SE-CSE13

Group Discussion Write-up

Debugging – Session 1

Thomas Epperly	LLNL
Zvonimir Racamaric’	University of Utah
Alan Humphrey	University of Utah

General Discussion Topics: Debugging CSE software at large scale; techniques, tools and approaches used by discussion group members and general experiences.

NOTE: this group was the result of a split from a larger discussion group and was coincidentally comprised solely of people with Computer Science (SE) backgrounds.

Overarching Problem and Focus of Discussion: Many bugs in scientific codes, computational frameworks, etc, only manifest at scales that surpass the capabilities of even the most advanced debuggers. Others, including non-deterministic bugs and concurrency-related issues, e.g. race conditions and deadlocks are also not easily identified or localized with traditional debugging tools and techniques. This being said, our discussion focused on each individuals experiences with debugging on large-scale HPC systems. A brief summary of each topic follows:

1.) With potentially thousands of nodes on modern and emerging large-scale HPC systems, the initial question when debugging software becomes one related to fault tolerance and resilience; it needs to be determined initially if the issue/bug is actually a hardware or software problem.

2.) Even modern debuggers, Allinea DDT and Totalview for example, only work to a few thousand cores and it's proven somewhat unrealistic to get an interactive session for 16,000+ cores, etc on large-scale systems.

3.) LLNL has developed the Stack Trace Analysis Tool (STAT) to help identify groups of processes in a parallel application that exhibit similar behavior. A single representative of these groups can then be examined with a parallel debugger like DDT or TotalView.

4.) The University of Utah has also done work using similar techniques using stack trace analysis (the topic of the Utah submission to SE-CSE) to instrument the Uintah Computational Framework to help locate bugs.

5.) The University of Utah has also done work with dynamic formal verification tools to detect the presence of deadlocks in MPI C/C++ and Fortran applications. These techniques work on up to a few hundred MPI processes currently, making the stack trace tools far more attractive.

6.) Much work involving schedule perturbation has also been researched and explored by all parties present.

7.) Another somewhat primitive but effective technique used by the Uintah framework in some cases is to throw the application into an infinite loop when an exception is thrown for instance, and the framework infrastructure then provides a process ID for the offending MPI rank. Using this information, a developer can SSH into a back node and attach a command line debugger such as GDB to the given process, walk up the stack and examine the state of the system at the time the error occurred.

8.) Other root cause analysis techniques explored at the University of Wisconsin, and now at LLNL use statistical debugging methods to instrument programs to record program behaviors at run time. An offline statistical analysis then identifies the program behaviors that are highly correlated with program failure. Work in this area has shown that an efficient feedback-collection strategy can capture results from up to 500,000 processes with relatively small amounts of storage (~50MB).

9.) Alas, “crude methods” such as the age-old printf() are still commonly used when other methods fail.

Summary: Although this group was small and somewhat biased based on the lack of diversity in backgrounds, the discussion was helpful in terms of providing a forum to discuss various approaches and tools used in debugging real scientific codes at large scale, as well as a way to share our individual experiences and opinions on each.