Allinea DDT as a Parallel Debugging Alternative to Totalview

K.B. Antypas
NERSC
Ernest Orlando Lawrence Berkeley National Laboratory
Berkeley, CA 94707
kantypas@lbl.gov

Abstract

As the cost and complexity of Totalview has increased over the years, scientific computing centers have started searching for a viable parallel debugging alternative. DDT (Distributed Debugging Tool) from Allinea Software is a relatively new parallel debugging tool which aims to provide much of the same functionality as Totalview. This review outlines the basic features and limitations of DDT to determine if it can be a reasonable substitute for Totalview. DDT was tested on the NERSC platforms Bassi, Seaborg, Jacquard and Davinci with Fortran90, C, and C++ codes using MPI and OpenMP for parallelism.

1. Introduction

Totalview, from the Etnus Corporation, is a sophisticated and feature rich software debugger for parallel applications. As Totalview has gained in popularity and market share its pricing model has increased to the point where it is often prohibitively expensive for massively parallel supercomputers. Additionally, many of Totalview's advanced features are not used by members of the scientific computing community. For these reasons, supercomputing centers have begun to search for a basic parallel debugging tool which can be used as an alternative to Totalview.

DDT (Distributed Debugging Tool) from the Allinea Corporation is a parallel debugger which provides many of the same basic features as Totalview, as well as some new elements. While Totalview has a much larger feature set than DDT, with debugging capability for more than one executable at a time, machine level language support, Tcl command line options, and other advanced components, these are not the primary reasons why scientists need a parallel debugger. Most scientists need a simple and user friendly way to set breakpoints, step through code and halt execution while they examine variable values and code logic across different processors.

This review examines and compares the basic debugging features of DDT and Totalview which are most commonly used in scientific computing. Four different codes were used to test the features, interface, and usability of DDT on the NERSC platforms, Bassi an IBM Power5, Seaborg an IBM SP, both running AIX, Jacquard, an Opteron Linux cluster running MVAPICH and Suse 9 and Davinci an SGI Altix running SGI ProPack 4, based on SUSE Linux Enterprise Server 9. The test codes included GTC (5), a Fortran90, NERSC benchmark code, FLASH (4), a mixed Fortran90/C adaptive mesh code running with the HDF5 parallel IO

This work was supported by the Director, Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

library, a C++ Chombo (3) Poisson solve application, also using an adaptive mesh, but as a library, and finally a very simple OpenMP example code to test the basic capabilities of DDT to handle multi-threaded code. Both DDT and Totalview come with memory analysis tools, packages which help visualize data, core file debugging capabilities, and methods to attach to running processes. These features are not reviewed here.

2. Comparison of Basic Features

Both DDT and Totalview provide the basic features of any serial debugger. A user can set breakpoints, step through a program and into subroutines, view stack frames, evaluate expressions, dive into arrays, dereference pointers, load and save sessions, as well as set watches and conditional breakpoints. Both debuggers also have the additional parallel debugging features for stepping through individual processors and viewing variable values across processors. DDT has an additional new feature called the Parallel Stack View which is discussed in section 2.2.

2.1 User Interfaces

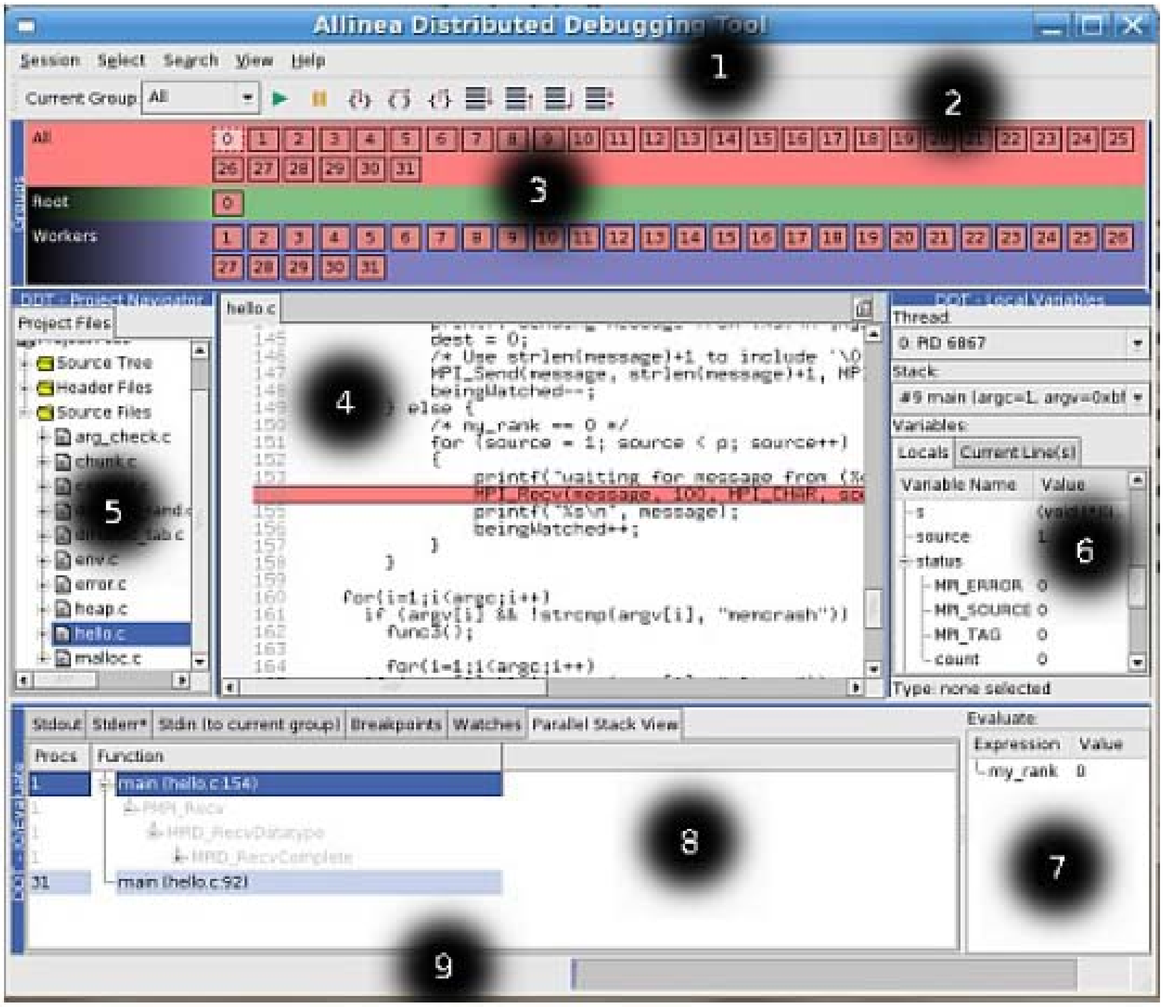
Although the interfaces to the two debuggers are quite similar, DDT's interface seemed more intuitive than Totalview's for a beginning user. DDT's main window (figure 1) includes a *code* window where all the source code and header files are listed, a main *project* window to step through the program, a *variable* window showing variable types, sizes and values, an *evaluate* window to dereference pointers or evaluate expressions, and an *output* window. Totalview on the other hand has a main window and then multiple smaller windows, one called the *root* window (figure 2) which shows processor information and a different window to evaluate expressions. It does not include a window where all the source code to an executable is listed.

An additional interface difference between the two debuggers is the manner in which a user navigates from one processor to the next. DDT has a box icon for each processor which is colored red if it has stopped and green if it is running. This makes it easy to tell the present state of each processor and if they are synchronized. Totalview has a few different ways to examine processor status. A user can press a +/- button to move to the next processor or a user can get an overview of each processor's state using the *root* window which displays if a process is running or if a process has stopped and in which routine it is located. The +/- button versus numbered processors icons is an irrelevant distinction. The important and useful feature is the ability to see what all processors are doing at once from a single window. DDT's answer and improvement to Totalview's *root* window processor overview is a new feature called the *Parallel Stack View* which shows the line of code where each processor is located, and groups the processors together by location. The *Parallel Stack View* is described in detail in section 2.2.

Neither the DDT or Totalview interface is ultimately ideal for scalability. In DDT, the space for numbered processor icons will run out and the +/- button feature in Totalview will quickly lose its appeal with hundreds of processors. In these cases, the only solution is to use Totalview's *root* window or DDT's *Parallel Stack View*.

When running a parallel job with Totalview, the first experience a user has with the interface is the pop-up message asking, *"Do you want to stop the job now?"*. Users are often confused about what is happening. No source code appears, and it isn't clear if the job has already started running or not. For a user to start stepping through the program from the beginning, he or she must choose to stop the job when this window first appears. DDT, on the other hand, automatically pulls up the source code and stops the program at the call to MPI_Init and then hands the control back to the user. In some cases, where DDT could not find the user's source code, the *project* window appears with basic instructions on how to find the source code and start the debugger by adding a breakpoint.

There are a few other minor interface differences. In the DDT *variable* window for example, variables are shown in alphabetical order, which can be a great help in scientific computing where there are often many variables. In addition, the user has the option of viewing all local variables or only variables on a specific line. In Totalview, the variable window distinguishes between routine arguments and local variables. DDT does not. Both Totalview and DDT allow the user to dive into multi-dimensional arrays, take slices and subarrays. There



```
        dest = 0;
        /* Use strlen(message)+1 to include '\0
        MPI_Send(message, strlen(message)+1, MP
        beingWatched--;
    } else {
        /* my_rank == 0 */
        for (source = 1; source < p; source++)
        {
            printf("waiting for message from (%
            MPI_Recv(message, 100, MPI_CHAR, so
            printf("%s\n", message);
            beingWatched++;
        }
    }

  for(i=1;i<argc;i++)
    if (argv[i] && !strcmp(argv[i], "memcrash"))
      func3();

      for(i=1;i<argc;i++)
```

Variable Name	Value
s	(void)(*))
source	1
status	
MPI_ERROR	0
MPI_SOURCE	0
MPI_TAG	0
count	0

Procs	Function
1	main (hello.c:154)
1	PMPI_Recv
1	MPID_RecvDatatype
1	MPID_RecvComplete
31	main (hello.c:92)

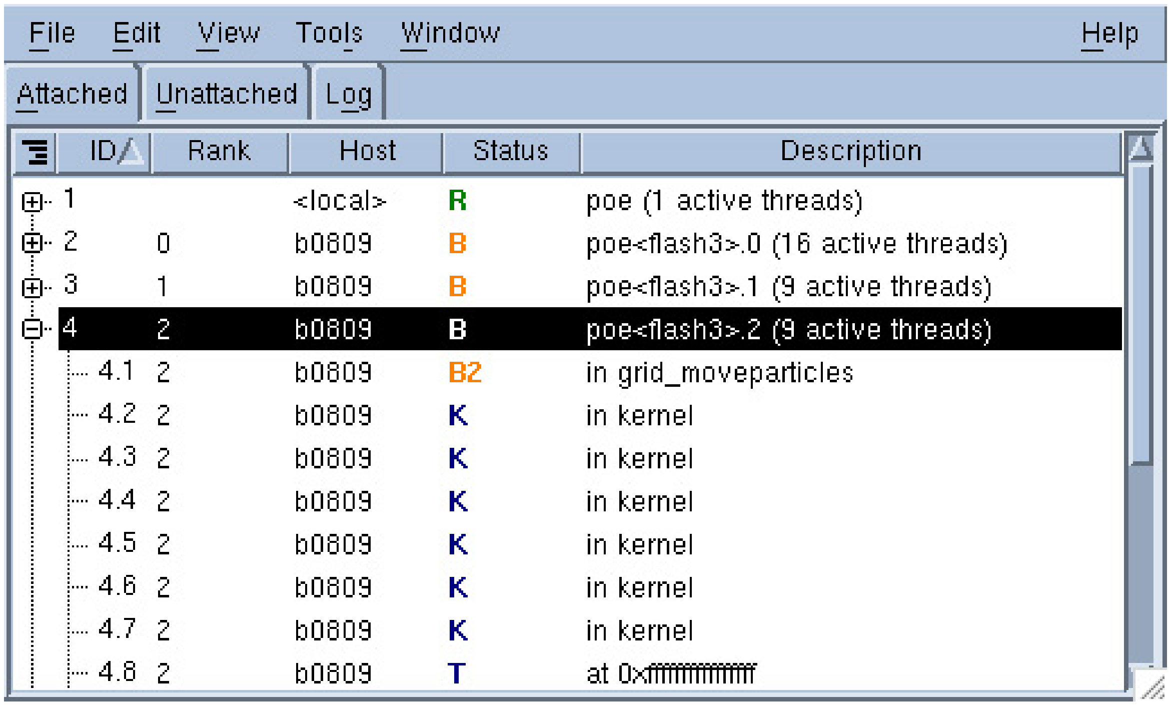
Expression	Value
my_rank	0

Figure 1. DDT Main Window (1) 1. Menu Bar 2. Process Controls 3. Process Group Window 4. Project Window 5. Code Window 6. Variable Window 7. Evaluate Window 8. Parallel Stack View and Ouput Window 9. Status Bar

is no ideal way to look at a greater than two dimensional array in a two dimensional space; however, DDT allows users to choose if they want to see data in rows or columns and allows users to output arrays to other formats. Both debuggers have the useful feature of being able to see maximum, minimum, infinity and NaN count values of a given array. Totalview can also run some more advanced statistics on array data .

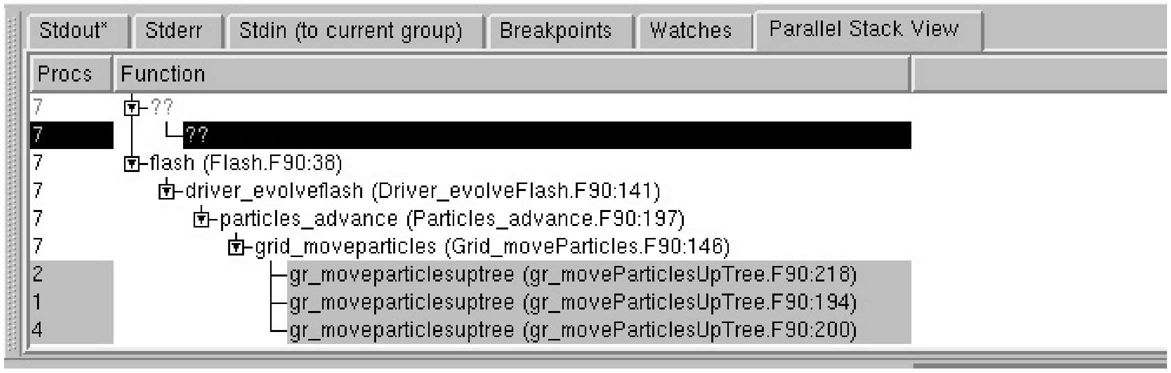
2.2 Parallel Stack View in DDT

By far the most useful new feature in DDT is called the *Parallel Stack View* which allows the programmer to see the position of all processors in a code at the same time from the main window. A program is displayed as a branching tree and the number and list of processors at each point in the tree, down to the line of source code level, is visible in the *Parallel Stack View* (figure 3). This DDT feature is an important addition to parallel debugging. Instead of clicking through each processor to determine where or if each has stopped, or scanning through a list of processors, the *Parallel Stack View* presents an overview of all the processors' locations. This feature should prove useful to developers trying to find logic bugs in code because it quickly and easily shows when a single or group of processors breaks away from others. The *Parallel Stack View* feature also logically leads to the grouping of processors by their position in a code. One of the greatest challenges of debugging



ID	Rank	Host	Status	Description
1		<local>	R	poe (1 active threads)
2	0	b0809	B	poe<flash3>.0 (16 active threads)
3	1	b0809	B	poe<flash3>.1 (9 active threads)
4	2	b0809	B	poe<flash3>.2 (9 active threads)
4.1	2	b0809	B2	in grid_moveparticles
4.2	2	b0809	K	in kernel
4.3	2	b0809	K	in kernel
4.4	2	b0809	K	in kernel
4.5	2	b0809	K	in kernel
4.6	2	b0809	K	in kernel
4.7	2	b0809	K	in kernel
4.8	2	b0809	T	at 0xffffffffffffffff

Figure 2. Totalview's Root Window (2)



```
Procs  Function
7      ??
7        ??
7      flash (Flash.F90:38)
7        driver_evolveflash (Driver_evolveFlash.F90:141)
7          particles_advance (Particles_advance.F90:197)
7            grid_moveparticles (Grid_moveParticles.F90:146)
2              gr_moveparticlesuptree (gr_moveParticlesUpTree.F90:218)
1              gr_moveparticlesuptree (gr_moveParticlesUpTree.F90:194)
4              gr_moveparticlesuptree (gr_moveParticlesUpTree.F90:200)
```

Figure 3. DDT's Parallel Stack View Window (1)

codes at high concurrencies is monitoring individual tasks. In DDT a user can create a sub-group of processors using the *Parallel Stack View* which then appear in the *processor group* window. A user can then step through the program looking only at the processors in this current group, simplifying the complex task of managing hundreds of processes. In Totalview, a user can also create custom groups, but the user must do this by hand rather than have it done automatically by processor location in the code. A further discussion of debugging at high concurrencies is found in section 3.4

3. Platform and Code Specifics

3.1 Fortran 90 Modules

Fortran90 modules prove to be tricky for both Totalview and DDT. The *use* feature of Fortran90 modules is not fully accessible to the debuggers, meaning from a subroutine a user can not dive into a Fortran90 module and view all data inside, as if it were any other local variable. Both Totalview and DDT allow the user to examine module data in a variable window when it is being used locally. Since Fortran90 modules are being used more and more often in scientific computing codes, the ability to view all Fortran90 module data at once is a helpful feature which both DDT and Totalview attempt to support. DDT has recently implemented a way to do this, although it is not supported by the underlying debuggers on all platforms. In the *code* window there is a *Fortran Modules* tab which will let the user see all module data, even if the code is not using data in that particular module at the moment. Totalview has *Fortran Modules* support on all platforms, and while the feature is very useful, it is also awkward because it is not connected to the main process window. When a user moves from one processor to the next in the main window, the *Fortran Modules* window is not updated for the new processor.

3.2 Experiences on AIX

AIX is one of DDT's latest supported operating systems, and Allinea has warned that not all features are supported yet. The installation process on both Bassi and Seaborg went smoothly. The user must select the appropriate MPI implementation, in this case IBM PE, and then configure the execution of jobs. Although the documentation could use some clarification, setting up DDT to run though the batch queue was fairly straight forward, and DDT's example batch scripts proved helpful. By default, DDT is setup to run with the system *mpirun* command, but simple modifications allow the program to be run with *mpiexec*, *poe*, other custom commands, or a queuing system. On AIX the user sets an environment variable DDTMPIRUN to the full path of *poe*. Each user can have his or her own configuration file, or DDT can be set up to point to a configuration file for the entire system.

For the Fortran90 codes, most of the basic debugging features worked from the start. Setting breakpoints and watches, diving into variables, stepping through the programs, evaluating expressions or loading and saving sessions all worked as advertised. User defined datatypes in Fortran90 are not yet well supported. A user defined linked list type showed some of the attributes but others appeared as *incomplete types*. Initially, there was a problem with multi-dimensional allocatable arrays showing the incorrect dimensions and an issue with corrupted data in pointer allocated arrays. A bug report and sample code were sent to the DDT developers and they quickly confirmed the bug and sent a fix back correcting the errors.

On AIX, the underlying debugger does not support the new *Fortran Modules* feature located in the *code* window in DDT, but module data can still be viewed from the *variable* window. In a few instances, locally used Fortran90 module data did not appear in the *locals* tab of the *variable* window. However, if the user clicks on a specific line of code with data from a Fortran90 module, that data always appeared in DDT's *current line* variable tab. This minor bug was reported to DDT. Additionally, in the FLASH code Fortran90 routines call C functions. DDT handled this interface between two routines of different languages without problems.

The C++ Chombo code did not fair as well as the Fortran90 codes. Allinea states that xlC C++ is not fully supported on AIX and this proved to be the case. DDT was not able to discern C++ classes. It viewed complex classes as incomplete types where as Totalview was able to dive into these classes and view the attributes. For example, a variable referenced by myClass.myVariable is distinguisable with Totalview but not with DDT. Allinea is aware of this limitation, but has thus far had few customers using xlC on AIX and so has not made fixing it a priority. DDT does support xlC for the Linux PowerPC and also fully supports the GNU compilers on AIX. The developers did not believe adding basic C++ class support for xlC would be very difficult.

DDT is set to release a new version of the software in a few weeks which will fully support OpenMP applications, allowing the user to step through individual and groups of threads. Currently, a user can run an OpenMP or mixed OpenMP/MPI program with DDT, however the debugger will skip over the threaded sections of the code. Totalview fully supports threaded applications on all platforms. The user can step through threads, set breakpoints, barriers, watches and perform most of the other features which can be performed on a processor.

DDT on AIX with the IBM compilers is quite functional, but still a bit buggy. Before fully recommending DDT on AIX, the upcoming release with multi-threading support should be tested and xlC C++ support should be added.

3.3 Experiences on Linux

Linux is DDT's strongest supported operating system, with full support for the *Fortran Modules* feature and C++ classes, however there have been some difficulties getting DDT running on the NERSC machine Jacquard which runs MVAPICH on Suse 9 with the Pathscale compilers. The software installation went smoothly and codes can be submitted to the batch queue. A simple parallel hello world, application runs fine on multiple processors but in more complex codes, the debugger was not handing control back to the user and in some cases crashed with segmentation faults. The DDT developers confirmed that DDT has successfully run with MVAPICH as well as with the Pathscale compilers and Suse 9. They have an account set up on Jacquard and are currently looking into the issue.

In order to test Linux support, DDT was installed on two other machines, Davinci, an SGI Altix machine running SGI's native MPI implementation with the Intel compilers and a dual processor desktop machine running Fedora 3 and MPICH, also with the Intel compilers. On both these machines, the new *Fortran Modules* feature is fully supported in DDT where Fortran90 data is accessible to the user even when a process is not currently in or using a particular module as explained in section 3.1. Additionally, most of the issues which turned up on AIX did not on Linux. Multi-dimensional arrays show the correct dimensions and C++ features are fully supported meaning the user can see all the attributes and variables in a C++ class, however, user defined Fortran90 datatypes are not well supported. A user can not always see all the attributes of a complex user defined datatype. Additionally, as on AIX, DDT was able to smoothly handle the interface between Fortran and C calls. Support for multi-threaded applications is not yet available, but is planned in the upcoming release.

3.4 Debugging Large Concurrency Applications

Debugging at high concurrencies is difficult with or without the aid of a debugger. Even when using a parallel debugger like Totalview or DDT, the sheer number of processors, variables, stacks and screens a user must manage is daunting at any level above about 32 processors. Fortunately, with many parallel programs, a bug which occurs at 256 processors will often show the same error at smaller concurrencies making debugging easier. For those applications where a programming bug only occurs at high concurrencies, the developer is faced with limited options. While both Totalview and DDT officially run at 64, 128, 256 and higher processor counts, both are understandably slow and awkward. DDT and Totalview were tested at 64 and 128 processors and to step through a program from one line to the next took about 20 seconds for each in completely synchronized parts of the code. DDT and Totalview each stopped responding a few times when attempting to run in more complicated sections of the code where processors executed different parts of the code. In each case, some processors never handed control back to the debugger. Although there may be no other good option, only the most desperate developer will likely tolerate these latencies. NERSC and other centers could limit the number of large processor licenses as these are likely rarely used.

There is no easy scalable way to display 256 processors on a screen and DDT and Totalview attempt to handle this case differently. Very quickly Totalview's +/- button to click through processors becomes unmanageable and DDT's processor icons will start to dominate the screen. The best case in Totalview is to use the root window to see where each processor is located and move from one processor to the next, however it could get tedious scanning through hundreds of processors. The biggest advantage DDT has over Totalview in debugging at large concurrencies is the Parallel Stack View feature described in section 2.2. The user can create groups of processors, based on process position in the code and from a single screen can see where each processor is located in the code. This feature will be useful when debugging a program running with a large number of processors, but it will not help the user step through the program any faster.

3.5 Remote Access

Since most of the NERSC users will be running either Totalview or DDT remotely it was important to compare the X window delay when running from outside of NERSC. Both Totalview and DDT performed about the same for remote access. There was a small delay for both, but nothing different than any other remote X window application.

4. DDT Limitations

While DDT has a few new features which should prove useful to developers of scientific codes, there are still holes in DDT's support for various platforms and functionalities. DDT does not currently support the Cray XT series machines. The developers they are working on this capability, however no time frame has yet been given. And as mentioned above, DDT has limited support for the xlC C++ compiler on the AIX platform. Finally, DDT is soon to, but has not yet released full support for threaded applications. The next release is due out in a few weeks.

5. Interaction with DDT developers

Interaction with the DDT developers has been responsive and timely. When the multi-dimensional allocatable array bug was found on AIX, the DDT developers communicated frequently with NERSC and delivered a bug fix in a matter of days. Should NERSC or any other center choose to purchase DDT, it would be necessary for this same high quality interaction to continue as a support model to verify that all bugs are fixed as more codes use DDT and new platforms become supported.

6. Conclusions

Allinea's parallel debugger, DDT is a viable alternative to Totalview. Along with matching Totalview's key features, DDT has improved a few areas such as the *Parallel Stack View* and a simple user friendly interface. Compared to Totalview, the greatest drawback to DDT is its lack of maturity as a software application. DDT runs on fewer platforms, and feature support is not uniform across the supported platforms. NERSC's new largest machine Franklin, a Cray XT, is not yet a supported DDT platform and there are still major deficiencies in xlC C++ support on AIX. Additionally, since DDT does not have as many users as Totalview, fewer codes have been tested and thus more bugs are likely to be found. With that said, the DDT developers were eager to help and explored, confirmed and fixed bugs quickly. If the community is prepared to help report bugs found in the DDT code as new platform support becomes available, they will find a powerful parallel debugging alternative to Totalview.

References

[1] DDT Software and User's Guide, Allinea Software, February 2007, http://www.allinea.com.

[2] Totalview Software and User's Guide, Etnus Corporation, February 2007, http://www.etnus.com/Documentation.

[3] Chombo Infrastructure for Adaptive Mesh Refinement, Applied Numerical Algorithms Group, Lawrence Berkeley National Laboratory, http://seesar.lbl.gov/ANAG/chombo.

[4] The FLASH Code, DOE-supported ASC/ Alliance Center for Astrophysical Thermonuclear Flashes, The University of Chicago, http://flash.uchicago.edu.

[5] Gyrokinetic Toroidal Code (GTC), Princeton Plasma Physics Laboratory, http://w3.pppl.gov/theory/GPSC.html.