DsTool 2.0

Programming Changes From Version 1.1

Author: Patrick A. Worfolk
Center for Applied Mathematics
Cornell University
657 Engineering and Theory Center
Ithaca, New York 14853
paw@cam.cornell.edu

Date: July 1993 Draft - Updated June 1995

1 Overview

The main purpose of these changes was to install a more flexible text based postmaster. This facilitated the installation of a text control language (Tcl) and the parallelization of several algorithms using PVM. The main changes we discuss are listed here:

- Text based postmaster
- Separation of window code
- Command line arguments
- Tcl control language
- Save/Load format
- PVM parallel algorithms
- Installation of system and user panels
- Window structs
- Extensions to Prop_DataS
- Mouse interaction
- Interrupts
- New user features
- Portability and Makefiles

We also make suggestions for future changes and improvements which have not yet been made.

2 Text based postmaster

The postmaster is now text based. This means that all objects and elements are now referenced with character string labels as opposed to integer constants. This allows postmaster objects and elements to be created at run time. See the accompanying document for the details of the new postmaster.

Here we discuss the changes made to the code to accommodate the new postmaster that go beyond the change of each individual postmaster call which was changed to use the new string identifiers. This is essentially a discussion of the standard installation of postmaster objects and elements.

Two routines have been written for each postmaster object. For example we will look at the "Selected" object. The first routine is called `selected_install` and creates the "Selected" object and all its elements. This routine is called once when DsTool is first initialized. The second routine is called `selected_reset`. This routine is called after the installation routine and again each time the dynamical system model is changed. These two routines are entered into structs in the files `sys_install.c` and `user_install.c` depending on whether or not they are critical to the core of DsTool.

See the files `sys_install.c` and `user_install.c` in `$DSTOOL/src/main`. Panels not guaranteed to be compiled into user DsTool's generally call their postmaster initialization and reset procedures from their open and field_manager routines, respectively.

3 Separation of window code

We have completely separated the windowing code from the computational code. This allows us to build a version of DsTool without having the XView libraries available. This is useful for batch mode and for running as a PVM slave. It also produces substantially smaller executables which may be desirable for these modes even when the XView libraries are available. Changes necessary to do this were minor and often resulted simply in the reorganization of a few procedures. There are a few "stub" routines necessary for an easy implementation of creating a "no_win" version. These are found in the file `stubs.c` in `$DSTOOL/src/main`. This code is compiled into a different main library called `mainlib_nowin.a` using the makefile `Makefile_nowin`. This library and executable can be created from the makefile in `$DSTOOL/src` with the command `make dstool_nowin`. The resulting executable is entitled `dstool_nowin`.

4 Command line arguments

The command line arguments have been changed. The new usage line is given by:

```
dstool [-windows] [-tcl] [-pvm] [-version] [-debug] [-help]
       [-infile <filename>] [-outfile <filename>]
```

-windows Puts DsTool in windows mode. This is the default.

-tcl Puts DsTool in Tcl mode. This is discussed in the following section on the Tcl command language.

-pvm Tells DsTool that it is a pvm host. This enables parallel algorithms and requires that the pvm daemon be up and running. This is discussed further in the section on pvm parallel algorithms.

-version Prints the version of DsTool and forces an exit with error code 1.

-debug Turns on debug mode within DsTool. This currently sets a postmaster item "Control.Debug" to TRUE but has no other effect.

-help Prints out the usage string given above and exits with error code 1.

-infile filename Sets the given file to be an input file. The input file is not currently used, but the code should be modified to automatically read this file as a configuration file.

-outfile filename Sets the given file to be an output file. This is currently used only when in Tcl mode.

5 Tcl control language

"Tcl" is a text control language [2] that may be used to control DsTool. We use it both to write custom algorithms as well as to serve as a batch language. The current mode of use is to create a text file using any editor and then to run it through DsTool and the Tcl interpreter. This is done in a nonwindowing mode by specifying the `-tcl` command line argument flag or optionally in windowing mode via the Tcl window opened from the panels pull down menu.

When Tcl is run in batch mode the output file is used to write the contents of the memory objects to when execution of the Tcl file is complete. The input is through stdin, so commands may be typed at the console or may come from a file. For example, it may be used in the following fashion:

```
dstool -tcl -outfile tcl_out < tcl_in
```

Here DsTool will execute the Tcl commands in the file `tcl_in` and on completion write the memory objects to the file `tcl_out`.

In no_windows mode, results of Tcl commands are printed to stderr. If the postmaster field "Tcl.Echo" (type integer) is set non-zero, then input strings are also echoed; this is useful in batch mode, or when the Tcl window runs in verbose mode.

The postmaster field "Tcl.Verbose" (type integer) controls whether individual command results are printed during Tcl window evaluation of a file. When set to 0, only the final result is printed.

5.1 Tcl commands

All standard Tcl commands are available to be used in Tcl files. In addition there are a set of commands which interface to DsTool. These essentially allow the execution of postmaster commands thus allowing a Tcl file to control almost all of the computations within DsTool. The available commands are:

pm [EXEC — PUT — GET] ¡object.element¿ [data ...] This command is used to interface with the postmaster and is the same as the DsTool 2.0 load format.

[SET] ¡name¿ [data ...] This command translates to a Tcl set. When name is "new_object" and data a list of list of lists, this is the DsTool 2.0 format for loading a memory object. While the load routines utilize this format for loading memory objects, the Tcl facilities do not as yet.

The implementation of the Tcl interface is not yet complete. Ideally the format of the commands should be the same as the input format so that files can be loaded through Tcl, if desirable. This will avoid multiple control languages.

6 Save/Load format

The format of DsTool files has been changed. This is essentially due to the nature of the new postmaster. The postmaster can now write a file which reflects much of its status. This file is written in tcl format and may be executed by tcl in order to reset the postmaster as much as possible to the state it was in when the file was saved. This file can be read directly by DsTool or passed through the tcl interpreter. Additional tcl commands may be embedded in this file, but then it must be read through the tcl interpreter.

The "Win_Names" postmaster category contains a list of *shortnames* of panels which are installed. The shortname of a window is the first word in its title, as specified in the menu list of user_panels.c. This shortname is used by the saveload routines to locate window configuration information within the "Win" postmaster category.

Because of the dependence on shortnames, it is important that user DsTool configurations not change (the first word in) the menu entry for each user panel.

7 PVM parallel algorithms

DsTool can run in parallel using the PVM message passing library [1]. Currently the only algorithm which is written to take advantage of this is the Multiple panel. The pvm daemons should be started in the usual way before running the DsTool program attempts to access them. The dstool program must be informed that the pvm daemons are up and running. This is done by passing in a `-pvm` on the command line on program startup (if pvmd is already active) or by executing a "Pvm.Start" command in the postmaster (via tcl, for example) once pvmd has been made active. This will then start a pvm slave DsTool program on each processor registered with pvm. The slave program is called `dstool_pvm` and must be in the user's directory $HOME/pvm/$`ARCH` as documented in the PVM User's Manual [1]. This program should either be a copy of dstool or a nonwindowing version of dstool. Alternatively, it may be a softlink to one of the programs. The executable will AUTOMATICALLY start up in a pvm slave mode when its name contains the string "pvm".

Installation of pvm requires including $(DSTOOL)/src/pvm/pvmlib.a in the PRE_LIBS library of $DSTOOL/src/Makefile as well as adding "-lpvm" to the libraries linked in at the end. Because of the preliminary state of the current pvm work, pvm is not currently linked in during default installation of DsTool.

7.1 Multiple trajectories

When a multiple trajectory computation is started in the usual manner, the code checks to see whether there are PVM slaves. If there are then the computation of trajectories is split evenly between the slave processes. Each slave process is given a set of initial conditions and we wait for all of them to finish and return. We do not take advantage of possible different rates of computation between the processors nor possible different quantities of computation for different initial conditions. Depending on the quantity of trajectories needed to be computed and the number of processors, it might be a better strategy to send dispatch the computations in a greater number of smaller batches as each processor is ready for more work. We note that currently the user has no control over how the computation is split up. In fact, the parallel interface is transparent to the user with the acknowledgement that he must have started the pvmd daemons and told DsTool to use them as it sees fit.

This implementation is NOT complete. Currently, only the mult forwards computation takes advantage of the virtual parallel machines. Also, the "Copy Final to Initial" button on the Mult

panel will not correctly update the initial points PVM is being used. Additionally, the Continue feature does not work correctly in PVM mode.

8 Installation of system and user panels

See the files `sys_panels.c` and `user_panels.c` in `$DSTOOL/src/windows`. These need to be further isolated for the ease of custom user panel installation. We still have not addressed the issue of the simplest way for a user to add a new panel.

9 Window structs

The `ui_windows` struct has been eliminated. The instance pointers for each window are now local or global depending on whether they are needed beyond the data_refresh and read_window routines. The `ui_init.h` file now simply contains the declaration to the instance pointer for the Command Window. This is required since it needs to be the owner of all the new windows created.

10 Extensions to Prop_DataS

The `Prop_DataS` structure has been extended to include a pointer to the auxiliary functions and a pointer to the function which plots trajectory data. These function pointers are both grabbed from the postmaster on initialization of the structure. The auxiliary function is used when colorcoding the plotted trajectories so that the a postmaster call is not made for each point plotted. The addition of new plot trajectory procedures allows more control of how data is "plotted" as it is computed. In particular, this is used in our parallel algorithms to send data back to the master program which displays the data on the user's windows and stores it in local memory.

11 Mouse interaction

The functions which are called when a mouse button is pressed have been isolated and moved into a file by themselves. This facilitates the customization of mouse commands. These procedures could be moved into the postmaster so that they could be changed at run time.

See the file `mouse.c` in `$DSTOOL/src/twoD`.

12 Interrupts

The interrupt mechanism has been removed from the postmaster. This is to improve efficiency so that interrupt checking may be done frequently in computational algorithms without a great loss in speed. The following routines should be used for setting, resetting and querying the interrupt status:

```
int set_interrupt();
int reset_interrupt();
int interrupt();
```

The procedures take no arguments and all return the final status of the interrupt flag. Consequently, `set_interrupt()` always returns TRUE and `reset_interrupt()` always returns FALSE. All calls to `prop_halt()` have been replaced by a call to `interrupt()`.

See the file `interrupt.c` in `$DSTOOL/src/utilities`.

13 New user features

The only visible new feature is the activation of the Record Data option on the Defaults Settings Panel. This field is examined by the code which computes orbits (regular and multiple). When set to "NO", trajectory data is not stored in the memory objects. This means that the data cannot be refreshed to a view window, nor can it be stored to a file or printed via the DsTool window.

14 Portability and Makefiles

We have attempted to eliminate the need for two versions of DsTool. Previously one worked for Suns and the other was more general. The main change is that the tree of source files has been flattened and more Makefiles have been changed. Note that there are now multiple `targets*.mk` files and new `win_targets*.mk` files in `$DSTOOL/site_specific`. This is because many make programs do not like empty targets. The new files prefixed with `win_` are for use only in the windowing code libraries. This enables a version of DsTool to be compiled that does not use the windowing system which is useful for running in both batch mode and a slave PVM mode. DsTool can be compiled for these limited uses on machines for which the XView libraries are not available.

Additionally the Sun specific math functions `nint` and `iszero` are no longer used in the code.

In order to compile the code you must do the following:

1. Set the `DSTOOL` environment variable.
2. Edit the file `site_specific/lib_incl.mk` file to set the `ARCH` variable to `SUN4` or `RIOS`.
3. Make the programs `dstool` and `dstool_nowin`

References

[1] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and V. Sunderam. A users' guide to PVM parallel virtual machine. Technical Report ORNL/TM-11826, Oak Rdge National Laboratory, 1991.

[2] J. Ousterhout. *An Introduction to Tcl and Tk.* Addison Wesley, 1993. Draft.