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.

```bash
~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

```
joc.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
Application 206080 exit codes: 135
Application 206080 resources: utime ~0s, stime ~0s, Rss ~8052, inblocks ~650, outblocks ~709
```

The code `bugc.c` resembles:
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 &`

   ![Statview Graphic]

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

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)
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?

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();

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

   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?

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

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`
1. **!$acc parallel loop**
   - A region starting at line 12 and ending at line 18 was placed on the accelerator.
   - If not already present: allocate memory and copy whole array "a" to accelerator, copy back at line 18 (acc_copy).
   - If not already present: allocate memory and copy whole array "b" to accelerator, copy back at line 18 (acc_copy).
   - If not already present: allocate memory and copy whole array "c" to accelerator, copy back at line 18 (acc_copy).
   - ...  
   - Allocate memory and copy variable "total" to accelerator, copy back at line 55 (acc_copy).

   ```fortran
   DO j = 1, M
   A loop starting at line 51 was partitioned across the threadblocks and the 128 threads within a threadblock.
   ```

2. Setup ddt to run your OpenACC code

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

4. Pressing play again moves through executions of warps. Note the progress in the array assignments.
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?
Alternative approaches (MPMD example with hello_world and strace)

1. Simple mpi code hello_world.c:

```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.
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