ORGANIZATION AND USAGE

The README file distributed with the package describes the physical organization of the package into files, and includes the basic instructions for compiling, testing, and running the installed serial and parallel codes. This document describes the organization and usage of the key modules, driver subroutines, and test programs.

Package Organization

Figure 5.1 shows the high level organization of VTDIRECT95. The module `VTdirect_MOD` declares the user called driver subroutine `VTdirect` for the serial code. Correspondingly, the module `pVTdirect_MOD` declares the user called parallel driver subroutine `pVTdirect`, the subroutine `pVTdirect_init` for MPI initialization, the subroutine `pVTdirect_finalize` for MPI finalization, as well as the data types, parameters, and auxiliary functions used exclusively in the parallel code.

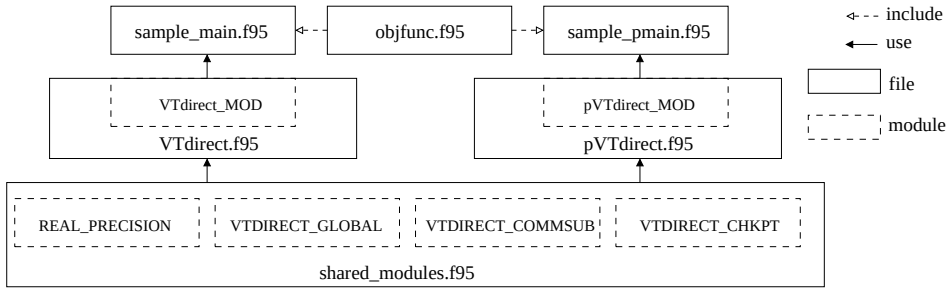

Fig. 5.1. The module/file dependency map.

The two driver subroutines `VTdirect` and `pVTdirect` share the modules: (1) `REAL_PRECISION` from HOMPACK90 (Watson et al. [1997]) for specifying the real data type, (2) `VTDIRECT_GLOBAL` containing definitions of derived data types, parameters, and module procedures, (3) `VTDIRECT_COMMSUB` containing the subroutines and functions common to both the serial and parallel versions, and (4) `VTDIRECT_CHKPT` defining data types and module procedures for the checkpointing feature. These shared modules are merged in the file `shared_modules.f95` as shown in Figure 5.1. `sample_main` and `sample_pmain` are sample main programs that call `VTdirect` and `pVTdirect`, respectively, to optimize five test objective functions defined in `objfunc.f95` and verify the installation. The dependencies between the package components are depicted in Figure 5.1.

In the sample serial main program `sample_main` each test objective function illustrates a different way of calling the driver subroutine `VTdirect`. The calls illustrate the four different stopping rules—maximum number of iterations `MAX_ITER`, maximum number of function evaluations `MAX_EVL`, minimum box diameter `MIN_DIA`, and minimum relative decrease in objective function value `OBJ_CONV`. For the last objective function, a multiple best box (MBB) output is illustrated. Details of the arguments are in comments at the beginning of the subroutine `VTdirect`. Different parallel schemes are used in the test cases for `pVTdirect`, called by the sample parallel main program `sample_pmain`. Both sample main programs print to standard

out the stopping rule satisfied, the minimum objective function value, the minimum box diameter, and the number of iterations, function evaluations, and the minimum vector(s). In addition, the test output for pVTdirect lists the number of masters per subdomain and the number of subdomains.

Different computation precision and different compiled code on different systems may require different numbers of iterations or evaluations to reach the desired solution accuracy (1.0E-03) specified in the test programs. If a test program fails to locate the optimum value or the optimum point given the stopping conditions in the supplied namelist input file, the stopping conditions can be adjusted accordingly.

Using VTDIRECT95

One of the virtues of DIRECT, shared by VTDIRECT95, is that it only has one tuning parameter (ϵ) beyond the problem definition and stopping condition. Using VTDIRECT95 basically takes three simple steps. First, define the objective function with an input argument for the point coordinates (`c`), an output argument for evaluation status (`iflag`), and an output variable for the returned function value (`f`). A nonzero return value for `iflag` is used to indicate that `c` is infeasible or `f` is undefined at `c`. The user written objective function is a FUNCTION procedure that must conform to the interface:

```
INTERFACE
  FUNCTION Obj_Func(c, iflag) RESULT(f)
    USE REAL_PRECISION, ONLY: R8
    REAL(KIND = R8), DIMENSION(:), INTENT(IN)::  c
    INTEGER, INTENT(OUT)::  iflag
    REAL(KIND = R8)::  f
  END FUNCTION Obj_Func
END INTERFACE
```

Second, allocate the required arrays and specify appropriate input parameters to call one of the driver subroutines. In the parallel case, the MPI initialization and finalization subroutines need to be called before and after calling the parallel driver subroutine (pVTdirect), unless MPI is initialized and finalized elsewhere in the same application. The required arrays include the input lower (`L`) and upper (`U`) bounds, and an output array for the optimum vector (`X`). Additionally, in the parallel version, the return status is also an array, required to be allocated beforehand, to hold statuses returned from subdomains, even if only one domain exists, in which case the size of the status array is one. If the user desires to specify the optional input argument `BOX_SET`, an array of boxes must be allocated and an optional weight array `W` for dimensional scaling may also be allocated.

All other input parameters specified in the argument list of the driver subroutine are conveniently read in from a NAMELIST file, as illustrated in the sample main programs. Using namelist files is an elegant way of varying input parameters as needed, without recompiling the program. The namelist file `pdirectRO.nml` shown below is to test `pVTdirect` for optimizing the 4-dimensional problem RO. The parameters are grouped into four categories (NAMELISTs): parallel scheme

PSCHEME, problem configuration PROBLEM, optimization parameters OPTPARM, and checkpointing option CHKPTOP. This example uses two subdomains and two masters per subdomain, and the stopping condition is when the minimum box diameter reaches 1.0E-05. The checkpointing feature is activated when chkpt_start equals 1 (saving) or 2 (recovery). The program will terminate if the checkpoint file errors occur as explained in the section on error handling (cf. Section 3.4). It is the user's responsibility to maintain the checkpoint files, including renaming or removing old files.

```
&PSCHEME n_subdomains=2 n_masters=2 bin=1 /

&PROBLEM N=4
    LB(1:4)=-2.048,-2.048,-2.048,-2.048
    UB(1:4)=2.048,2.048,2.048,2.048 /

&OPTPARM iter_lim=0 eval_lim=0 diam_lim=1.0E-5 objf_conv=0.0
    eps_fmin=0.0 c_switch=1 min_sep=0.0 weight(1:4)=1,1,1,1
    n_optbox=1 /

&CHKPTOP chkpt_start=0 /
```

Finally, the last step is to interpret the return status, collect the results, and deallocate the arrays as needed. The return status consists of two digits. The tens digit indicates the general status: 0 for a successful run, 1 for an input parameter error, 2 for a memory allocation error or failure, and 3 for a checkpoint file error. The stopping condition for a successful run is further indicated in the units digit, which also points to the exact source of error if a nonzero status is returned. For example, a return status of 33 means the checkpoint file header does not match with the current setting. All the error codes and interpretations can be found in the source code documentation. A successful run returns the optimum value and vector(s) in the user-prepared variables and arrays. In order to receive a report on the actual number of iterations/evaluations, or minimum box diameter, these optional arguments must be present in the argument list. The final results of calling pVTdirect are merged on processor 0 (the root master), so proc_id is returned to designate the root to report the results. VTDIRECT95 is designed so that the optimization results may be directly fed to another procedure or process, the typical situation in large scale scientific computing.