Developing Guide

INTRO

This short text is intended to be a help for developing new functions within the GIFA project, as well as a help for porting GIFA to new platforms.

GIFA has been ported today (april-1998) to :

-Silicon Graphics / IRIX 4.x 5.x

-IBM Risc 6000 / Aix

-HP 700 running HP-UX9.x 10.x

-PC /Linux

-Mac /Mklinux

-f2c fortran compiler on a generic sysV machine

-SUN/Solaris

obsolete versions, used to work :

-Macintosh running MachTen 2.1.1

-Spec 1

-Vax/VMS VWS

-Vax/VMS DecWindows

-Alliant

-DEC DS 5000

FILE SETUP

The source is on the ftp serveur, in an encrypted format. It is available to the fully licensed users of the program.

-all the source files for GIFA are called *.for and *.inc (for fortran files) or *.c and *.h (guess for which files!), and should be located in a single directory, usually gifav4/source

-binary files are located in specific directories, under source, named according to the architecture. These directories are absent in the distribution, but a copy is present at the same level as source (for instance sgi_obj, hp_obj, etc..) These copies contain the important specific files (usually makefile).

-there is a Makefile for GIFA that should help. There are several entries in this makefile that you may wish to examine.

-Gifa is compiled in static sizes (at least some parts), 4 sizes are preset. Small shell scripts : `tiny, small, norm, big` are used to switch the source from one size to the other. They copy some reference include files to active ones, so the recompilation is quite lengthy.

-the fortran compiler has to know several fortran extension :

*support the 'implicit none' command, that forces the declaration of the type of all the variables (switches off the implicit typing of fortran and generate error messages during compilation for undeclared variables)

*support the ! sign for in-line comments

*support the special syntaxes for DO-LOOP :

```
    DO I=1,N
          :
    ENDDO
```

standing for

```
    DO 100 I=1,N
          :    ! in standard fortran 77
100       CONTINUE
```

* and in some few cases

```
    DO WHILE(logical test)
          :
    ENDDO
```

standing for

```
100       IF (logical test) THEN
          :
          GOTO 100
    ENDIF
```

If your compiler does not handle those structures, you should either seek for another compiler (try f2c or g77), or try to modify the pre-processor.

MAKEFILE SETUP

The main makefile : source/Makefile is meant to be the main entry, from which all operation are realised. There is an additional makefile in each binary specific directory (for instance source/hp_obj/makefile) that holds all the specific stuff : actually the rules for compiling C and FORTRAN. The process handle the fact that the source files and the *.o are located in different directories by using the $VPATH capability of make. The dependence list for the source files is source/depend_list. It is included in the machine-specfific makefile with the include command.
Make determines on which machine it is running with the $ARCHI macro, that you might have to adapt.

The following entries are available :

make gifa : compiles gifa for the current architecture, using all current definitions, optimise

make debug: compiles gifa for the current architecture, using all current definitions, does not optimise but rather add debugging code, then start Gifa with debugger.

make depend:	builds the dependence list. Uses a perl script (makedepend), and a starting file (depend_list.in)
make all_version:	Compiles Gifa in all 4 static sizes : tiny, small, norm, big. Quite lengthy ! To be used before a binary distribution.
make clean:	removes all temporary and *.o files for the current architecture
make all_clean :	removes all temporary and *.o files for all architectures
make install:	installs gifa on this machine

make kit_bin:	makes a distribution kit for the binaries of the current machine
make kit_base:	makes a distribution kit for generic files
make kit_data:	makes a distribution kit for demo data files
make backup:	builds a gifa.tgz backup file in the $(RES_DIR) definition

make run:	runs gifa for the current architecture, (which is in the $obj dircetory)
make show_doc:	builds and shows the programmer documentation in html format
make test:	runs tests on gifa

FORTRAN PREPROCESSOR

The fortran sources files are intended to be preprocessed before compilation. The preprocessor is a perl program called fpp.

*the lines in processed files that need preprocessing hold a fpp entry at the begin of the line of the form :
#xxx where xxx determine which action should be taken, either coding parts specific to a operating system, or including a files.
*the following action are possible :
#include "file_name" file_name is included in source. #includes may be nested

#hp	this line is specific to HP machines
#sgi	this line is specific to SGI machines
#sun	this line is specific to SUN machines
#adec	this line is specific to DEC/Unix machines
#aix	this line is specific to IBM/Aix machines
#f2c	this line is specific to the f2c compiler
#vms	this line is specific to vms machines
#unix	this line is common to all unix machines
#_xxx	this line is left in all cases but xxx (e.g. #_hp)
#d	this lis is left only if a -D (debug) is passed to fpp

#assert (fortran condition) if in debug mode (-D) a line is inserted that will issue a message to the user whenever (fortran condition) is false. The program will condition.

Typical uses are :

```
#aix      line specific to aix machine
#_aix     line generic to all other version
#include  "file_to_include"
#vms#include   "include_file_typical_to_vms"
```

C TO FORTRAN INTERFACE

-Since C and FORTRAN are mixed in Gifa, it is a requisite to be able to call C code from FORTRAN.

On most machine, C is called from fortran with the following rules

```
    INTEGER I
    REAL X
    CALL FOO(I,X)                       in fortran

    void foo_(int *I, float *X);        in C.
```

note the underscore in the C function name.

On the IBM Risc 6000/AIX and HP 700/HP-UX machines, the underscore should not be present, a label UNDERSCORE should be defined accordingly in the compilation of the C routines.

SOURCE SETUP

The main entry point for GIFA is the file gifa.for, which holds the declaration and the initialization of all the static variables and the main event loops (command driven or event driven). The command parser is located in the execute.for file.

All static variables held in COMMON blocks are declared in include files, such that no common miss alignment may occur. The main data structures are declared in SIZEPARAM.INC which includes a size definition files SIZEBASE.INC. In this latter file a parameter SMXMAX is declared which is the size of a chunk of memory. 8 such chunks are declared and aligned. Thus GIFA may processed classically data-sets up to 8xSMXMAX (actually called SMXBIG) long; or may use maximum entropy, linear prediction, etc... on up to SMXMAX big data-sets. A parameter SIZEMAX determines the size of the largest 1D possible when processing a 2D data-set. You will find several examples of SIZEBASE.INC : (sizebase.norm (for 4meg buffers) sizebase.big (for 16meg buffers) and sizebase.small (for 1meg buffers)). The data buffers are then available from each routines that will include this file. See examples from other source files. As a general idea, the 2D data-sets are always contiguous in memory, and can always be considered as a large 1D buffer if needed. When processing 2D row by row, each row can be accessed directly, when processing in column, the 2 subroutines GETCOL and PUTCOL should be used to access each column. See tdftcpx.for for instance.

All the mathematics involving 1D vectors are performed through calls to the special routines xxxvect() gathered in the file vect.for. This routines are compiled such as to be called concurrently on multi-processors machines, so you are urged to call these routines in order to keep the program running in a optimized fashion on different platforms. (actually this set-up is a bit old, and no parallele support is there anymore, however that might come one day...(PVM is planned))

All the user I/O are performed through a set of special routines, which permits to use the GIFA environment. For getting input form the users, use GETINT2, GETREAL2, GETSTRING2. These routines will be able to catch input from the command line, variables, or to prompt the user if nothing is present on the command line. For writing to the user, use either GIFAOUT(string) GIFAOUTI(string,integer) GIFAOUTR(string,real) for unconditional output, or MESSAGE() to prompt for a value (in this case the string will be issued only if the command line is empty). Both routines will also copy the output to the log file. GIFAERR(string) is used to issue an error message.

The file unix_lib.for (vms_lib.for) holds code for forking a unix function or a shell (csh), waiting a few milliseconds, checking cpu and elapsed time, and all this kind of O.S. specific stuff. You may have to fiddle around with it.

The cache system (in C) uses a large static buffer to hold the cache memory. The size (and structure) of this buffer is determine in a the file config.h Three such files are provided (config.small config.norm config.big) which you should choose depending on your system.

DEVELOPING NEW FUNCTIONS

Developing a new functionality in GIFA consists in adding commands to the parser in execute.for. Each command will then either branch to routines for specific actions, or will set values of internal variables. Errors are handled through the branching to a set of labels at the end of execute.for; therefore, all called subroutines should back-propagate errors to the main loop in order to handle them properly.

You may consider using include files for all your code, in order to modify as less as possible the main source files. In this case, you might use 3 include files : XXXV in the declaration zone, XXXI in the initialisation zone, XXXP in the parser zone. See how linear prediction is inserted for instance.

You may want to create new internal variables available from Gifa, in which case add your variable in variable.for:getvar()

I think that's about all, and should not be an enormous work. If you manage to port GIFA to a new machine or set-up a usefull functionality, please let me know, in order for me to make that new version available to other users.

Good luck!

Marc-André DELSUC

Marc-André Delsuc | Centre de Biochimie Structurale
mad@cbs.univ-montp1.fr | Faculte de Pharmacie
tel : (33) 67 04 34 36 | 34060 Montpellier cedex
fax : (33) 67 52 96 23 | FRANCE