

                MPI_Request myrequest;
                MPI_Status mystatus;
                MPI_Irecv( array, 10, MPI_INT,0 /*source*/,
                123 /*tag*/, MPI_COMM_WORLD, &myrequest);

                MPI_Wait(&myrequest, &mystatus);
        }
        MPI_Finalize();
```

Current Stack:

```
#3 main (argc=1, argv=0x7fffffff9948) at /mnt/a/u
#2 PMPI_Wait () (at 0x000000002004037e)
#1 MPIR_Wait_impl () (at 0x00000000200400d
#0 MPIDI_CH3I_Progress () (at 0x0000000000200
```

Stacks:

Processes	Function
1	main (thread_model_detect.c:65)
1	PMPI_Wait
7	main (thread_model_detect.c:67)
7	MPI_Finalize

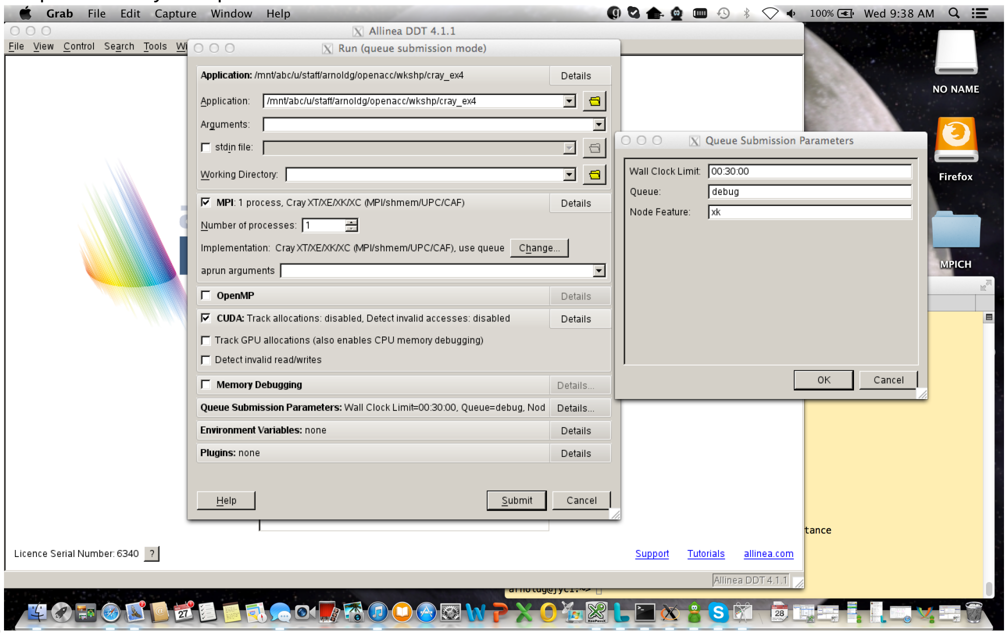
e. Why does the misuse of MPI_Bcast() result in a program crash while the MPI_Irecv() and MPI_Wait() version of the code hangs?

Example 2. OpenACC code with XK node and GPU.

1. Build cray_ex4.f90 with OpenACC support
   a. `ftn -g -h acc,msgs -o cray_ex4 cray_ex4.f90`

```
!$acc parallel loop
ftn-6405 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 12
  A region starting at line 12 and ending at line 18 was placed on the accelerator.
ftn-6416 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 12
  If not already present: allocate memory and copy whole array "a" to accelerator, copy back
at line 18 (acc_copy).
ftn-6416 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 12
  If not already present: allocate memory and copy whole array "b" to accelerator, copy back
at line 18 (acc_copy).
ftn-6416 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 12
  If not already present: allocate memory and copy whole array "c" to accelerator, copy back
at line 18 (acc_copy).
...
ftn-6415 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 48
  Allocate memory and copy variable "total" to accelerator, copy back at line 55 (acc_copy).

                 DO j = 1,M
ftn-6430 crayftn: ACCEL TEST_OPENACC, File = cray_ex4.f90, Line = 51
  A loop starting at line 51 was partitioned across the threadblocks and the 128 threads
within a threadblock.

Cray Fortran : Version 8.2.1 (u82093f82212i82224p82394a82022e82011z82394)
Cray Fortran :               (x8232r82023w82008t8213b82042k82020)
Cray Fortran : Wed Nov 27, 2013  09:15:01
Cray Fortran : Compile time:  0.0840 seconds
Cray Fortran : 66 source lines
Cray Fortran : 0 errors, 0 warnings, 21 other messages, 0 ansi
```

2. Setup ddt to run your OpenACC code


Grab File Edit Capture Window Help
Wed 9:38 AM
Allinea DDT 4.1.1
File View Control Search Tools Wi
NO NAME
Firefox
MPICH
Licence Serial Number: 6340
Support Tutorials allinea.com

Run (queue submission mode)

Application: /mnt/abc/u/staff/arnoldg/openacc/wkshp/cray_ex4

Application: /mnt/abc/u/staff/arnoldg/openacc/wkshp/cray_ex4

Arguments:

☐ stdin file:

Working Directory:

☑ **MPI:** 1 process, Cray XT/XE/XK/XC (MPI/shmem/UPC/CAF)

Number of processes: 1

Implementation: Cray XT/XE/XK/XC (MPI/shmem/UPC/CAF), use queue

aprun arguments

☐ **OpenMP**

☑ **CUDA:** Track allocations: disabled, Detect invalid accesses: disabled

☐ Track GPU allocations (also enables CPU memory debugging)

☐ Detect invalid read/writes

☐ **Memory Debugging**

Queue Submission Parameters: Wall Clock Limit=00:30:00, Queue=debug, Nod

Environment Variables: none

Plugins: none

Queue Submission Parameters

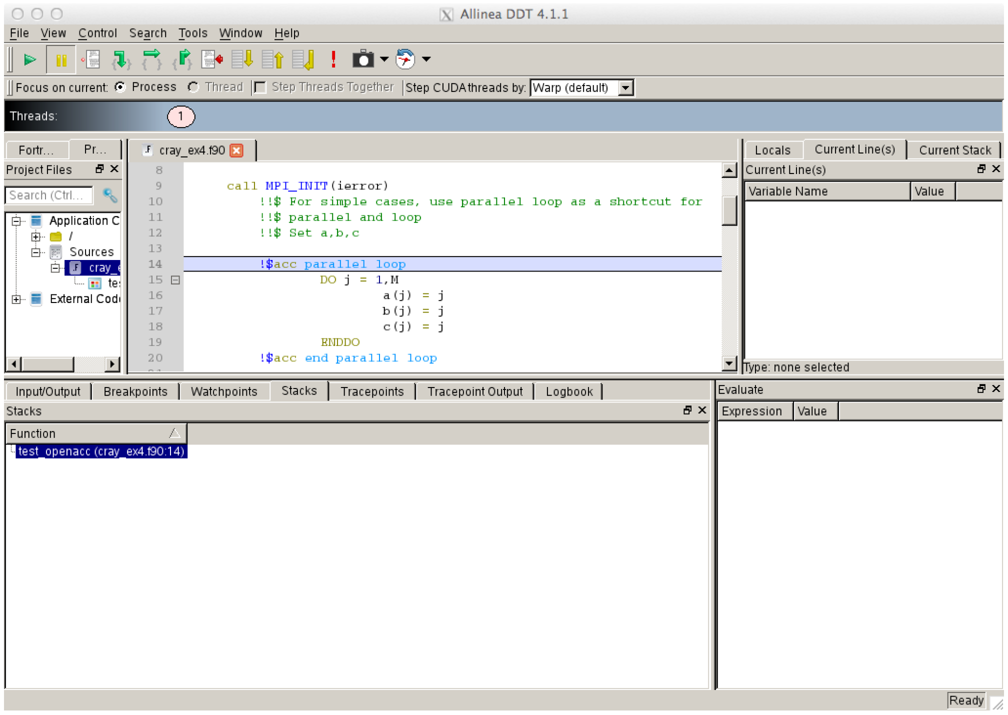
Wall Clock Limit: 00:30:00

Queue: debug

Node Feature: xk

.

3. The program is initialized and stopped at the first accelerated region-- waiting to be run with the Play green-arrow at top left.


Allinea DDT 4.1.1

Focus on current: Process | Step CUDA threads by: Warp (default)

Threads: 1

cray_ex4.f90

```
    call MPI_INIT(ierror)
       !!$ For simple cases, use parallel loop as a shortcut for
       !!$ parallel and loop
       !!$ Set a,b,c

       !$acc parallel loop
            DO j = 1,M
                  a(j) = j
                  b(j) = j
                  c(j) = j
            ENDDO
       !$acc end parallel loop
```

Current Line(s)

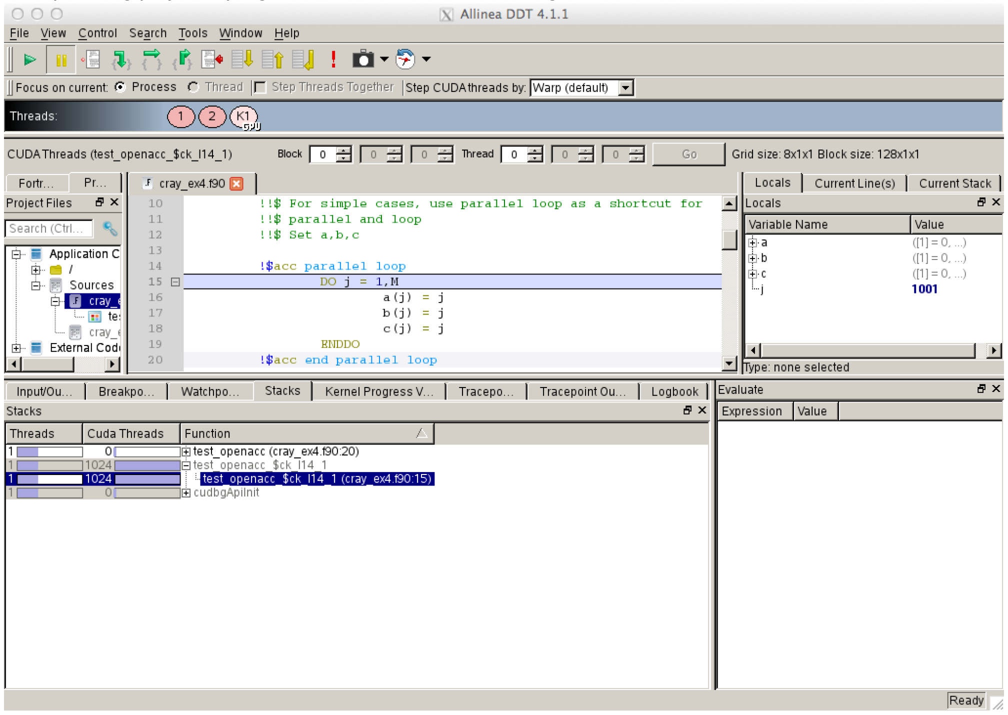
Type: none selected

Stacks

Function
test_openacc (cray_ex4.f90:14)

Ready

4. After pressing play, the program enters the accelerated region.


Allinea DDT 4.1.1

Focus on current: Process | Step CUDA threads by: Warp (default)

Threads: 1 2 K1

CUDA Threads (test_openacc_$ck_l14_1) Block 0 0 0 Thread 0 0 0 Go Grid size: 8x1x1 Block size: 128x1x1

cray_ex4.f90

```
       !!$ For simple cases, use parallel loop as a shortcut for
       !!$ parallel and loop
       !!$ Set a,b,c

       !$acc parallel loop
            DO j = 1,M
                  a(j) = j
                  b(j) = j
                  c(j) = j
            ENDDO
       !$acc end parallel loop
```

Locals

Variable Name	Value
a	([1] = 0, ...)
b	([1] = 0, ...)
c	([1] = 0, ...)
j	1001

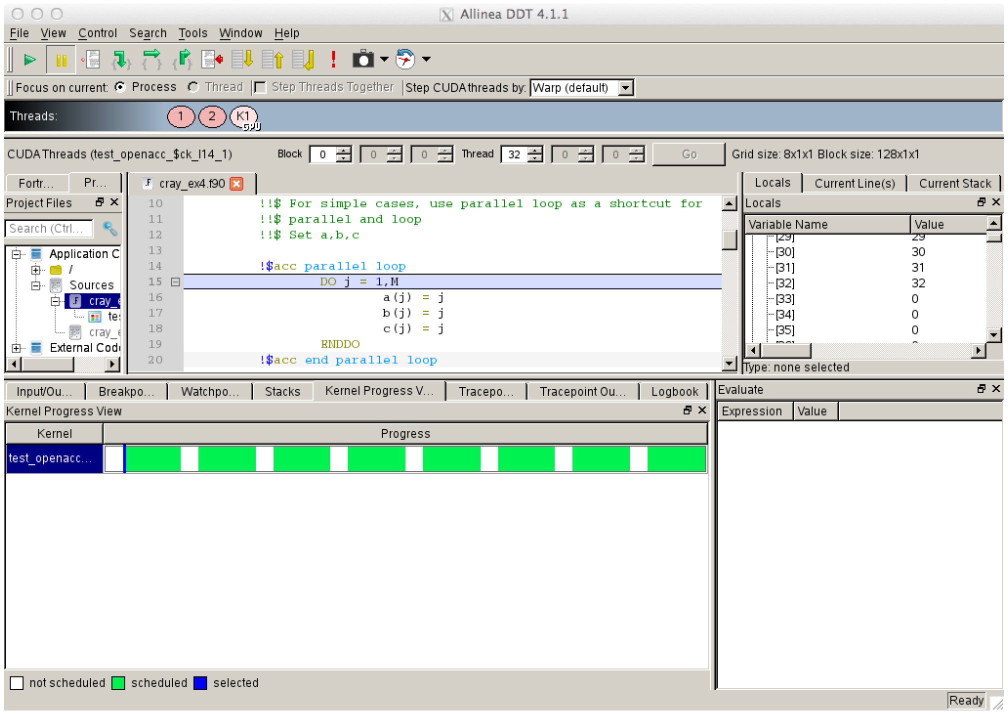
Type: none selected

Stacks

Threads	Cuda Threads	Function
1	0	test_openacc (cray_ex4.f90:20)
1	1024	test_openacc_$ck_l14_1
1	1024	test_openacc_$ck_l14_1 (cray_ex4.f90:15)
1	0	cudbgApiInit

Ready

5. Pressing play again moves through executions of warps. Note the progress in the array assignments.


Allinea DDT 4.1.1
File View Control Search Tools Window Help
Threads: 1 2 K1 GPU
Fortr… Pr… Project Files
Search (Ctrl…
Application C / Sources cray_ te: cray_ External Code
cray_ex4.f90
Locals Current Line(s) Current Stack
Input/Ou… Breakpo… Watchpo… Stacks Kernel Progress V… Tracepo… Tracepoint Ou… Logbook
Kernel Progress View
Kernel Progress
test_openacc…
Evaluate
Expression Value
Ready

Focus on current: Process Thread Step Threads Together Step CUDA threads by: Warp (default)

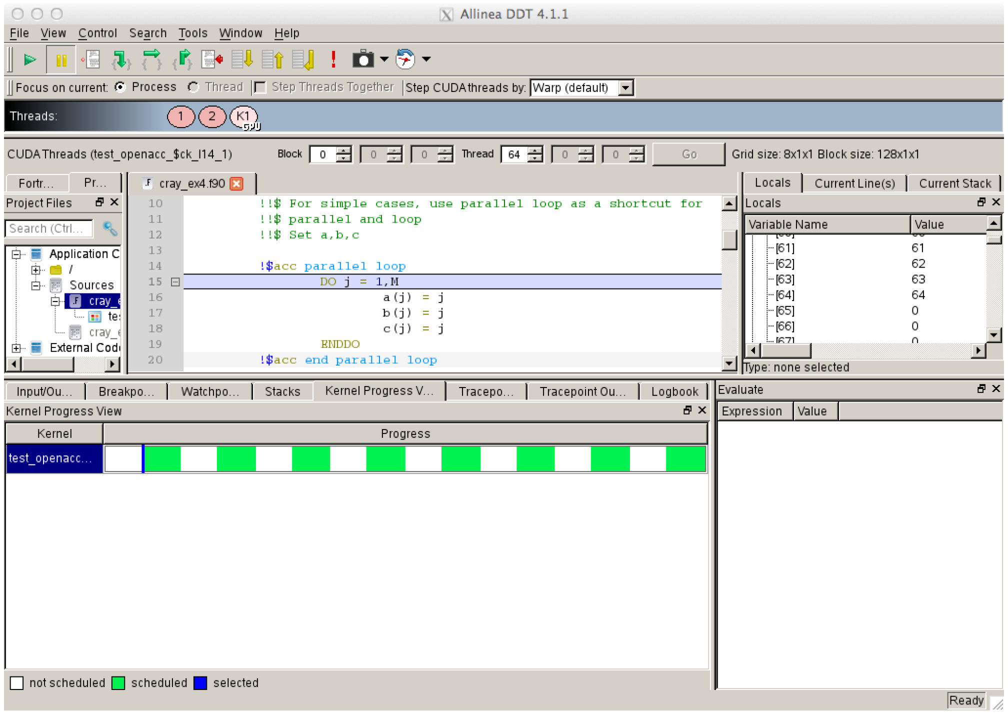
CUDA Threads (test_openacc_$ck_l14_1) Block 0 0 0 Thread 32 0 0 Go Grid size: 8x1x1 Block size: 128x1x1

```
      !!$ For simple cases, use parallel loop as a shortcut for
      !!$ parallel and loop
      !!$ Set a,b,c

      !$acc parallel loop
              DO j = 1,M
                      a(j) = j
                      b(j) = j
                      c(j) = j
              ENDDO
      !$acc end parallel loop
```

Variable Name	Value
[29]	29
[30]	30
[31]	31
[32]	32
[33]	0
[34]	0
[35]	0

Type: none selected

□ not scheduled 🟩 scheduled 🟦 selected


Allinea DDT 4.1.1
File View Control Search Tools Window Help
Threads: 1 2 K1 GPU
Fortr… Pr… Project Files
Search (Ctrl…
Application C / Sources cray_ te: cray_ External Code
cray_ex4.f90
Locals Current Line(s) Current Stack
Input/Ou… Breakpo… Watchpo… Stacks Kernel Progress V… Tracepo… Tracepoint Ou… Logbook
Kernel Progress View
Kernel Progress
test_openacc…
Evaluate
Expression Value
Ready

Focus on current: Process Thread Step Threads Together Step CUDA threads by: Warp (default)

CUDA Threads (test_openacc_$ck_l14_1) Block 0 0 0 Thread 64 0 0 Go Grid size: 8x1x1 Block size: 128x1x1

```
      !!$ For simple cases, use parallel loop as a shortcut for
      !!$ parallel and loop
      !!$ Set a,b,c

      !$acc parallel loop
              DO j = 1,M
                      a(j) = j
                      b(j) = j
                      c(j) = j
              ENDDO
      !$acc end parallel loop
```

Variable Name	Value
[61]	61
[62]	62
[63]	63
[64]	64
[65]	0
[66]	0

Type: none selected

□ not scheduled 🟩 scheduled 🟦 selected

6. Compare to Cray's CRAY_ACC_DEBUG runtime environment kernel tracing. Do the number mouse presses on DDT's Play button match up with the number of kernel executions shown by CRAY_ACC_DEBUG ?

```
arnoldg@jyc1:~/openacc/wkshp> qsub -I -V -lnodes=1:ppn=4:xk,walltime=00:10:00
Job submitted to account: fyy
qsub: waiting for job 98630.bw-esms2 to start

[IN_JOB]arnoldg@nid00014:~/openacc/wkshp> \
export CRAY_ACC_DEBUG=1

[IN_JOB]arnoldg@nid00014:~/openacc/wkshp> aprun -n 1 ./cray_ex4
ACC: Transfer 3 items (to acc 12000 bytes, to host 0 bytes) from cray_ex4.f90:14
ACC: Execute kernel test_openacc_$ck_L14_1 async(auto) from cray_ex4.f90:14
ACC: Wait async(auto) from cray_ex4.f90:20
ACC: Transfer 3 items (to acc 0 bytes, to host 12000 bytes) from cray_ex4.f90:20
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from cray_ex4.f90:23
ACC: Execute kernel test_openacc_$ck_L23_2 async(auto) from cray_ex4.f90:23
ACC: Wait async(auto) from cray_ex4.f90:29
ACC: Transfer 1 items (to acc 0 bytes, to host 4000 bytes) from cray_ex4.f90:29
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from cray_ex4.f90:32
ACC: Execute kernel test_openacc_$ck_L32_3 async(auto) from cray_ex4.f90:32
ACC: Wait async(auto) from cray_ex4.f90:38
ACC: Transfer 1 items (to acc 0 bytes, to host 4000 bytes) from cray_ex4.f90:38
ACC: Transfer 3 items (to acc 8000 bytes, to host 0 bytes) from cray_ex4.f90:41
...
ACC: Wait async(auto) from cray_ex4.f90:57
ACC: Transfer 4 items (to acc 0 bytes, to host 8 bytes) from cray_ex4.f90:57
 Result:  500500
 Verified:  T
Application 206251 resources: utime ~0s, stime ~0s, Rss ~92864, inblocks ~637, outblocks ~976
```

The DDT setup for our Cray is configured to work with MPI codes. It's simplest to wrap any non-MPI (serial or GPU) code with MPI_Init() and MPI_Finalize() even if you have not yet ported to MPI.

Alternative approaches (MPMD example with hello_world and strace)

1. Simple mpi code hello_world.c:

```
#include <stdio.h>
#include <mpi.h>
#include <sched.h>

int main(int argc, char *argv[])
{
        int             rank, size, len, core;
        char            name[MPI_MAX_PROCESSOR_NAME];
        MPI_Init(argc, argv);

        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &size);

        MPI_Get_processor_name(name, &len);

        core= sched_getcpu();
        printf ("rank %d of %d on %s core %d\n", rank, size, name, core);

        MPI_Finalize();
}
```

2. Goal: Obtain strace data from just a single rank to gain insights into how time is spent in system calls (the stime field from aprun's final output) for that rank.

```
[IN_JOB]arnoldg@nid00010:~/c> aprun -n 1 strace -c hello_world : -n 3 hello_world

rank 0 of 4 on nid00046 core 0
rank 2 of 4 on nid00047 core 1
rank 1 of 4 on nid00047 core 0
rank 3 of 4 on nid00047 core 2
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 53.45    0.140009      140009         1           wait4
 45.94    0.120351        3086        39           fadvise64
  0.61    0.001587           5       328       281 open
  0.01    0.000020           0        83           read
  0.00    0.000000           0         6           write
  0.00    0.000000           0        54           close
  0.00    0.000000           0        51        41 stat

...
  0.00    0.000000           0         1           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00    0.261967                   861       327 total
Application 206271 resources: utime ~0s, stime ~1s, Rss ~17300, inblocks ~1788, outblocks ~1285
```

How would you setup a job to run a single rank compiled with gprof support (-pg) so that you have some basic timing information by routine for just that rank?

Strace defaults to tracing each system call along with its arguments. The output can run long even for trivial programs, but if you're having trouble isolating where a bug or hang occurs, it can be quite useful in some scenarios. The aprun MPMD launch method could be used with other tools where you only want to trace/time/instrument a few ranks out of the total program.

Related articles

Debugging on Blue Waters