

# Portable Application Compilation and Building for Fortran 90.

R.J. Allan and Y.F. Hu  
Computational Science and Engineering Department,  
CLRC Daresbury Laboratory,  
Daresbury, Warrington WA4 4AD, UK

January 18, 2002

## Abstract

In this report we outline several procedures which have been developed at Daresbury Laboratory to build application software written in Fortran 90. In particular we provide examples of how to construct portable makefiles for programs which have modules in several source files or directories. We developed this procedure for porting the CLIPS library.

Some automatic tools, such as `makemake`, for producing makefiles from source code are described.

We also include illustrations of the choice of compiler flags which have been used on a variety of platforms and comments on the use of `cpp` for target- and language-specific source lines.

**Keywords:** Fortran 90, compilation, make, makemake, cpp, portable software, application building, parallel computing, CLIPS library.

**This is a Technical Report of the UKHEC Collaboration.**

Report available from <http://www.ukhec.ac.uk/publications>

© **UKHEC 2001.** Neither the UKHEC Collaboration nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Fortran 90 . . . . .	1
1.2	Portable Parallel Applications . . . . .	2
1.3	The CLIPS Library . . . . .	2
1.4	Computers Considered . . . . .	3
<b>2</b>	<b>Makefiles</b>	<b>3</b>
2.1	Layout of the prototype <b>makefile</b> . . . . .	3
2.2	Default Rules for makefiles . . . . .	4
2.3	Fortran File Names . . . . .	5
2.4	Environment Variables . . . . .	5
2.5	Tools and Scripts for makefiles . . . . .	7
2.5.1	GNU Make . . . . .	7
2.5.2	makemake . . . . .	7
2.5.3	fmkmf and mkmf . . . . .	7
2.5.4	fimake . . . . .	8
2.5.5	sfmakedepend . . . . .	8
2.6	Use of <b>cpp</b> for Conditional Compilation . . . . .	8
<b>3</b>	<b>Compiler Options</b>	<b>10</b>
3.1	Absoft Fortran v1.01 on Linux Pentium Platforms . . . . .	10
3.2	Compaq . . . . .	10
3.3	Cray . . . . .	11
3.4	Fujitsu . . . . .	11
3.5	HP . . . . .	11
3.6	Hitachi . . . . .	11

3.7	IBM Fortran v6.1 and v7.1 . . . . .	12
3.8	Pacific Sierra Research . . . . .	12
3.9	Portland Group Compiler on Linux Platforms . . . . .	12
3.10	SGI . . . . .	13
3.11	SUN . . . . .	13
<b>4</b>	<b>Module Information Files</b>	<b>13</b>
4.1	Absoft . . . . .	14
4.2	Compaq . . . . .	14
4.3	Cray T Series . . . . .	14
4.4	Hitachi . . . . .	14
4.5	IBM . . . . .	14
4.6	Portland Group . . . . .	14
4.7	SGI . . . . .	15
<b>5</b>	<b>Examples of Makefiles</b>	<b>15</b>
5.1	Recursive Make Procedure for Computational Applications . . . . .	15
5.2	Makefiles for a portable Fortran 90 Library . . . . .	15
<b>6</b>	<b>Acknowledgements</b>	<b>16</b>
<b>A</b>	<b>makemake</b>	<b>18</b>
<b>B</b>	<b>Example Recursive Makefile</b>	<b>22</b>
<b>C</b>	<b>Examples of included File used in CLIPS</b>	<b>25</b>

# 1 Introduction

## 1.1 Fortran 90

The standard for Fortran used in this report is ISO/IEC 1539-1991 (in the USA, ANSI X3.198-1992), the so-called Fortran 90 standard. This has a number of significant advantages over previous dialects of Fortran and is particularly useful for providing modular and maintainable software for floating-point numerical applications. We recommend the book by Metcalf and Reid [9] for more information. Some of the important features introduced in this standard include:

- array operations;
- pointers;
- improved facilities for numerical computations including a set of numerical inquiry functions;
- parameterisation of the intrinsic types to permit processors to support short integers, very large character sets, more than two precisions for real and complex and packed logicals;
- user-defined derived data types composed of arbitrary data structures and operations upon those data structures;
- facilities for defining collections called “modules”, useful for global data definitions and for procedure libraries. These support a safe method of encapsulating derived data types;
- requirements on a compiler to detect the use of constructs that do not conform to syntax of the language or are obsolescent;
- a new source form, more appropriate to use at a terminal;
- new control constructs such as the SELECT CASE construct and a new form of the DO construct;
- the ability to write internal procedures and recursive procedures and to call procedures with optional and keyword arguments;
- dynamic storage (automatic arrays, allocatable arrays and pointers);
- improvements to the input-output facilities, including handling partial records and a standardised NAMELIST facility;
- many new intrinsic procedures, including those for machine constants.

We have found Fortran 90 particularly useful for our purposes, particularly in the development of new code and in constructing the CLIPS library [2]. Nevertheless there are some difficulties, especially in the area of support for irregular structures and sparse matrices and in binding to other libraries which were designed for older languages (e.g. C).

The difficulties encountered in using makefiles with Fortran 90, something not considered in the standard, and in devising a portable build procedure for the range of HPC platforms in current use, including pre-processing and linking to other languages, are addressed here.

## 1.2 Portable Parallel Applications

In a separate report [1] we introduced a suite of application codes which are used to measure performance of a variety of MPP and commodity-cluster systems. They are typical of those in use on current UK top-end facilities.

The example codes chosen for this initial study are ANGUS [14], FLITE3D [15], REALC [16], CRYSTAL98 [17], CASTEP [19], POL-ERSEM [18] and DL-POLY [13].

In addition to these whole codes, a number of single-processor kernels are run to assess system parameters affecting performance. These are taken from the DisCo Benchmark Suite [21]. All benchmarks are used annually to present a comparison of current HPC platforms at the Daresbury Machine Evaluation Workshop, see e.g. 12MEW [20].

The application codes have, through the work of the HPCI Centres [22] been optimised to produce the best run times on Daresbury IBM SP2-SC and EPCC Cray T3E-900 systems and then ported to the T3E-1200E which was the original core of the national CSAR service hosted by the University of Manchester. Subsequently the codes have been ported to the CSAR SGI Origin 2000 and Origin 3000 systems for production use and the Daresbury IBM SP3 for algorithmic development. Our analysis of the “best” compilation options is principally derived from experiences with these codes, see Section 3. Note that the codes mainly test double-precision floating-point performance, memory performance and parallel scalability.

For these benchmark codes, we have developed a portable recursive makefile procedure with an entry point for the target computer architecture. The target name can either be set as an environment variable, e.g. `setenv TARGET IBM-SP3` or given as an argument to `make`, e.g. `make IBM-SP3`. This procedure is described in Section 2. Another possibility, used in developing the parallel numerical algorithm library named CLIPS, is to use a machine-dependent makefile for each platform (via an included definition file). This is particularly useful in building more complex applications with a directory structure and is described in the same section.

## 1.3 The CLIPS Library

Over the last five years a number of parallel algorithms have been written at Daresbury Laboratory in order to optimise parallel applications, such as those mentioned above. With a view to making these routines more widely available and to promote better software re-usability and modularity a parallel library is being maintained. It is named CLIPS (the CLRC Library of Parallel Subroutines).

The Library is written mainly in Fortran 90 (with a little C where necessary) and uses standard MPI for communication. It is intended to be portable to most, if not all, contemporary serial and parallel computers including commodity clusters of PCs [7] and workstations.

Routines are being added to CLIPS as required. SMP support using OpenMP directives is also planned as it becomes more widely available and offers positive performance benefits. There is so far, however, no attempt to address any particular algorithmic area exhaustively, although there is some emphasis on eigensolvers [3] and multi-dimensional FFT [4] which are both efficient procedures widely found in modern applications. Documentation, example programs and a test suite are provided as part of the

Library distribution.

Further information about CLIPS is available in the separate technical report [2] and User Manual [6].

## 1.4 Computers Considered

In this report we only consider currently-available high-performance computers which are in widespread use, particularly in the UK. Examples of specific systems are:

Hitachi SR2100 or SR8000

Fujitsu VPP300 or VPP700

NEC SX/4 or SX/5

Cray T3E

SGI Origin or Octane series with R10k or R12k MIPS processors

IBM SP2 or SP3 or equivalent workstations e.g. model 4079/260

IBM SP4 systems with Power4 processors

Sun Ultra 2 and Ultra 3

Compaq XP1000 EV6 or EV67

HP C3000 or J5000 with PA8500 processor

Intel Pentium- or Athlon-based Linux commodity-cluster platforms

Alpha linux commodity-cluster platforms

## 2 Makefiles

The portable makefile procedures described in this report rely on using a “standard” set of file name extensions and make environment variables. They also depend on knowing the required order of compilation – the “dependency tree”.

Firstly let us clear up a common mis-understanding. the `make` utility searches for default file names in the order (i) `makefile`, then (ii) `Makefile`, after which it requires a file name to be specified with the `make -f` option.

### 2.1 Layout of the prototype makefile

We have adopted the following organisation for the makefile:

```
# comments
# include file (if any)
# default variable definitions
# objects required to build program
# suffixes
    all: $(TARGET)
# definition of program $(PROG) and installation procedures
```

```
# object and source dependencies to create dependency tree
# target-dependent definitions which over-ride defaults of the form
target:
    $(MAKE) $(PROG)
```

In this report we describe two different implementations of a makefile which can be used to build a “portable” Fortran 90 application. The first of these is implemented as above. The `make` utility parses the file to create its dependency tree and set of default definitions. It then calls `make` again starting at the `$(TARGET)` entry line which can over-ride the default definitions and has a list of actual targets. We refer to this as a “recursive” makefile.

In the second implementation we use a included file with separate definitions for each computer architecture, but no re-entry point. A super set of variables are chosen which are appropriate for the architectures most commonly encountered, and some may be left blank. These variables are used in the definitions to build the target as will now be described.

## 2.2 Default Rules for makefiles

We have used the following definitions of SUFFIXES:

```
.SUFFIXES:
.SUFFIXES:.F .f .o
.f.o: ;
    $(F90) $(F90FLAGS) $(INCLUDES) $(PFLAGS) $(INFO) $*.f
.F.o: ;
    $(CPP) $(CPFLAGS) $(LCPFLAGS) $*.F > $*.f
    $(F90) $(F90FLAGS) $(INCLUDES) $(PFLAGS) $(INFO) $*.f
```

Problems arise if files in the compilation set need different optimisation levels, which may for instance arise due to internal compiler errors. This requires a different specification for that file as part of the dependency tree or target analysis.

There are other exceptions, and we note that on the Cray, if `cpp` is to be invoked as a separate step, the rules must be modified somewhat to:

```
# for Cray
.F.o: ;
    $(CPP) $(CPFLAGS) $(LCPFLAGS) $*.F
    mv $*.i $*.f
    $(F90) $(F90LAGS) $(INCLUDES) $(PFLAGS) $(INFO) $*.f
```

## 2.3 Fortran File Names

We try to stick to the use of `*.f` or `*.F` for Fortran source files and those which must be pre-processed by `cpp` respectively. This is because Fortran 90 is now the standard Fortran, and we don't want to have to update to `*.f95`, `*.f2000` etc. However, for compatibility reasons, most compilers assume the `*.f` contains fixed-format rather than free-format source. A flag is therefore provided on the relevant `make` variables described below, e.g. `f90 -c -free`.

In the case where this is not possible another variant of the SUFFIXES lines in the makefiles must be used:

```
# e.g. for Absoft
.f.o: ;
    mv $*.f $*.f90
    $(F90) $(F90FLAGS) $(INCLUDES) $*.f90
.F.o: ;
    $(CPP) $(CPFLAGS) $(LCPFLAGS) $*.F > $*.f
    mv $*.f $*.f90
    $(F90) $(F90FLAGS) $(PFLAGS) $(INFO) $*.f90
```

## 2.4 Environment Variables

We use the following environment variables for `make`. Sticking to a “standard” set makes it easier to create new makefiles, and to define architecture-dependent include files.

TARGET= target architecture, can usually be set as a shell environment variable. This is particularly useful on Cray T-series systems where it is already required by the compilers.

ARCH= usually can be the same as TARGET. We may use it in the CPFLAGS

INCLUDES= where to look for module information when a USE statement is met.

LINCLUDES= local definitions – defined in actual makefile

CPP= name of C pre-processor or equivalent for single-step pre-processing e.g. `/usr/lib/cpp`

CPFLAGS= global define flags for CPP

LCPFLAGS= local define flags for CPP – defined in the actual makefile

CC= name of C compiler e.g. `cc -c`

CFLAGS= optimisation flags for C compiler

F90= name of free-format Fortran 90 compiler e.g. `f90 -c -free`

F90FLAGS= optimisation flags for Fortran compiler

PFLAGS= flags for shared-memory parallelisation

INFO= flags for printing information, e.g. intermediate source code

LD= name of loader e.g. f90 -o

LDFLAGS= flags for the loader e.g. -qipa

LIBS= names of any libraries required with path specified by -L <path> -l<name>

STAMP= a file to touch if compilation is successful. May be required to check dependencies in a directory structure

MODS= name of module information files which may need to be copied elsewhere as part of the installation. They should be in a path referenced via the INCLUDES variable

AR= name of library archive program e.g. ar rv

RAN= name of library table randomiser (if it exists, otherwise use e.g. RAN=ls)

The following set of default variable declarations is used in our software development projects:

```
ARCH=
INCLUDES=
LINCLUDES=
CPP= /usr/lib/cpp
CPFLAGS= -traditional -P -C
LCPFLAGS=
CC= cc -c
CFLAGS= -O
F90= f90 -c -free
F90FLAGS= -O
PFLAGS=
INFO=
LD= f90 -o
LDFLAGS=
LIBS= -lblas -lmpi
STAMP= stamp
MODS=
AR= ar rv
RAN= ranlib
```

These are used as a basis for the discussion of system-dependent preferred options in Section 3.

## 2.5 Tools and Scripts for makefiles

### 2.5.1 GNU Make

GNU make has some useful extensions to the usual UNIX make utility. These are required in a number of public-domain software suites. GNU make is the default with Linux systems.

A useful introduction to GNU Make by Richard Stallman and Roland McGrath (although dating from 1995) can be found at [http://www.cl.cam.ac.uk/texinfodoc/make\\_toc.html](http://www.cl.cam.ac.uk/texinfodoc/make_toc.html).

### 2.5.2 makemake

MakeMake is the name of a Polynesian God, said to have been the creator of Easter Island and responsible for its continuing fertility. `makemake` is a Perl script which attempts to automatically generate a makefile from a given set of source files. The source may be either C, FORTRAN77 or Fortran 90 languages or some combination of these. In our work we have used `makemake` to generate a dependency tree for Fortran modules and include files. This is then copied into the generic `makefile` which we are describing here.

Usage:

```
makemake <program name> <F90 compiler or fc or f77 or cc or c>
```

To run `makemake`, it is necessary to modify the first line of the script to point to the actual location of Perl, typically `/usr/bin/perl`. Other lines can be edited to specify the default compiler options, see Appendix A. We also modified the script to search Fortran files with all extensions for include and USE statements by changing lines such as

```
foreach $file (<*.F *.f90 *.f>)
```

The original `makemake` was written by Michael Wester, University of New Mexico, Albuquerque in 1995. See <http://www.math.unm.edu/~wester>. Modified versions are bundled with software development tools such as Code Crusader and the older Code Warrior for Linux platforms. See for instance the New Planet Software Web site for download options <http://www.newplanetsoftware.com/jcc>.

### 2.5.3 fmkmf and mkmf

`mkmf` is an SGI/GFDL Perl script to construct a makefile for Fortran 90 programs.

`fmkmf`, in addition to being an un-pronounceable pallindrome, is another Perl script to construct makefiles for Fortran 90 programs. `fmkmf` was written in late 1999 by H.C. Pumphrey, of the Edinburgh Department of Meteorology, see <http://www.met.ed.ac.uk/~hpc/fmkmf.html>. It has not yet been fully tested.

### 2.5.4 fmake

`fmake` is from Kate Hedstrom of Rutgers University, USA, and contains configuration files for a large number of compilers. See <http://maine.rutgers.edu/po/perl.html>.

### 2.5.5 sfmakedepend

`sfmakedepend` is another Perl script which creates Fortran 90 makefile dependencies. An early version was available from <ftp://ahab.rutgers.edu/pub/perl/perl4>. This is also by Kate Hedstrom.

## 2.6 Use of `cpp` for Conditional Compilation

Pre-processing, using the pragmas associated with `cpp`, is now accepted as the best way to do conditional compilation for both Fortran and C codes. Many modern Fortran compilers have an in-built pre-processing stage which can be used explicitly or implicitly (e.g. on Cray, SGI, Portland Group). This is often invoked using `.F` or `.F90` file extensions.

It is useful to standarise on the way pre-processing is done. We therefore propose a set of flags which can be used in a fairly general way.

We discussed a set of directives with functional declarations:

```
machine_*
vendor_*
architecture_*
message_*
compiler_*
developer_*
```

Some explicit examples, which will be extended in the future, are:

MACHINE:

```
machine_crayt3e
machine_irix64
machine_irix32
machine_ibmrios
machine_decalpha
machine_fujitsu
machine_hp700
machine_linux86
```

PLATFORM:

x11

VENDOR:

vendor\_cray  
vendor\_ibm  
vendor\_fujitsu  
vendor\_sgi  
vendor\_compaq  
vendor\_hp  
vendor\_sun

ARCHITECTURE

MESSAGE

SHMEM  
MPI  
message\_shmem  
message\_openmp  
message\_mpich  
message\_lammpi

COMPILER

compiler\_pgi

DEVELOPER

developer\_rja  
developer\_yfh

If pre-processing is carried out as an explicit step, as we propose here, there are possible snags. Some compilers, for example `f90 -eP` on the Cray produce an intermediate file with name `*.i`. This is non-standard and we have edited the makefile `SUFFIXES` to include the extra line as described above:

```
cp $*.i $*.f
```

None of these proposed `cpp` directives have so far been used.

### 3 Compiler Options

We consider the available compilers on HPC platforms and list some of the options which have been found to be useful. References are given to further information on these compilers. Other useful information can be found from various Web sites on which benchmark results are regularly published, e.g. SPECfp95, see <http://www.specbench.org>. In the main, our default compilation options are found to be similar to the SPECfp95 baseline options. Some further performance may however be possible in certain cases, but careful experimentation is required and options may be different for individual routines which really does not fit the portable strategies proposed here.

Note that we only reproduce below differences from the template makefile given in Section 2.

#### 3.1 Absoft Fortran v1.01 on Linux Pentium Platforms

```
CPP= gcc -E
F90= f90 -c
```

The compiler requires free-form source to be in files with extension `.f90`. If modules (with the `.mod` extension) are present in another directory the `-p` flag should be used. However there was a bug which prevented this working correctly in the version tested.

See <http://www.absoft.com>.

For use on parallel systems, e.g. Beowulf clusters, either the LAM-mpi from Ohio State University or MPICH from Argonne National Laboratory may be installed. We will assume they are installed in a standard place such as `/usr/local/mpich` or `/usr/local/lam`.

##### Linking with LAM-mpi

```
LIBS= -L/usr/local/lam/lib -lmpi -larg -ltstdio -ltrillium -lt -lblas
```

##### Linking with MPICH

```
LIBS= -lpgftnrtl -L/usr/local/mpich/lib -lmpich -lmpichf -lblas
```

#### 3.2 Compaq

```
CC= cc -c -std
F90=f90 -c -free
F90FLAGS= -O5 -fast -arch ev6 -tune ev6
# enable parallelism
PFLAGS=-omp
# OpenMP, threads and DMXL optimised library if available
LIBS=-lpthread -ldmxlp -lmpi
```

### 3.3 Cray

```

CPP= f90 -eP
CPPFLAGS= -Wp'-P'
F90= f90 -c -ffree
F90FLAGS= -dp -O3,unroll2,pipeline2
LDFLAGS=blas.cld
LIBS=

```

Note that the SciLib library, which contains BLAS, LAPACK and ScaLAPACK routines and also the SHMEM and MPI libraries are loaded automatically. We also note that Cray consider 64-bit arithmetic to be the default and all their BLAS and other numerical routines are supplied with “single precision” names, such as SAXPY. These actually apply to what are more commonly referred to as “double precision” data. Thus a code which calls DAXPY will not work on the Cray. Calling names of routines can however be changed in the link step by supplying a file “blas.cld” along with any libraries required. This file contains a list of names with the format:

```
equiv(DAXPY)=SAXPY
```

### 3.4 Fujitsu

```

CC= ccpx -c
CFLAGS= +p -DNDEBUG -DFUNCPROTO
F90= frptx -c
F90FLAGS= -Kfast,V8PFMADD,prefetch,gs -Oe -KVPP700 -Wv,-m1 -X7 -Pf -Sw
-Elmipue -Z \${*.list}
LD= /usr/local/bin/vppld -o
LDFLAGS= -Wl,-J,-P -dn
LIBS= -L/usr/lang/mpi2/lib -lmpi -lmp -lpx -lcvp -lgen -lssl2vp

```

### 3.5 HP

```

F90= f90 -c
F90FLAGS= +Oall +0dataprefetch -Wl,+pd,64K
LDFLAGS= -Wl,-aarchive

```

### 3.6 Hitachi

```

F90= xf90 -c
F90FLAGS= -W0,'opt(o(3)),langlvl(hf(77),save(0))'

```

### 3.7 IBM Fortran v6.1 and v7.1

```

F90= mpixlf_r -c -qfree=f90
F90FLAGS= -O4 -qhot -qipa=level=2:partition=large -qarch=pwr3
-qtune=pwr3 -qnosave
# switch on processing of OpenMP directives
PFLAGS= -qsmp=nested_par
INFO= -qreport=smpelist
LD= mpixlf_r -o
LDFLAGS= -qipa=level=2:partition=large -qsmp -bmaxdata:2000000000
# optimised IBM library if available
LIBS= -lesslsmpl

```

We found that the `-qreport=smpelist` flag produced internal errors with some codes on v6.1. We also found that the `-qipa` option was not reliable and produced incorrect code in some rare cases. Explicit code inlining could be used instead via `tt -qipa=inline=xxx` where `xxx` is a list of files to inline.

On Power4 systems the above options will work or `pwr3` can be replaced with `pwr4` with a slight improvement in performance. The `-qhot` options should be tested as it is not always beneficial.

### 3.8 Pacific Sierra Research

PSR offer a F90 compiler for Linux systems, a free version is available for personal use. See <http://www.psrv.com/lxf90.html>.

### 3.9 Portland Group Compiler on Linux Platforms

See <http://www.pgi.com>.

#### Linking with LAM-mpi compiled with the GNU compilers

```

F90= pgf90 -c -Mfree
INCLUDES= -I/usr/local/lam/include
F90FLAGS= -fast -Knoieee -Mdalign -Msecond_underscore
LD= pgf90 -o
LIBS= -L/usr/local/lam/lib -lmpi -largz -ltstdio -ltrillium -lt -lblas

```

#### Linking with MPICH compiled with the PGI compilers

Note that if MPICH is compiled with the GNU compilers it will not work with a PGI F90 application.

```

F90= pgf90 -c -Mfree
INCLUDES= -I/usr/local/mpich/include

```

```
F90FLAGS= -fast -Knoieeee -Mdalign
LD= pgf90 -o
LIBS= -lpgftnrtl -L/usr/local/mpich/lib -lmpich -lmpichf -lblas
```

For linking to other libraries compiled with the GNU compiler set there is a compatibility flag to include on the LDFLAGS line: `-lg77libs`. This searches for the library `/usr/lib/libf2c.a` which should be a link to

```
/usr/local/gnu/i386-redhat-linu8/egcs-1.66.99/pibg2c.a
```

### 3.10 SGI

```
F90= f90 -c -freeform
F90FLAGS= -Ofast -OPT:Olimit=0 -LN0:prefetch_ahead=1:auto_dist=on
# swith on automatic parallelisation
PFLAGS= -pfa -mplist
LDFLAGS= -mp
LIBS= -lblas -lmpi -lmp -lmpc -lftn -lbsd -lm
```

### 3.11 SUN

```
F90= /opt2/SUNWspro/bin/f90 -c
F90FLAGS=-fast -xarch=v8plusa -xchip=ultra2 -xprefetch
PFLAGS= -xparallel
LD= /opt2/SUNWspro/bin/f90 -o
# optimised SunPerf library if available
LIBS= -xlic_lib=sunperf
```

## 4 Module Information Files

There is no standard make procedure for Fortran 90 applications. This is particularly apparent when we consider how the compiler might acquire the information required to process `USE` statements. This information principally consists of an implicit interface defining how function and subroutine arguments and returned variables are defined and used. This is particularly important for optional arguments and array sections. If no interface is available the old FORTRAN77 conventions are assumed and the compiler may have to copy variables onto the stack to ensure data is contiguous in memory.

Explicit interfaces may be provided, and information passed to the compiler about how to find these. We will return to this point when discussing how to build a portable library later.

## 4.1 Absoft

The Absoft compiler requires free-form Fortran 90 code to be in files with extension “.f90”. We noted in Section 2 how we modified the SUFFIXES definitions in the general-purpose makefile in this case. The compiler produces “.mod” files which should be therefore available in the source directory or on a path specified with the `-p` option. However when we tried this we found a bug which may have been fixed in later releases. This bug meant there was a confusion between upper and lower case module file names required in different phases of the compilation procedure.

See <http://www.absoft.com>.

## 4.2 Compaq

Compaq true64 compilers read “.f” “.F” or “.f90” files and produce “.o” and “.mod” files. This is the same procedure as with IBM.

## 4.3 Cray T Series

Cray MPP Fortran 90 compilers read “.f” “.F” or “.f90” files and produce “.o” files. All module information is contained in the “.o” files which should be therefore available in the source directory or on a path specified with the `-p` option. “.a” files are an alternative source of module information. This means that Fortran 90 programs require no special treatment.

## 4.4 Hitachi

The Hitachi Fortran 90 compiler takes files with a “.f90” extension. Only one module must be present in the file which must have the same name as the module. The source file is parsed when a `USE` statement is found. Only “.o” files are produced. For user-accessible modules a “.f90” file must be available in the distribution directory with at least a list of explicit public interfaces.

## 4.5 IBM

IBM compilers read “.f” “.F” or “.f90” files and produce “.o” and “.mod” files. The “.mod” files in the current directory or a directory referenced via the `-I` flag are parsed when a `USE` statement is met. A “.mod” file is therefore required for each user-accessible library module which must contain at least a list of explicit public interfaces.

## 4.6 Portland Group

See <http://www.pgi.com>.

## 4.7 SGI

IBM compilers read “.f” “.F” or “.f90” files and produce “.o” and “.mod” files. This is the same procedure as with IBM.

Note older, now obsolete, versions produced “.kmo” files.

## 5 Examples of Makefiles

### 5.1 Recursive Make Procedure for Computational Applications

The procedure, already described in Section 2, was originally devised by Tim Forrester when he was developing the DL\_POLY Molecular Dynamics Program [13]. We now use this widely for stand-alone applications. An example of a such a makefile for an atomic physics application is given in Appendix B.

### 5.2 Makefiles for a portable Fortran 90 Library

In this section we describe how we build a portable Fortran 90 library, CLIPS, based on the procedures already described.

A variant of the set of environment variables presented in Section 2 is used in a file `INCLUDE.make` defined for each architecture of interest. This is illustrated by the include file for the IBM Power3 SP system shown in Appendix C.

A typical makefile for one of the CLIPS library test programs can then be simply written as follows:

```
#####
# Makefile for BFG test code from CLIPS library
#####
# include file as above
include ../src/INCLUDE.make

# name of program and source files
PROG=test.out
TEST= test_eso.o get_map.o etest.o

# what is to be done
all: $(PROG)

$(PROG): $(TEST)
    $(LD) $(PROG) $(LDFLAGS) $(TEST) -L ../lib -lbfg $(LIBS)

# any dependencies could be listed here

# clean up
clean:
```

```
rm -f $(TRASH)
```

The CLIPS library is generally provided in compiled form via a randomised library file `libclips.a` which contains the code objects for a particular architecture. However it is also necessary to provide a set of interface blocks for publicly accessible routines, in order for the Fortran 90 compiler to validate subroutine calls and optimise the passing of data structures on the subroutine boundaries. There is no cross-architecture standard for the way this is to be done. Platforms may expect files with extensions such as “.mod”, “.o” “.a”, as produced by compiling a prior module in the dependancy tree, or may simply parse the relevant source files again when a USE statement is encountered. See section 3. If “.mod” files can be used they can be copied to `$(CLIPSHOME)/include`. Alternatively a set of explicit interfaces in source form can be made available as a separate file.

## 6 Acknowledgements

The preparation of this report, and the work described, was funded by EPSRC partly through a grant GR/K82635 to the HPCI Centre at Daresbury Laboratory, grant GR/N09688 to the UKHEC Collaboration and partly through its Service Level Agreement with the CSE Department.

## References

- [1] R.J. Allan and M.F. Guest *Performance of HPC Applications – the CARC Benchmark Suite* (CLRC Daresbury Laboratory, 2000)
- [2] R.J. Allan and Y.F. Hu *The CLRC Library of Parallel Subroutines – CLIPS* (Daresbury Laboratory, 1999) draft
- [3] R.J. Allan and I.J. Bush *Parallel Diagonalisation Routines* Edition 1 (CLRC Daresbury Laboratory, 1996)
- [4] R.J. Allan and I.J. Bush *Serial and Parallel FFT Routines* Edition 1 (CLRC Daresbury Laboratory, 1996)
- [5] R.J. Allan, Y.F. Hu and P. Lockey *Survey of Parallel Numerical Analysis Software* Edition 2, Technical Report DLT-99-01 (CLRC Daresbury Laboratory, April 1999)
- [6] R.J. Allan, Y.F. Hu, I.J. Bush and A.G. Sunderland *CLIPS: CLRC Library of Parallel Subroutines. User Manual and Specifications* (CLRC Daresbury Laboratory, 1999)
- [7] R.J. Allan, S.J. Andrews, M.F. Guest, P.M. Oliver, D. Henty, L. Smith, S. Telford and S. Booth *Designing and Building Beowulf-class Cluster Computers* (UKHEC 2000) See <http://www.ukhec.ac.uk>
- [8] J. Dongarra and J. Waśniewski *High Performance Linear Algebra Package – LAPACK90* UNI-C Report UNIC-98-01 (Danish Computing Centre for Research and Education, Technical University of Denmark, 1998)
- [9] M. Metcalf and J. Reid *Fortran 90 Explained* (Oxford University Press, 1990)
- [10] Fortran 90 standard ISO/IEC 1539-1991 and ANSI X3.198-1992
- [11] *MPI: A message-passing Interface Standard* MPI Forum, (June 1995)  
A. Skjellum, N.E. Doss and P.V. Bangalore *Writing libraries in MPI* in “Proceedings of the Scalable Parallel Libraries conference” A. Skjellum and D.S. Reese (eds.) (IEEE Computer Society Press, 1993). Available by anonymous ftp from  
<ftp://aurora.cs.msstate.edu/pub/reports/SPLC93>
- [12] *NetLib* On-line repository of numerical algorithms and other high-performance computing software at URL <http://www.netlib.org>

- [13] W. Smith [http://www.dl.ac.uk/TCSC/Software/DL\\_POLY](http://www.dl.ac.uk/TCSC/Software/DL_POLY) DL\_POLY is a general parallel molecular-dynamics program used by several HPCI Consortia.
- [14] ANGUS is a demonstrator code of a regular-grid CFD engineering code which uses parallel domain decomposition.
- [15] FLITE3D is an irregular grid finite-element code which is used by British Aerospace.
- [16] REALC is a program used by the ChemReact'98 consortium for calculation of chemical reaction rates in atom-diatom collisions.
- [17] CRYSTAL is a periodic Hartree-Fock code used by the Materials Chemistry HPCI Consortium
- [18] the Water Quality Model code is used by the Bidston Shel-Sea modelling HPCI Consortium.
- [19] CASTEP is a code which implements the Car-Parrinello algorithm for the total energy calculations of large atomic systems. It is used by the UKCP Consortium.
- [20] R.J. Allan, S.J. Andrews and M.F. Guest (eds.) Proc. 12th Daresbury Machine Evaluation Workshop, (Daresbury Laboratory, 2001). See <http://www.cse.clrc.ac.uk/Activity/DisCo>
- [21] DisCo Benchmark Suite and results See <http://www.cse.clrc.ac.uk/Activity/DisCo>
- [22] R.J. Allan, M.F. Guest, D.S. Henty, D. Nicole and A.D. Simpson (eds.) *High Performance Computing* Proc. HPCI'98 Conference 1998, (Plenum/Kluwer Publishing, 1999) ISBN 0-306-46034-3. See <http://www.cse.clrc.ac.uk/Activity/HPCI>
- [23] IBM XL Fortran for AIX Users' Guide v5.1 (IBM, November 1997) order number SC09-2606-00

## A makemake

The makemake Perl script is as follows. It can be downloaded from the Web at URL <http://www.math.unm.edu/~wester>.

```
#!/usr/bin/perl
#####
# Usage: makemake {<program name> {<F90 compiler or fc or f77 or cc or c>}}
#
# Generate a Makefile from the sources in the current directory. The source
# files may be in either C, FORTRAN 77, Fortran 90 or some combination of
# these languages. If the F90 compiler specified is cray or parasoft, then
# the Makefile generated will conform to the conventions of these compilers.
# To run makemake, it will be necessary to modify the first line of this script
# to point to the actual location of Perl on your system.
#
# Written by Michael Wester <wester@math.unm.edu> February 16, 1995
# Cotopaxi (Consulting), Albuquerque, New Mexico
# sent to me by Joel Schulman <schulman@madmax.hrl.hac.com>
#####
#
open(MAKEFILE, "> Makefile");
#
print MAKEFILE "PROG =\t$ARGV[0]\n\n";
#
# Source listing
#
print MAKEFILE "SRCS =\t";
@srcs = <*.f90 *.f *.F *.c>;
&PrintWords(8, 0, @srcs);
print MAKEFILE "\n\n";
#
# Object listing
#
print MAKEFILE "OBS =\t";
@objs = @srcs;
foreach (@objs) { s/\.[^.]+\$/./o/ };
&PrintWords(8, 0, @objs);
print MAKEFILE "\n\n";
#
# Define common macros
#
print MAKEFILE "LIBS =\t\n\n";
print MAKEFILE "CC = cc\n";
print MAKEFILE "CFLAGS = -O\n";
print MAKEFILE "FC = f90\n";
print MAKEFILE "FFLAGS = -O\n";
print MAKEFILE "F90 = f90\n";
print MAKEFILE "F90FLAGS = -O\n";
print MAKEFILE "LDFlags = \n\n";
#
# make
#
```

```

print MAKEFILE "all: \$(PROG)\n\n";
print MAKEFILE "\$(PROG): \$(OBJS)\n";
print MAKEFILE "\t\$((", &LanguageCompiler($ARGV[1], @srcs);
print MAKEFILE ") \$(LDFLAGS) -o \$$@ \$(OBJS) \$(LIBS)\n\n";
#
# make clean
#
print MAKEFILE "clean:\n";
print MAKEFILE "\trm -f \$(PROG) \$(OBJS) *.mod\n\n";
#
# Make .f90 a valid suffix
#
print MAKEFILE ".SUFFIXES: \$(SUFFIXES) .f90\n\n";
#
# .f90 -> .o
#
print MAKEFILE ".f90.o:\n";
print MAKEFILE "\t\$(F90) \$(F90FLAGS) -c \$$<\n\n";
#
# Dependency listings
#
&MakeDependsf90($ARGV[1]);
&MakeDepends("*.f *.F", '^\\s*include\\s+["\\']([^\\"\\']+)["\\']');
&MakeDepends("*.c", '^\\s*#\\s*include\\s+["\\']([^\\"\\']+)["\\']');
#
# &PrintWords(current output column, extra tab?, word list); --- print words
#   nicely
#
sub PrintWords {
    local($columns) = 78 - shift(@_);
    local($extratab) = shift(@_);
    local($wordlength);
    #
    print MAKEFILE @_[0];
    $columns -= length(shift(@_));
    foreach $word (@_) {
        $wordlength = length($word);
        if ($wordlength + 1 < $columns) {
            print MAKEFILE " $word";
            $columns -= $wordlength + 1;
        }
        else {
            #
            # Continue onto a new line
            #
            if ($extratab) {
                print MAKEFILE " \\n\t\t\t$word";
                $columns = 62 - $wordlength;
            }
            else {
                print MAKEFILE " \\n\t\t\t$word";
                $columns = 70 - $wordlength;
            }
        }
    }
}

```

```

    }
}
#
# &LanguageCompiler(compiler, sources); --- determine the correct language
#   compiler
#
sub LanguageCompiler {
    local($compiler) = &toLower(shift(@_));
    local(@srcs) = @_;
    #
    if (length($compiler) > 0) {
        CASE: {
            grep(/^$compiler$/, ("fc", "f77")) &&
                do { $compiler = "FC"; last CASE; };
            grep(/^$compiler$/, ("cc", "c")) &&
                do { $compiler = "CC"; last CASE; };
            $compiler = "F90";
        }
    }
    else {
        CASE: {
            grep(/\..f90$/, @srcs) && do { $compiler = "F90"; last CASE; };
            grep(/\.(f|F)$/, @srcs) && do { $compiler = "FC"; last CASE; };
            grep(/\..c$/, @srcs) && do { $compiler = "CC"; last CASE; };
            $compiler = "???" ;
        }
    }
    $compiler;
}
#
# &toLower(string); --- convert string into lower case
#
sub toLower {
    local($string) = @_[0];
    $string =~ tr/A-Z/a-z/;
    $string;
}
#
# &uniq(sorted word list); --- remove adjacent duplicate words
#
sub uniq {
    local(@words);
    foreach $word (@_) {
        if ($word ne $words[$#words]) {
            push(@words, $word);
        }
    }
    @words;
}
#
# &MakeDepends(language pattern, include file sed pattern); --- dependency
#   maker
#
sub MakeDepends {

```

```

local(@incs);
local($lang) = @_[0];
local($pattern) = @_[1];
#
foreach $file (<${lang}>) {
    open(FILE, $file) || warn "Cannot open $file: ${!}\n";
    while (<FILE>) {
        /$pattern/i && push(@incs, $1);
    }
    if (defined @incs) {
        $file =~ s/\.[^.]+\$/./;
        print MAKEFILE "$file: ";
        &PrintWords(length($file) + 2, 0, @incs);
        print MAKEFILE "\n";
        undef @incs;
    }
}
}
#
# &MakeDependsf90(f90 compiler); --- FORTRAN 90 dependency maker
#
sub MakeDependsf90 {
    local($compiler) = &toLower(@_[0]);
    local(@dependencies);
    local(%filename);
    local(@incs);
    local(@modules);
    local($objfile);
    #
    # Associate each module with the name of the file that contains it
    #
    foreach $file (<*.F *.f90 *.f>) {
        open(FILE, $file) || warn "Cannot open $file: ${!}\n";
        while (<FILE>) {
            /\s*module\s+([\s!]+)/i &&
                ($filename{&toLower($1)} = $file) =~ s/\.f90$/./;
        }
    }
    #
    # Print the dependencies of each file that has one or more include's or
    # references one or more modules
    #
    foreach $file (<*.F *.f90 *.f>) {
        open(FILE, $file);
        while (<FILE>) {
            /\s*include\s+[\"'\](["\']+)[\"']/i && push(@incs, $1);
            /\s*use\s+([\s!]+)/i && push(@modules, &toLower($1));
        }
        if (defined @incs || defined @modules) {
            ($objfile = $file) =~ s/\.f90$/./;
            print MAKEFILE "$objfile: ";
            undef @dependencies;
            foreach $module (@modules) {
                push(@dependencies, $filename{$module});
            }
        }
    }
}

```

```

    }
    @dependencies = &uniq(sort(@dependencies));
    &PrintWords(length($objfile) + 2, 0,
                @dependencies, &uniq(sort(@incs)));
    print MAKEFILE "\n";
    undef @incs;
    undef @modules;
    #
    # Cray F90 compiler
    #
    if ($compiler eq "cray") {
        print MAKEFILE "\t\$(F90) \$(F90FLAGS) -c ";
        foreach $depend (@dependencies) {
            push(@modules, "-p", $depend);
        }
        push(@modules, $file);
        &PrintWords(30, 1, @modules);
        print MAKEFILE "\n";
        undef @modules;
    }
    #
    # ParaSoft F90 compiler
    #
    if ($compiler eq "parasoft") {
        print MAKEFILE "\t\$(F90) \$(F90FLAGS) -c ";
        foreach $depend (@dependencies) {
            $depend =~ s/\.$/.f90/;
            push(@modules, "-module", $depend);
        }
        push(@modules, $file);
        &PrintWords(30, 1, @modules);
        print MAKEFILE "\n";
        undef @modules;
    }
}
}
}

```

## B Example Recursive Makefile

```

#####
# example makefile based on original by Tim Forrester
# RJA 1999
#####

# initial set of generic declarations, can over-ride for other machines
CPP= /usr/lib/cpp
CPFLAGS=-P -C -DNOFLUSH
FC=f90 -c
FFLAGS=-O
PFLAGS=

```

```

INFO=
LD=f90 -o
LDFLAGS=
LIBS=-lblas

# define which specific subroutine to use
POT= h+h2
MPI=precision.o mpisp2.o mpidummy.o
SOURCEP=commons2.o main.o hpsi2.o initialp.o setexpp.o \
    getvj.o jacobis2.o angles.o gridx.o $(POT).o vfftpk.o \
    jacobio.o fft.o plotwfn.o angkin.o clips.o readslab.o

#defaults
.SUFFIXES:
.SUFFIXES:.F .f .o
.F.o:
# for Cray
#   $(CPP) $(CPFLAGS) $*.F
#   cp $*.i $*.f
#   $(FC) $(FFLAGS) $(PFLAGS) $(INFO) $*.f
#   $(CPP) $(CPFLAGS) $*.F $*.f
#   $(FC) $(FFLAGS) $(PFLAGS) $(INFO) $*.f
.f.o: ;
    $(FC) $(FFLAGS) $(PFLAGS) $(INFO) $*.f

# what is to be done?
all: $(TARGET)

$(PROGP): $(MPI) $(SOURCEP) $(BLAS)
    $(LD) $(PROGP) $(LDFLAGS) $(PFLAGS) $(INFO) $(SOURCEP) $(BLAS) $(MPI) $(FLIBS)

# dependencies of modules - perhaps generated with makemake ?
main.o: main.f $(MPI) commons2.o setexpp.o initialp.o hpsi2.o plotwfn.o jacobis2.o clips.o readslab.o
hpsi2.o: hpsi2.f $(MPI) commons2.o clips.o angkin.o
initialp.o: initialp.f precision.o commons2.o fft.o angles.f
initialr.o: initialr.f precision.o commons2.o fft.o
jacobis2.o: jacobis2.f precision.o commons2.o
angles.o: angles.f precision.o commons2.o
getvj.o: getvj.f precision.o jacobio.o
setexpp.o: setexpp.f precision.o commons2.o angkin.o $(POT).o jacobis2.o angles.o getvj.o gridx.o clips.o
setexpr.o: setexpr.f precision.o commons2.o angkin.o $(POT).o jacobis2.o angles.o getvj.o gridx.o clips.o
gridx.o: gridx.f precision.o clips.o
$(POT).o: $(POT).f precision.o
plotwfn.o: plotwfn.f precision.o commons2.o
angkin.o: angkin.f precision.o commons2.o
commons2.o: commons2.f precision.o clips.o
clips.o: clips.F precision.o
vfftpk.o: vfftpk.F
mpi.o: mpi.f
mpisp2.o: mpisp2.f precision.o
readslab.o: readslab.f $(MPI)
diffcross.o: diffcross.f angles.o

# Specific make for each target now follows

```

```
##### serial Linux absoft f90 #####
absoft:
    $(MAKE) BLAS="dcopy.o daxpy.o" FLIBS="" $(PROGP)

##### serial Linux vast f90 #####
vast:
    $(MAKE) BLAS="dcopy.o daxpy.o" FLIBS="" $(PROGP)

##### Beowulf Absoft-MPI #####
Absoft-lam:
    $(MAKE) CPP="/bin/true" CPFLAGS="'-DIRTC -DNOFLUSH -P'" \
    FFLAGS="-YEXT_SFX=_ -YEXT_NAMES=LCS -I/usr/local/lam/h" \
    FLIBS="-L/usr/local/lam/lib -lmpi -largs -ltstdio -ltrillium -lt -L/
usr/local/lib -lblas" $(PROGP)

##### Beowulf PGI-MPI #####
PGI-lam:
    $(MAKE) CPP="/bin/true" CPFLAGS="'-DIRTC -DNOFLUSH -P'" \
    FC="pgf90 -c -g77libs" LD="pgf90 -o" \
    FFLAGS="-O -Msecond_underscore -Mdalign -I/usr/local/lam/h" \
    MPI="precision.o mpilam.o" \
    FLIBS="-L/usr/local/lam/lib -lmpi -largs -ltstdio -ltrillium -lt -L/
usr/local/lib -lblas" $(PROGP)

##### serial IBM PPC Workstation #####
SUN-E:
    $(MAKE) $(PROGP)

##### IBM SP2 parallel system #####
IBM-sp2:
    $(MAKE) FC="mpxlf -c -qfree=f90" FFLAGS="-g -qnosave -qarch=pwr2" \
    FLIBS="-lesslp2 -lmpi" MPI="precision.o mpisp2.o" LD="mpxlf -o" $(PROGP)

##### IBM smp Workstation#####
IBM-smp:
    $(MAKE) FC="xlf90_r -qfree=f90 -c" \
    FFLAGS="-O3 -qsmp=noauto -qnosave -qalias=noaryovrlp -qarch=auto" \
    FLIBS="-lmass -lesslsmf" LD="xlf90_r -o" $(PROGP)

##### serial SGI Workstation #####
SGI:
    $(MAKE) FC="f90 -c -freeform" MPI="precision.o mpisgi.o" \
    FFLAGS="-O3 -64 -mips4 -r10000 -OPT:const_copy_limit=10838" \
    LDFLAGS="-64 -mips4" FLIBS="-lblas -lmpi" $(PROGP)

##### Cray T3E #####
Cray-t3e:
    $(MAKE) CPP="f90 -eP" CPFLAGS="-Wp' -DIRTC -DNOFLUSH -P -DCray'" \
    FFLAGS="-dp -ffree -O3,unroll2,pipeline2" MPI="precision.o mpit3e.o" \
    FLIBS="blas.cld -lmfastv" $(PROGP)

##### clean #####
clean:
    rm *.o *.exe *~ *# *.mod *.M *.kmo *.f90
```

```
##### touch #####
touch:
    touch *.f
    touch *.F
```

## C Examples of included File used in CLIPS

An examples of the included file “INCLUDE.make” used in CLIPS is shown here for the instantiation on the IBM SP3.

```
# variable definitions
ARCH=IBM-sp3
INCLUDES=-I $(CLIPSHOME)/include -I ../include
LINCLUDES=
CPP=/usr/lib/cpp
CPFLAGS=-DSP2 -DMPI -Dibmrios
LCPFLAGS=-DDCFT
CC=xlc_r -c
CFLAGS=$(INCLUDES)
FC=mpxlf_r -c
FFLAGS=-O3 -qarch=pwr3 -qnosave
F90=mpxlf_r -c -qfree=f90
F90FLAGS=-O3 -qarch=pwr3 -qtune=pwr3 -qnosave
PFLAGS= -qsmp=nested_par
INFO=
LD=mpxlf_r -o
LDFLAGS=-qsmp=nested_par -bmaxdata:200000000
LIBS=-L $(CLIPSHOME)/lib -lclips -lesslmp -lmpi
AR= ar rv
RAN=ls

# all the temporary and module files
STAMP=stamp
MODS=*.mod
TRASH= *.o $(MODS) *.out *~ stamp

# default builds
.SUFFIXES:
.SUFFIXES: .F .f90 .f .c .o
.f90.o:
    cp $*.f90 $*.f
    $(F90) $(F90FLAGS) $(INCLUDES) $(LINCLUDES) $(PFLAGS) $(INFO) $*.f
.f.o: ;
    $(F90) $(F90FLAGS) $(INCLUDES) $(LINCLUDES) $(PFLAGS) $(INFO) $*.f
.F.o: ;
    $(CPP) $(CPFLAGS) $(LCPFLAGS) $*.F > $*.f
    $(F90) $(F90FLAGS) $(INCLUDES) $(LINCLUDES) $(PFLAGS) $(INFO) $*.f
.c.o:
    $(CC) $(CFLAGS) $*.c
```