Parallel Application Software on High Performance Computers

A guide to Fortran 90 programming and the Cray T3D

The CCLRC HPCI Centre at Daresbury Laboratory

A.Sunderland, C.J.Noble, R.J. Allan, I.J.Bush and K.Maguire

Edition 2: 9th July 1996

Abstract

The `f90` compiler v0.1.1 on the Cray T3D massively parallel processor is a subset of the Fortran 90 international standard ISO/IEC 1539-1991. The most obvious deviation is the lack of modules and consequent difference of scope in variable declarations. We examine some consequences of this reduced language implementation and describe how a typical Fortran 90 program might be ported to the T3D. We also comment on some other programming styles inherited from FORTRAN77 days, and their possible side effects.

This report is available from http://www.dl.ac.uk/TCSC/HPCI/

Contents

1 Introduction

Cray's Fortran 90 v0.1.1 for the T3D became available in mid January 1996. It is notable that whilst `f90` on all other Cray platforms, including SPARC-based ones, is fully compliant with the international standard ISO/IEC 1539-1991 [10] `f90` on the current MPP system, the T3D is not. It has however been suggested by Cray that subsequent releases for the T3E will contain a full implementation of all new language features.

Fortran 90 is an 'evolution' of the previous standard FORTRAN77. This policy has allowed compiler technology developed over some 30 years to be extended to embrace new features with no loss of the superb optimisation, which is why Fortran has become the language of choice for computational science. There are now a number of other languages with good optimising compilers, but Fortran is still the most suitable for many applications, especially those involving multi-dimensional data structures. Applications requiring more complex dynamic memory and data handling, such as linked lists or objects to be manipulated in, for instance, visualisation packages, have recently used C, C++ or a combination of these with Fortran bindings. Fortran 90 is therefore a response to the sophisticated requirements of programmers now able to access supercomputer power, but specifically targeted at the numerical computing community.

There are now numerous good books describing Fortran 90, we cite some in the bibliography.

In summary the new features compared to FORTRAN77 are:

- A means for the language to evolve by labelling some features 'obsolescent';
- Array operations;
- Pointers;
- Improved facilities for portable numerical computation including a set of numeric enquiry functions;
- Parameterisation of the intrinsic types, to permit processors to support, for example, short integers, very large character sets, more than two precisions for real;
- User-defined derived data types composed of arbitrary data structures and operations upon these structures;
- Facilities for defining collections called 'modules', useful in 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 the syntax of the language, or are obsolescent;
- A new source form, more appropriate to the terminal rather than punched cards;
- New control constructs such as SELECT CASE, WHERE and a generalised form of DO;
- The ability to write internal procedures and recursive procedures, and to call procedures with optional keywords and arguments;

- Dynamic storage (automatic arrays, allocatable arrays, pointers);
- Improvements to the i/o facilities including handling partial records and a standardised NAMELIST facility;
- Many new intrinsic procedures, the vast majority able to act element-wise on arrays.

Many of these new featuers in fact existed in so-called 'extensions' to FORTRAN77 on a number of computers. `DO ... END DO` and `DO WHILE` for instance were widely available, and some compilers permitted procedures to be recursive. Automatic arrays were also handled on many platforms. Almost all of these extensions were incorporated providing they were compatible with the general philosophy.

Other features are clearly extensions of what was already built into FORTRAN77 syntax. For instance the optional arguments now possible on user-defined procedures extend the philosophy of the READ and WRITE statement syntax where it was possible to write `READ(UNIT=5, FMT='(...)', ERR=99)...` or `READ(5,'(...)',99)...`

Obsolescent features should definately *not* be used in any new coding, since they will almost certainly be removed from future language releases. This is either because they are bad practice or hinder optimisation. The current list is:

- arithmetic IF;
- branching to an END IF statement from outside its block;
- real and double precision DO variables and control expressions;
- shared DO termination and DO termination on a statement other than on a CONTINUE or END DO statement;
- ASSIGN and the assigned GO TO;
- alternate RETURN;
- PAUSE;
- assigned FORMAT specifiers.

At EPCC cross compilers are accessible on front end Kelvin (J916). Before doing a compilation you need to set the environment variable TARGET to Cray-T3D. Then compile as follows:

```
f90 -o hello.out hello.f
```

where file extensions `.f` imply fixed format and `.f90` imply free format. The flag `-f free` can also be used.

The default size for a REAL is 64 bits, but packed 32-bit reals can be defined by using `REAL(KIND=4)`, which may lead to performance improvements for codes requiring only single precision (or half precision as Cray now call it). 32-bit packed data types are also provided for INTEGER and LOGICAL, and packed 64-bit for COMPLEX.

The following Fortran 90 features are not supported in this release: modules, internal procedures, array constructors and declarations contining intrinsic functions.

2 Modules and scope rules

Modules and contained functions must be removed, which means that any variables accessed through module definitions must be provided in common blocks to all functions requiring them. This of course also means that definitions must be repeated wherever the common block is used, for instance with derived types. The same is true if derived types are passed through argument lists.

It is important to understand the difference between a 'definition' and a 'declaration'. In FORTRAN77, mainly because the language is static and has no derived types, declarations and definitions take place together. In Fortran 90 it is possible to have a declaration, e.g. a variable name in a common block or ALLOCATE statement, which associates some memory with the name, and definition, for instance of an array shape, or derived type separately. On the Cray T3D f90 declarations can be made global in scope via common blocks. There is however no mechanism for sharing definitions between program units which would normally be done via a module – hence all the problems described below. We return to this point in the conclusions.

A particularly tricky scenario occurs if interface blocks contain derived types. These must be re-declared, and are then treated as local in scope, and therefore different to the declarations in the calling (or called) routines. If you try to overload an operator or generic function this way it simply won't work.

For the same reason passing derived types as arguments is dangerous, unless a SEQUENCE statement is included to force contiguous storage both inside and outside the routine. The compiler is not guaranteed to do this.

If there are conflicts of name between a contained function and an external one then the names must be changed when the MODULE declarations are removed.

Of course PRIVATE and PUBLIC declarations are not very meaningful any more.

3 Procedure interfaces and pre-compiled libraries, e.g. BLAS, MPI

On the good side allocatable and automatic arrays and dynamic memory have been tested and found to work, as have pointers and array sections, which can now be passed as arguments and items in i/o lists. Interface blocks are however needed so that the compiler knows how to map an array section onto an array argument.

Some caution must be exercised if a Fortran 90 program is to use external procedures from a pre-compiled library in a 'foreign' language, e.g. FORTRAN77 or C for which no explicit interface is available. No explicit interface can be contructed because the programmer does not know the storage details inside the external routines, only that they expect to have contiguous storage with the starting address passed as a dummy argument.

3.1 Interfaces

Fortran 90 procedures can expect dummy arguments of many types; optional, assumed-shape, pointer, target, etc.

An explicit interface in Fortran 90 is required if:

1. A reference to a procedure appears:
   - with an argument keyword;
   - as a defined assignment (for subroutines);
   - in an expression as a defined operator (for functions);
   - as a reference by its generic name;
2. The procedure has:
   - an optional dummy argument;
   - an array-valued result (for functions);
   - a dummy argument that is a pointer, target, or an assumed-shape; array;
   - a result whose length type parameter is neither assumed nor constant (for character functions);
   - a pointer result (for functions);

Procedures included in the scope unit of a module have explicit interfaces produced automatically by the compiler. This also applies to all public procedures accessible through a USE statement. Other interfaces are implicit. Clearly because no modules are available on the Cray T3D explicit interfaces should be written wherever required by the above rules.

Because no explicit interface exists to an external 'foreign' procedure the compiler must make sure that storage is 'contiguous' (see below) before passing the starting address at the routine call, and similarly revert to the preferred storage after the call. The effect of this is quite clear if we consider passing an array section, or pointer to an array section – a local copy must be made twice (since we cannot even specify the INTENT of the argument). Likewise we cannot know if the external routine expects arguments to be passed as %VAL or %REF, so the address is used to remain compatible with FORTRAN77 where appropriate.

On Cray systems it appears that a local copy of dummy arguments is also made if a reference to a procedures appears:

- a dummy argument that is an assumed-shape array ;
- a dummy argument that is a pointer (must be declared as assumed shape);
- a dummy argument that is an array section;

3.2 BLAS, LAPACK and MPI

We tested calls to the SAXPY and CSSCAL subroutines from the Cray BLAS in libsci.a and the IBM BLAS in libesslp2.a, but conlusions carry over to any external procedures. On the Cray systems, T3D and J90, calls to BLAS with assumed-shape arrays or pointer arguments were respectively 10 and 5 times slower than using explicitly-dimensioned arrays for the vector lengths of sizes 55296 to 884736 tested. On the IBM RS/6000 systems no degredation in performance was seen.

On Cray systems therefore external procedures should not be called with arguments which are assumed-shape arrays or pointers. (The calling procedure should explictly dimension the array, which means it must be an automatic array, allocated array or explicit-shape array.) This probably means passing the actual dimensions into the procedure unless they are related to dimensions of some other know quantity in the first case.

At this point some comments can be made on programming practice which may be at variance with the FORTRAN77 standard ISO/IEC 1539-1980 [14], which remains a subset of Fortran 90. Section 15.9.3 describes association of dummy and actual arguments. It states that

a valid association occurs only if the type of the actual argument is the same as the type of the corresponding dummy argument.

This means that any program which passes a COMPLEX type and interprets it as a pair of REALs is erroneous, such as occurs for instance with a good many FFT routines. Clearly there is no hope of passing derived types to external foreign procedures!

Section 15.9.3.3 discusses arrays as dummy arguments. It states that

the number and size of dimensions in an actual argument may be different from the number and size of the dimensions in an associated dummy argument array declarator.

And ... *the size of the dummy argument array must not exceed the size of the actual argument array, and each actual argument array element becomes associated with the dummy array argument element that has the same subscript value as the actual array argument element.*

The simplest interpretation of this is that storage is contiguous, but that is not guaranteed and depends on how the compiler is written. It does however mean that, if arrays have different numbers of dimensions, an explicit interfaces **cannot** be written. Since actual dimensions must be given in such cases this is not at odds with the above rules specifying when an interface is required. For this reason interfaces **should not** be written for external 'foreign' procedures. This is erroneous and on Cray systems may lead to addressing exceptions.

Finally, we have carried out independent tests of the performance of array arithmetic using FORTRAN77, Fortran 90 and the vendors' BLAS library on both the T3D and SP2. These are being published in a separate report [25].

3.3 ScaLAPACK, PBLAS and BLACS, Cray parallel subroutines

The Cray T3D implementation of the PBLAS and BLACS which are called from ScaLAPACK (along with LAPACK and BLAS discussed above) internally use the low-level Cray-specific SHMEM direct processor to processor memory copy routines to do message passing. These routines do not compute remote memory addesses, but assume that an object (array or variable) declared on one processor will occupy the same physical memory locations on *all* processors. Such behaviour is of course *not* guaranteed if dynamic memory allocation of any form is used, and indeed the commonly recommended course of action is to include all such variables in a common block to force static allocation. The Cray compiler in fact fixes common block addresses to be the same on all processors because of the SPMD philosophy. Failure to comply with this recommendation will lead to disastrous consequences, usually ending in an operand range error and core dump!

These comments apply equally to all Cray parallel library routines which invoke SHMEM e.g. the parallel 2D and 3D FFTs which are widely used.

Although Fortran 90 ALLOCATE does not guarantee correct addressing, Cray provide their own routine `shpalloc()`, for SHMEM heap allocate, which does ensure this providing all processors call it concurrently with the same length arguments. Cray pointers rather must be used with this routine. The routine communicates to make addresses consistent and will hang if arguments are wrong or not all PEs participate in the call. Calling procedure is explained in the MPP man pages. A routine `shpdeallc()` is also provided so the heap can be re-used.

We have been able to test behaviour of some ScaLAPACK routines in separate applications and conclusions are presented separately [48, 21]. Performance of the Cray parallel 2D and 3D FFTs are also reported in [22].

4 A case study

So how can one port a Fortran 90 program to the T3D? A few hints are provided below.

The case study was extracted from a larger electron collision code written by Cliff Noble. This example uses dynamic memory management system routines packaged in the module `dyn_memry` in the file `dyn_memry.f90` to allocate and organise memory used subsequently.

The complete code would then proceed as follows:

```
! get memory for associated 3D sector Hamiltonian H matrix etc.
  get_r3mem('h_sect',d1,d2,d3)
! generate and propagate through each sector:
  DO i=1,nsect+1
    compute potential matrix V at asect(i)
    generate h_sect at asect(i)
    propogate R matrix from asect(i) to  asect(i+1)
  END DO
  output results
```

No changes to the simplified demo driver are needed, however the first changes are seen in `SUBROUTINE propagate()`. The statement `USE dyn_memry` has been replaced by an explicit interface block for `SUBROUTINE get_r1mem()`, the only one actually used in this example. In a real code a fairly extensive list might be involved, which could be put inline through an INCLUDE statement.

If we now turn to the dynamic memory module itself some more extensive re-coding has been required, shown only for `SUBROUTINE get_r1mem()`. Because it was not possible to initialise static variables in the MODULE, a BLOCK DATA routine had to be used with COMMON blocks. This is not very elegant. `COMMON/counters/` and `COMMON/hash_info/` are also added to the subroutines replacing the saved variables from the module. The `SAVE/counters/,/hash_info/` statement ensures that values are saved between routine calls. If this were not present the named common blocks would strictly have to be included higher in the calling tree unless a compile-time option were used to save everything.

The derived type definitions are no longer at the top of the MODULE, but must be placed in the non-executable statements of each routine, *e.g.* `SUBROUTINE add_mem_blk(), SUBROUTINE get_r1mem()` and the new INTERFACE block. Again this could be done conveniently with an include file. SEQUENCE statements have been added to the derived type definitions to ensure that the members are located contiguously in the computer's memory as noted above. We then know that, although the definitions are local, the compiler will pass a declared derived type by its start address and correct values will be accessed for all members.

We were lucky that in this example there is no operator overloading, However in the original code the compiler could select a method, such as `SUBROUTINE get_r1mem()` from a generic interface `SUBROUTINE get_mem()` according to the type of the pointer passed to it as its second argument. On the T3D this would only work with intrinsic types, such as `INTEGER, REAL*8` and would not work with a defined type such as `REAL_12` as in the example. This is inelegant as it means the correct routine must be called for each type of variable, undermining the utility of the package. **we think however SEQUENCE derived types may be OK**

5 Recommendations

With some obvious restrictions existing Fortran 90 codes can be edited to run on the Cray T3D, however a considerable effort would be involved for a large code.

We would recommend that new code development tarteged at this machine should, for the time being, be limited to language features which do not require modules. Most functionality should then work. If dynamically allocated objects are passed to procedures using SHMEM `shpalloc()` should be used. Alternatively codes can be developed on the IBM SP2 or another system with a full Fortran 90 compiler, pending access to a large-scale facility such as a Cray T3E which does support the standard.

Explicit interfaces cannot be provided for external 'foreign' procedures such as those in linkable libraries. We recommend that assumed-size arrays, pointers and array sections are **not** passed to such procedures as there is a performance penalty. Data types of arguments must exactly match those used in the procedures, so none SEQUENCE derived types cannot be passed, but components can be passed.

A tool LOFT90 [43] exists for converting existing standard FORTRAN77 code to Fortran 90. It is most useful to put in the new declaration syntax and to provide interface blocks. It also does some convertion to array syntax where simple loop structures can be recognised. This tool does not insert modules, because clearly that would require global data analysis which LOFT90 does not do, so a translated code should compile on the T3D.

6 Other compilers on Cray T3D

Whilst it is clear that the `f90` compiler on the T3D has serious limitations there are two HPF compilers installed which may permit more straightforward porting. Fortran 90 is a subset of HPF and standard codes should therefore compile. We have not yet done any benchmarks to compare performance of the three compilers, but these will be done shortly.

6.1 Portland Group PGHPF

This has been evaluated by Alistair Ewing at EPCC. There is a `man` page on the T3D.

6.2 Pacific Sierra VAST-HPF

This is currently being evaluated by Julian Corallo at DL.

7 Conclusions

The `f90` compiler v0.1.1 on the Cray T3D is only a subset of the Fortran 90 international standard ISO/IEC 1539-1991 or even the equivalent American standard ANSI X3.198-1992 [11]. The most obvious deviation is the lack of modules and consequent lack of scope in variable definitions. We have demonstrated some of the steps which must be taken in converting a moderately sophisticated package to run on this machine. There are however serious inadequacies, which would not always be so straightforward to overcome, and in any case require significant programmer effort of a kind not cheaply available and in any case better directed at other tasks such as code optimisation, or even science!.

We suggest that one simple improvement to the `f90` compilation system which would permit some headway for users would be to include a single module of global scope, similar to an un-named common block. This could be used to house definitions, such as of derived types and so avoid the main scope tragedy of the current system.

8 Availability of Fortran 90 compilers on other platforms.

This section just summarises the current availabilty of Fortran 90 products. Additional information on Fortran 90 products can be found on WWW with the URL *http://www.fortran.com/fortran*

Absoft is about to beta-test a version of Cray's f90 for the Power Macintosh. Windows NT and 95 versions will follow (fortran@absoft.com).

ACE of Holland provides F90 and HPF for Parsytec PowerPC-based machines contact *marco@ace.nl*. They also supply a range of compilers for SPARC processors.

Apogee's compiler is highly optimised for SPARC architectures *info@apogee.com*. Used on the Meiko CS-2HA.

Cray Research has a fully-optimising, native compiler, f90, that is being marketed by them, and by Visual Numerics for workstations, starting with Suns craysoft@cray.com.

Digital has DEC Fortran 90, a native, optimising compiler for Digital UNIX on Alpha systems (with HPF), and for OpenVMS Alpha (without HPF). A Windows NT (Alpha) version is likely to follow *Fortran 90@digital.com* or *http://www.digital.com/info/hpc/f90*.

Edinburgh Portable Compilers has optimising, native compilers for x86, Sun, RS/6000, SGI and MIPS *info@epc.com* or *support@epc.ed.ac.uk*.

Fujitsu is marketing a native Fortran 90 Workbench for Solaris 1.1 and 2.x. Contact Unicomp *walt@fortran.com* or Fujitsu *info@ossi.com*.

HP has stated its intention to collaborate with EPC to produce a compiler for HP and Convex platforms, timescale not yet announced.

IBM has been shipping its optimising, native compiler for the RS/6000, xlf Version 3, as of 31 December, 1993. HPF is available with AIX v4.1

Lahey has been shipping a native LF90 compiler for DOS since 29 August, 1994 *sales@lahey.com*. It is particularly well optimised on the Pentium. Also on offer is elf90, a subset language that does not allow storage association and is very cheap.

Microsoft is working on a compiler, now in beta test, for Windows NT 3.5 and Windows 95 (Chicago) *fortran@microsoft.com*.

Microway NDP Fortran 90 for 386/486, Pentium and 860 is available *nina@microway.com*.

A compiler is available for most unix platforms, VMS and PCs (including Linux) from NAG *infodesk@nag.com*, *infodesk@nag.co.uk* or URL *http://www.nag.co.uk/*. This was the first F90 compiler, in 1991. An optimising version is to be produced in collaboration with ACE (see above). FORESYS 90, to convert from F77 source to F90, is available too.

N.A. Software supplies Fortran 90 Plus on 386/486, SPARC and T800 and T9000 transputers *marketing@nasoftwr.demon.co.uk*. They also supply a F77 to F90 syntax convertor, LOFT90, and also HPF.

VAST-F90 from PSR *info@psrv.com* is a complete F90 compiler, including a vectoriser, for UNIX, VMS and Convex. PSR also supplies VAST-77to90, to convert FORTRAN 77 programs into Fortran 90 syntax, as well as HPF.

A compiler from ParaSoft *f90-info@parasoft.com* is available.

PGI has released a subset Fortran 90/HPF compiler, in particular for SGI *sales@pgroup.com* or *http://www.pgroup.com/*.

A PC version of the NAG compiler is also available from Salford Software *sales@salfsoft.demon.co.uk*. A Windows NT version is planned.

SGI recently announced their F90 compiler

SofTech has a licence to sell its own versions of DEC's HPF/F90 compiler.

Sun has released an F90 compiler based on Cray's f90, initially for Solaris 2 URL *http://www.sun.com/sunsoft/Products/Developer-products*.

8.1 Other useful tools

FORGE90 and an HPF processor from APR *support@apri.com* or *http://www.infomall.org/apri/* are available.

A source form convertor, convert.f90, is obtainable by ftp from jkr.cc.rl.ac.uk in the directory /pub/MandR.

Both NAG (see above) and IMSL now Visual Numerics, *mktg@houston.vni.com* are offering F90 versions of their maths libraries that take full advantage of the language's library building capabilities.

f90split.f is a free program available from *Michel.Olagnon@ifremer.fr*. It splits a free-form F90 source code into separate files.

There are also available free codes for producing makefile stubs containing module dependencies.

Files for a Fortran 90 emacs mode can be retrieved by ftp to *ftp://mailbase.ac.uk/pub/lists/comp-fortran-90/files/* or by Gopher and WWW, URL *gopher://nisp.ncl.ac.uk/11/lists-a-e/comp-fortran-90/files* or by sending an e-mail to *mailbase@mailbase.ac.uk*, containing the command send ¡listname¿ ¡filename¿ (e.g. send comp-fortran-90 f90.el).

9 F90 example – original Fortran 90 version

Makefile

```
PROG= dem
SRCS= demo.f90 dyn_memry.f90
OBJS=demo.o dyn_memry.o

FC=xlf90
FFLAGS=-g -F:f90 -qlanglvl=90PURE -qcheck
LDFLAGS=-g -C -qcheck
```

```
all: $(PROG)

$(PROG): $(OBJS)
$(FC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

clean:
rm -f $(PROG) $(OBJS) *.mod

.SUFFIXES: $(SUFFIXES) .f90

.f90.o:
$(FC) $(FFLAGS) -c $<

demo.o: dyn_memry.o
```

demo.f90

```
PROGRAM demo ! original f90 program
  IMPLICIT NONE
  INTEGER, PARAMETER :: knd=SELECTED_REAL_KIND(12)
  REAL(knd)          :: rfirst,rlast,step_size
  rfirst=10.0_knd
  rlast=20.0_knd
  WRITE(*,'(a,f12.6)')'Input step_size : must be a divisor of ',rlast-rfirst
  READ *, step_size
! check validity of input
  IF(MOD(INT(rlast-rfirst),INT(step_size))/=0)THEN
    WRITE(*,'(f12.6,a,f12.6)')step_size, ' is not a divisor of ',rlast-rfirst
    STOP
  END IF
! propagate simulation
  CALL propagate(rfirst,rlast,step_size) ! complete code would contain
                                         ! matrix parameters, energies etc
END PROGRAM

SUBROUTINE propagate(rfirst,rlast,step_size)
! propagate the r-matrices through the sectors
  USE dyn_memry     ! use the module from 'dyn_memry.f90'
  IMPLICIT none
  INTEGER, PARAMETER :: knd=selected_real_kind(12)
  REAL(knd)               :: rfirst, rlast, step_size
  REAL(knd), pointer      :: asect(:)
  INTEGER                 :: nsect,i
  REAL(knd)               :: w_sect
  nsect=INT((rlast-rfirst)/step_size)  ! no sectors
  w_sect=(rlast-rfirst)/real(nsect,knd)   ! accurate sector width
```

```
  CALL get_r1mem('asect',asect,nsect+1)
! fill asect
  DO i=0,nsect
    asect(i+1)=real(i,knd)*step_size+rfirst
  END DO
  DO i=1,nsect+1
    WRITE(*,'(i4,5x,a,f12.6)')i, 'asect=', asect(i)
  END DO
  ...
END SUBROUTINE propagate
```

dyn_memry.f90

```
MODULE dyn_memry
  IMPLICIT NONE
  INTEGER, PARAMETER, PUBLIC :: max_char=16  ! length of name component
  INTEGER, PARAMETER         :: HASHSIZE=100 ! hash table size
  INTEGER, PARAMETER :: k=selected_real_kind(12) ! real kind value
! derived data types
  TYPE mem_blk
     CHARACTER(max_char)      :: name    ! name of variable
     REAL(K), pointer         :: r1(:) ! real data
     INTEGER                  :: d1 ! dimension
  END TYPE mem_blk
  TYPE hash_tab                      ! deferred shape array
     type(mem_blk), pointer   :: p
  END TYPE hash_tab
  INTEGER, SAVE             :: chash=1            ! current hash value
  TYPE(hash_tab), SAVE      :: h_tab(HASHSIZE)  ! hash table
  INTEGER, SAVE   :: r_tot=0, alloc_tot=0  ! object counts

CONTAINS

  SUBROUTINE add_mem_blk(ablk, name, d1)
! add new memory block to hash table
    TYPE(mem_blk), POINTER   :: ablk       ! first link
    CHARACTER(*), INTENT(IN) :: name       ! block name
    INTEGER, INTENT(IN)      :: d1         ! dimension information
    TYPE(mem_blk), POINTER   :: new_entry
    INTEGER                  :: status, dd
! allocate new_entry of data type mem_blk
    ALLOCATE(new_entry, stat=status)
    IF(status/=0)THEN
    WRITE(*,*)' add_mem_blk: memory allocation error ', status
    STOP
    END IF
    alloc_tot=alloc_tot+1
```

```
    new_entry%name=name
    nullify(new_entry%r1)
! allocate real vector space
    ALLOCATE(new_entry%r1(d1), stat=status)
    IF(status/=0)THEN
      WRITE(*,*)' add_mem_blk: real vector allocation error ', status
      STOP
    END IF
    r_tot=r_tot+d1
    new_entry%d1=d1
    ablk=>new_entry
    h_tab(chash)%p=>new_entry
  END SUBROUTINE add_mem_blk

  SUBROUTINE get_r1mem(name, r, d1)
! allocate memory block name of real type
    IMPLICIT NONE
    CHARACTER(*), INTENT(IN) :: name
    REAL(k), POINTER         :: r(:),rp(:)
    INTEGER, INTENT(IN)      :: d1
    IF(LEN(name)==0)THEN
      WRITE(*,*)' get_r1mem: input error, name=', name
      STOP
    END IF
    IF(d1<=0)THEN
      WRITE(*,*)' get_r1mem: input error, d1=', d1, ' name=', name
      STOP
    END IF
    CALL add_mem_blk(h_tab(chash)%p, name, d1)
    rp=>h_tab(chash)%p%r1
    r=>rp(2:d1+1)
    chash=chash+1
  END SUBROUTINE get_r1mem

END MODULE dyn_memry
```

10 F90 example – Cray T3D version

Makefile

```
PROG=demo
SRCS=demo.f90 dyn_memry.f90
OBJS=demo.o dyn_memry.o

FC=f90
```

```
FFLAGS=-c -dp
LDFLAGS=-Xl

$(PROG): $(OBJS)
$(FC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

clean:
rm -f $(PROG) $(OBJS) *.mod

.SUFFIXES: $(SUFFIXES) .f90

.f90.o:
$(FC) $(FFLAGS) $<

demo.o : dyn_memry.o
```

demo.f90

```
PROGRAM demo ! no changes were needed to the demo driver
...
END PROGRAM

SUBROUTINE propagate(rfirst,rlast,step_size)
! propagate the r-matrices through the sectors
! (in this limited example just allocate some memory for a real
! vector and fill with step points).
  IMPLICIT NONE
  INTERFACE
    ! interface to subroutine get_r1mem
    SUBROUTINE get_r1mem(name,r,d1)
      INTEGER, PARAMETER :: k=SELECTED_REAL_KIND(12) ! real kind value
      CHARACTER(*), INTENT(IN) :: name
      REAL(k), POINTER         :: r(:)
      INTEGER, INTENT(IN)      :: d1
    END SUBROUTINE get_r1mem
  END INTERFACE

  INTEGER, PARAMETER :: knd=SELECTED_REAL_KIND(12)
  REAL(knd)               :: rfirst, rlast, step_size
  REAL(knd), POINTER      :: asect(:)
  INTEGER                 :: nsect,i
  REAL(knd)               :: w_sect
  nsect=INT((rlast-rfirst)/step_size)  ! no. sectors
  w_sect=(rlast-rfirst)/real(nsect,knd)   ! accurate sector width
! assign some memory for integer vector asect
  CALL get_r1mem('asect',asect,nsect+1)
! fill asect
```

```
  DO i=0,nsect
    asect(i+1)=real(i,knd)*step_size+rfirst
  END DO
! output asect
  DO i=1,nsect+1
    WRITE(*,'(i4,5x,a,f12.6)')i, 'asect=', asect(i)
  END DO
  ...
END SUBROUTINE propagate
```

dyn_memry.f90

```
BLOCK DATA
  IMPLICIT NONE
  INTEGER               :: r_tot,alloc_tot,chash
  COMMON/counters/r_tot,alloc_tot,chash
  SAVE/counters/,/hash_info/
  DATA r_tot,alloc_tot,chash/0,0,1/  ! initialize counters to zero
END

SUBROUTINE add_mem_blk(ablk, name, d1)
! add new memory block to hash table
  IMPLICIT NONE
  INTEGER, PARAMETER :: k=SELECTED_REAL_KIND(12) ! real kind value
  INTEGER, PARAMETER :: max_char=16
  INTEGER, PARAMETER :: HASHSIZE=100
! derived type definitions
  TYPE mem_blk
    SEQUENCE                 ! sequence attribute now needed
    CHARACTER(max_char)     :: name    ! name of variable
    REAL(k), POINTER        :: r1(:) ! real data
    INTEGER                 :: d1 ! dimension
  END TYPE mem_blk
  TYPE hash_tab                     ! deferred shape array
    SEQUENCE
    TYPE(mem_blk), POINTER   :: p
  END TYPE hash_tab
  TYPE(mem_blk), POINTER   :: ablk       ! first link
  CHARACTER(*), INTENT(IN) :: name       ! block name
  INTEGER, INTENT(IN)      :: d1         ! dimension information
  TYPE(mem_blk), POINTER   :: new_entry
  INTEGER                  :: status, dd
  INTEGER                  :: r_tot,alloc_tot,chash
  TYPE(hash_tab)           :: h_tab(HASHSIZE)  ! hash table
  COMMON/counters/r_tot,alloc_tot,chash
  COMMON/hash_info/h_tab
  SAVE/counters/,/hash_info/
```

```
! allocate new entry of data type mem_blk
  ALLOCATE(new_entry, stat=status)
  IF(status/=0)THEN
     WRITE(*,*)' add_mem_blk: memory allocation error ', status
  STOP
  END IF
  alloc_tot=alloc_tot+1
  new_entry%name=name
  nullify(new_entry%r1)
! allocate real vector space
  ALLOCATE(new_entry%r1(d1), stat=status)
  IF(status/=0)THEN
    WRITE(*,*)' add_mem_blk: real vector allocation error ', status
  STOP
  END IF
  r_tot=r_tot+d1
  new_entry%d1=d1
  ablk=>new_entry
  h_tab(chash)%p=>new_entry
END SUBROUTINE add_mem_blk

SUBROUTINE get_r1mem(name, r, d1)
! allocate memory block name of real type
  IMPLICIT NONE
  INTERFACE
! interface body for external subroutine add_mem_blk
    SUBROUTINE add_mem_blk(ablk,name,d1)
      IMPLICIT NONE
      INTEGER, PARAMETER :: k=SELECTED_REAL_KIND(12) ! real kind value
      INTEGER, PARAMETER :: max_char=16
! derived type definitions
      TYPE mem_blk
        SEQUENCE                  ! sequence attribute now needed
        CHARACTER(max_char)      :: name    ! name of variable
        REAL(k), POINTER         :: r1(:) ! real data
        INTEGER                  :: d1 ! dimension
      END TYPE mem_blk
      TYPE(mem_blk), POINTER   :: ablk       ! first link
      CHARACTER(*), INTENT(IN) :: name       ! block name
      INTEGER, INTENT(IN)      :: d1         ! dimension information
    END SUBROUTINE add_mem_blk
  END INTERFACE
  INTEGER, PARAMETER :: k=SELECTED_REAL_KIND(12) ! real kind value
  INTEGER, PARAMETER :: max_char=16
  INTEGER, PARAMETER :: HASHSIZE=100
! derived type definitions
  TYPE mem_blk
    SEQUENCE                  ! sequence attribute now needed
```

```
    CHARACTER(max_char)      :: name    ! name of variable
    REAL(k), POINTER         :: r1(:) ! real data
    INTEGER                  :: d1 ! dimension
  END TYPE mem_blk
  TYPE hash_tab                        ! deferred shape array
    SEQUENCE
    TYPE(mem_blk), POINTER   :: p
  END TYPE hash_tab
  CHARACTER(*), INTENT(IN) :: name
  REAL(k), POINTER         :: r(:),rp(:)
  INTEGER, INTENT(IN)      :: d1
  INTEGER                  :: r_tot,alloc_tot,chash
  COMMON/counters/r_tot,alloc_tot,chash
  TYPE(hash_tab)           :: h_tab(HASHSIZE)  ! hash table
  COMMON/hash_info/h_tab
  SAVE/counters/,/hash_info/
  IF(LEN(name)==0)THEN
    WRITE(*,*)' get_r1mem: input error, name=', name
    STOP
  END IF
  IF(d1<=0)THEN
    WRITE(*,*)' get_r1mem: input error, d1=', d1, ' name=', name
    STOP
  END IF
  CALL add_mem_blk(h_tab(chash)%p, name, d1)
  rp=>h_tab(chash)%p%r1
  r=>rp(2:d1+1)
  chash=chash+1
END SUBROUTINE get_r1mem
```

Some other examples will follow in a future edition of this report

References

[1] *Fortran-90 course* Manchester Computer Centre: *http://www.hpctec.mcc.ac.uk/hpctec/courses/Fortran 90/F90course.html*

[2] *Fortran 90 Course* The University of Liverpool: *http://www.liv.ac.uk/HPC/F90page.html*

[3] *Fortran 90 Course* CERN *http://asis01.cern.ch/CN/CNTUT/f90/Overview.html*

[4] *Fortran 90 Course* Edinburgh Parallel Computer Centre *http://www.epcc.ed.ac.uk/epcc-tec/course-packages/HPF-Package-form.html*

[5] *Fortran 90 course* contact PSR (see above);

[6] *Fortran 90 or HPF course* CETech, Inc. 8196 SW Hall Blvd., Ste. 304, Beaverton, Oregon 97008, USA. *cetech@teleport.com*

[7] *Fortran 90 course* IT Independent Training Limited, 2 Windlebrook Green, Bracknell, Berkshire, UK tel. +44 1344 860172 fax. +44 1344 867992

[8] *Fortran 90 course* Simulog, attn. Mr. E. Plestan, 1 rue James Joule, F-78286 Guyancourt Cedex, France tel: +33 1 30 12 27 00 fax: +33 1 30 12 27 27

[9] *Fortran 90 course* CTS, Prinz-Otto Str. 7c, D-85521 Ottobrunn , Germany tel: +49-89-6083758 fax: +49-89-6083758

[10] *ISO/IEC 1539:1991 standard* obtainable for 185 Swiss francs from ISO Publications, 1 rue de Varembe, Case postale 56 CH-1211 Geneva 20, Switzerland Fax. + 41 22 734 10 79

[11] *ANSI X3.198-1992 standard* from ANSI, 1430 Broadway, New York, N.Y. 10018 or in electronic PostScript or ASCII form from Unicomp at a cost and under conditions agreed by ISO. *walt@fortran.com*

[12] A Corrigendum 1 was published by ISO on 1 September, 1994, and is available from them (cost about 30 Swiss francs).

[13] American National Standard X3.9-1978 *Programming language FORTRAN*

[14] ISO/IEC standard 1539-1980

[15] A Russian translation of the standard (translator S.G.Drobyshevich) is available from the editor, Alla Gorelik *gorelik@applmat.msk.su*

[16] *AIX XL Fortran Compiler/6000 Language Reference version3 release 1* (IBM, 1990.1993) system manual SC09-1611-00

[17] Adams, Brainerd, Martin, Smith and Wagener *Fortran 90 Handbook* (McGraw-Hill, 1992) ISBN 0-07-000406-4.

[18] Adams, Brainerd, Martin and Smith *Fortran Top 90-Ninety Key Features of Fortran 90* (Unicomp, 1994) ISBN 0-9640135-0-9.

[19] Aberti *Fortran 90; Initiation à partir du Fortran 77* (Série Informatique Editions, Menton, 1992) ISBN 2-090615-00-6.

[20] M.Ain *Savez-vous parler Fortran* (Bibliotheque des universites (de Boeck), 1994) ISBN 2-8041-1755-3

[21] R.J.Allan and I.J.Bush *Parallel Application Software on High Performance Computers: Parallel DIagonalisation Routines* (Daresbury Laboratory, 1996)

[22] R.J.Allan and I.J.Bush *Parallel Application Software on High Performance Computers: Serial and Parallel FFT routines* (Daresbury Laboratory, 1996)

[23] W.Brainerd *Fortran 90 course* contact *walt@fortran.com*

[24] Brainerd, Goldberg and Adams *Programmer's Guide to Fortran 90, second edition* (Unicomp, 1994)

[25] I.J.Bush, R.J. Allan and P.Walmsley *Parallel Application Software on High Performance Computers: Linear Algebra Routines* (Daresbury Laboratory, 1996) in preparation

[26] I.Chivers and J.Slightholme *Introducing Fortran 90* (Springer Verlag, London, 1995) ISBN 3-540-19940-3

[27] P.Corde and H.Delouis *Support de cours Fortran 90 IDRIS ftp://ftp.ifremer.fr/pub/ifremer/Fortran 90/f90_cours_4.ps.gz.*

[28] Counihan *Fortran 90* (Pitman, 1991) ISBN 0-273-03073-6.

[29] C.Delannoy *Programmer en Fortran 90* (Eyrolles, 1992) ISBN 2-212-08723-3

[30] M.Dubesset and J.Vignes *Les specificites du Fortran 90* (Editions Technip, 1993) ISBN 2-7108-0652-5

[31] Ellis, Philips and Lahey *Fortran 90 Programming* (Addison Wesley, Wokingham) 1994, ISBN 0-201-54446-6.

[32] W.Gehrke *Fortran 90 Referenz-Handbuch: der neue Fortran-Standard* (Carl Hansen Verlag, 1991) ISBN 3-446163-21-2.

[33] W.Gehrke *Fortran 90 Language Guide* (Springer-Verlag, London, 1995) ISBN 3-540-19926-8

[34] B.D.Hahn *Fortran 90 for Scientists and Engineers* (Edward Arnold, 1994) ISBN 0-340-60034-9.

[35] Heisterkamp *Fortran 90: eine informelle Einführung* (BI-Wissenschaftsverlag, 1991) ISBN 3-411153-21-0.

[36] J.F.Kerrigan *Migrating to Fortran 90* (O'Reilly Associates, 1993) ISBN 1-56592-049-X.

[37] P.Lignelet *Fortran 90; Approche par la Pratique* (Série Informatique Editions, Menton, 1993) ISBN 2-090615-01-4.

[38] P.Lignelet *Manuel complet du langage Fortran 90, et guide d'application* (S.I. editions, Jan. 1995) ISBN 2-909615-02-2

[39] Meissner *Fortran 90* (PWS Kent, Boston, 1995) ISBN 0-534-93372-6.

[40] M.Metcalf and J.Reid *Fortran 90 Explained* (Oxford Science Publications, 1990) ISBN 0-19-853772-7

[41] T.Michel *Fortran 90 Lehr- und Handbuch* (BI-Wissenschaftsverlag, 1994)

[42] Morgan and Schonfelder *Programming in Fortran 90* (Alfred Waller, Oxfordshire, 1993) ISBN 1-872474-06-3.

[43] N.A.Software *LOFT90 source code translator* from N.A.Software Limited, Roscoe House, 62 Roscoe Street, Liverpool L1 9DW.

[44] Ortega and Saunders *Introduction to Fortran 90 for Scientific Computing* (College Publishing, 1994) ISBN 0-030010198-0.

[45] Schick and Silverman *Fortran 90 and Engineering Computation* (John Wiley, 1994) ISBN 0-471-58512-2.

[46] Schobert *Programmierung in Fortran 90* (Oldenburg, 1991)

[47] I.M.Smith *Programming in Fortran 90* (Wiley) ISBN 0471-94185-9.

[48] A.Sunderland, J.Heggarty, C.J.Noble and R.J.Allan *Parallelisation of the R-matrix propagator code on the Cray T3D* (Daresbury Laboratory, 1996) in preparation

[49] Überhuber and Meditz *Software Entwicklung in Fortran 90* (Springer Verlag, 1993) ISBN 0-387-82450-2.

[50] B.Wojcieszynski and R.Wojcieszynski *Fortran 90* (Addison-Wesley, 1993) ISBN 3-89319-600-5.

[51] He Xingui, Xu Zuyuan, Wu Qingbao and Chen Mingyuan *Programming Language Fortran 90* (China Railway Publishing House, Beijing 1994) ISBN 7-113-01788-6/TP.187