HP Fortran for Tru64 UNIX Systems

Release Notes

September 2005

These release notes contain information about HP Fortran for HP Tru64 UNIX Alpha systems.

Software Version: HP Fortran Version 5.6

Hewlett-Packard Company
Palo Alto, California

Printed in the US

This document was prepared using DECdocument, Version 3.3-1b.

Contents

4 New Features for Versions Prior to Version 5

1

HP Fortran Version 5.6 Release Notes

This chapter contains the following sections:

- Section 1.1, Overview
- Section 1.2, Installation and Minimum Operating Systems Version
- Section 1.3, Contents of the Version 5.6 Kit
- Section 1.4, Known Problems in Version 5.6
- Section 1.5, Bug Fixes in Version 5.6

1.1 Overview

HP Fortran conforms to the Fortran 95 Standard, Fortran 90 Standard, and previous Fortran standards. It also includes support for High Performance Fortran (HPF), and contains many but not all of HP Fortran 77's extensions to the FORTRAN-77 standard. Except in rare instances, a valid FORTRAN-77 program is also a valid Fortran 95 program.

HP Fortran fully supports the Fortran 95 Standard (ISO/IEC 1539-1:1997(E)) and the multi-vendor OpenMP Fortran Specification, including support for directed parallel processing using OpenMP directives on shared memory multiprocessor systems.

The *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems* contains a detailed description of HP Fortran 77 source language compatibility. Provided the types and shapes of the actual and formal arguments agree, routines compiled with HP Fortran 77 can call (or be called by) routines compiled with HP Fortran.

1.2 Installation and Minimum Operating Systems Version

HP Fortran Version 5.6 requires Version 4.0F (or later) of the HP Tru64 UNIX operating system.

For a detailed description of the installation procedure, see the *HP Fortran Installation Guide for Tru64 UNIX Systems*.

You can also send comments, questions and suggestions about the HP Fortran product to the following mail address:

`fortran@hp.com`

Please note that this address is for informational inquiries and is not a formal support channel.

The HP Fortran home page is located at:

`http://www.hp.com/go/fortran/`

Click on “Fortran for Tru64 UNIX Alpha” for information about online documentation, software patch kits, example programs, and additional product information.

1.3 Contents of the Version 5.6 Kit

The HP Fortran Version 5.6 kit consists of the following setld sets:

- DFABASE560—HP Fortran 90 and 77 V5.6 for HP Tru64 UNIX Alpha Systems
- DFADOC560 —HP Fortran V5.6 Release Notes and Man Page
- DFACOM560 —HP Fortran V5.6 Tools & their Man Pages
- DFARTL406—HP Fortran RTL #406 for HP Tru64 UNIX Alpha Systems
- HPFLIBS192—HP Fortran V1.9-2 High Performance Fortran Runtime Libraries
- XMDCOM520 —the CXML common subset files
- XMDLIB4520—the CXML archive and shared libraries (serial and parallel) for EV4 systems
- XMDLIB5520—the CXML archive and shared libraries (serial and parallel) for EV5 systems
- XMDLIB6520—the CXML archive and shared libraries (serial and parallel) for EV6 systems
- XMDSCI520 —the SCIPORT Cray compatibility library and manpages
- XMDMAN520 —the CXML manpages
- XMDHTM520 —the CXML manpages in HTML format

The OTABASEnnn—Compiled Code Support Library subset is no longer included on this kit.

The HP Fortran DFABASE and DFACOM subsets include the HP Fortran 95/90 and HP Fortran 77 compilers and associated documentation. The DFADOC subset contains the compiler command manpages and release notes.

The XMD subsets contain the CXML routines, included in the HP Fortran kit as external routines.

CXML is distributed as a Development kit that allows you to link an application program with the CXML library and then run the executable image. No license is required. The CXML subsets are independent from one another, except that the user must link in the CXML library (either serial or parallel) after linking with the SCIPORT library.

For information including disk space requirements, see the *HP Fortran Installation Guide for Tru64 UNIX Systems*.

If you need to delete an older version of the Fortran Run-Time Library, delete the older subset before you install a new version. If you have installed multiple versions of the Run-Time Library and you delete one, you must reinstall the desired Run-Time Library version before you can correctly link Fortran programs.

The HP Fortran kit no longer includes the OTABASE subset (Compiled Code Support Library). The OTA libraries are now available only on the web from the following location:

`ftp://ftp.compaq.com/pub/products/fortran/Tru64/OTABASE221.tar`

This is the latest parallel processing library subset. Use `setld -l` to install.

Note

You may receive a message saying that the page cannot be displayed. However, the file is there, and you might have to access the file when the site is less congested.

The following changes occurred to the OSFCMPLRS operating system subset starting with Tru64 UNIX Version 4.0:

- Beginning with Version 4.0, the OSFCMPLRS subset now consists of multiple subsets: OSFCMPLRS (required for program development), OSFSDE (profiling tools), OSFLIBA (archive libraries), OSFINCLUDE (include files), OSFATOM (`atom` tools).

1.4 Known Problems in Version 5.6

Modules cannot be debugged unless the current Ladebug (which is Version 4.0-069) is installed. For information, see the Ladebug web page:

`http://www.hp.com/go/ladebug`

1.5 Bug Fixes in Version 5.6

This release fixed a number of LOC-related problems, and corrected and enhanced module debugging capability.

The following bugs were fixed:

- 104652: Remove an erroneous warning regarding multiple initializations which occurs during compilation with the option -std95.
- QXCM1000242932: Fix an infinite loop in the module-loading phase of compilation.
- 104654: Fix incorrect handling of a pointer field when copying a record, which resulted in slow compilation, culminating with an “Insufficent virtual memory” failure.
- Bug2878: FPE on system( ) call that goes away with -arch ev67.
- Bug2961: Change to default to `-assume cc_omp` when `-omp` is present.
- BUG2973: allowing `!$` conditional lines to compile.
- Bug3264: Data corruption/namespace collision at `-O2` and above.
- Bug3279: Assignment of allocatable array of records problem.
- Bug3292: PTR 221-1-3023. Assertion when making a DO variable private with `-omp` option.
- Bug2794: Allow STATIC on COMMON data (scalar or or array); allow INTRINSIC in BLOCK DATA.

- Bug3348: Fix inconsistency in adding generic nodes. Code was comparing declaring module with field from decl_rec_desc, but when adding node, using as declaring module the module form the 'use' statement. As a result matches were not found; multiple entries were made. Memory was exhausted and compilation time was outrageous.
- PTR 221-2-1067: QXCM1000208533: Compile-time failure for a function using some of its arguments to affect the size/shape of the return value for the function itself.
- QXCM1000214854: F90-F-FATAL: ICE when compiling program with a record named OR when there is a subsequent use of the .OR. operand.
- QXCM1000218797: Compilation with `-omp` option does not recognize "share" clause for array.
- QXCM1000221329: ACTPOS processing causes bad address in executable.
- QXCM1000214653: MCNP does not compile; internal compiler errors, OpenMP err. Don't diagnose common variables that are not explicitly attributed as shared, private, reduction, firstprivate, lastprivate if the common block is so attributed.
- QXCM1000205813: Allocate of allocatable components results in a segmentation violation at run time.
- QXCM1000221087: Public components of an object are inaccessible when the type name of that object is inaccessible.
- QXCM1000222701: Module procedure confused with nonstandard intrinsic FREE.
- QXCM1000235122: Incorrectly diagnosing common variables that are not explicitly attributed as shared, private, reduction, firstprivate, lastprivate if the common block is so attributed.

2
Documentation Information

The sections in this chapter:

- Describe HP Fortran documentation and online information (Section 2.1)
- Describe the main Compaq Parallel Software Environment documents (Section 2.2)
- List other sources of information about Fortran 95/90 (Section 2.3)

The HP Fortran Web page is at:

`http://www.hp.com/go/fortran`

Click on "Fortran for Tru64 UNIX Alpha" for information about online documentation, software patch kits, example programs, and additional product information.

2.1 HP Fortran Documentation and Online Information

The HP Fortran for Tru64 UNIX documentation is available at:

`http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,7155,00.html#dfau`

The HP Fortran documentation set includes the following:

- *HP Fortran Installation Guide for Tru64 UNIX Systems*

  Explains how to install HP Fortran (HP Fortran and HP Fortran 77) on an HP Tru64 UNIX Alpha system, including requirements.

  The installation guide is included with the HP Fortran kit.

- *Compaq Fortran Language Reference Manual* (AA–Q66SD–TK)

  Describes the HP Fortran source language for reference purposes, including the format and use of statements, intrinsic procedures, and other language elements. It also provides an overview of new Fortran 95/90 features (not available in FORTRAN-77).

  It identifies extensions to the Fortran 95 standard by blue-green color in the printed and HTML forms of this document.

  The *Compaq Fortran Language Reference Manual* is included with the Compaq Fortran (95/90) document kit, QA-MV2AA-GZ and is available on the Online Documentation Library CD-ROM in HTML form.

  The *Compaq Fortran Language Reference Manual* has been translated into Japanese and is available (see the HP Fortran Web site).

- *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems* (AA–Q66TE–TE)

Describes the HP Fortran program development and run-time environment on Tru64 UNIX Alpha systems. It describes compiling, linking, running, and debugging HP Fortran programs, performance guidelines, run-time I/O and error-handling support, data types, numeric data conversion, calling other procedures and library routines, and compatibility with HP Fortran 77. It provides information common to HP Fortran and the Compaq Parallel Software Environment as well as information about using directed parallel processing using OpenMP and HP Fortran directives.

The printed version of this document is included with the Compaq Fortran (95/90) document kit, QA-MV2AA-GZ and is on the Online Documentation Library CD-ROM in HTML form.

The HP Fortran Software Product Description (SPD) is provided as a file on the Software Product Library CD-ROM (media CD-ROM).

The following HP Fortran online information is available (once installed on the system):

- HP Fortran online reference pages (man pages)

  Describe the HP Fortran software components, including `f90`(1), `fpr`(1), `fsplit`(1), `intro`(3f), numerous Fortran library routines listed in `intro`(3f), and numerous parallel High Performance Fortran library routines listed in `intro`(3hpf).

- HP Fortran online release notes

  Provide more information on this version of HP Fortran, including known problems and a summary of the HP Fortran run-time error messages. These release notes are also provided on the Software Product Library CD-ROM (media CD-ROM).

  Once installed, the online release notes are located in:

  ```
  /usr/lib/cmplrs/fort90/relnotes90
  ```

  To view this file, use the `more` command (or `view` or similar command) on a system where HP Fortran is installed:

  ```
  % more /usr/lib/cmplrs/fort90/relnotes90
  ```

  To initiate a search within `more`, enter a slash (`/`) followed by the appropriate topic. For information about using the `more` command, see `more`(1).

- HP Fortran online help file

  This ASCII file provides online access to HP Fortran information, which includes error message descriptions, a summary of the language elements (statements, intrinsic functions, and so on), a glossary, and other information. The HP Fortran help file is located in:

  ```
  /usr/lib/cmplrs/fort90/decfortran90.hlp
  ```

  Use the `more` command or the `view` command to access the information available in this file. This help file is large and is not usually printed on a printer or read sequentially.

The HP Fortran for Tru64 UNIX documentation is available at:

```
http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/
1,1701,7155,00.html#dfau
```

2.2 Compaq HPF and Parallel Software Environment Documentation

The Compaq Parallel Software Environment product is no longer supported. The DIGITAL High Performance Fortran 90 HPF and PSE Manual (in the Compaq Parallel Software Environment documentation kit, QA-2ATAA-GZ), however, still contains important HPF documentation in Chapters 1-8. (The *Parallel Processing on Tru64 UNIX Systems* manual that describes NUMA parallel processing in Compaq Fortran is available after installation in the following location:

```
/usr/lib/cmplrs/fort90/fort55_parallel_manual.ps
```

The *DIGITAL High Performance Fortran 90 HPF and PSE Manual* explains both the (now retired) Parallel Software Environment (PSE) and the (still current) High Performance Fortran (HPF) programming language. It contains a tutorial describing how to write programs using the HPF extensions to Compaq Fortran.

2.3 Other Sources of Information About Fortran 95/90

This section lists sources of information about Fortran 95/90 other than the HP Fortran documentation.

The following publication is the copywritten standard for Fortran 90 and 95:

- American National Standard Programming Language Fortran 90, ANSI X3.198-1991, and International Standards Organization Programming Language standard ISO/IEC 1539:1991. (Simply referred to in documentation as the "Fortran 90 Standard".)
- American National Standard Programming Language Fortran 95, X3J3/96-007, and International Standards Organization Programming Language standard ISO/IEC 1539-1:1996. (Simply referred to in documentation as the "Fortran 95 Standard".)

Tutorial information about the Fortran 95/90 language is available in commercially published documents at major book stores or from their publishers. HP Fortran documentation does not usually provide such tutorial information. The following commercially published documents (listed in alphabetical order by title) in English provide reference or tutorial information about Fortran 90:

- *Fortran 90 Explained* by M. Metcalf and J. Reid, Published by Oxford University Press, ISBN 0-19-853772-7.
- *Fortran 90/95 Explained* by M. Metcalf and J. Reid, Published by Oxford University Press, ISBN 0-19-851888-9.
- *Fortran 90/95 for Scientists and Engineers* by S. Chapman, Published by WCB McGraw-Hill, ISBN 0-07-011938-4.
- *Fortran 90 Handbook* by J. Adams, W. Brainerd, J. Martin, B. Smith, and J. Wagener, Published by Intertext Publications (McGraw-Hill), ISBN 0-07-000406-4.
- *Fortran 90 Programming* by T. Ellis, I. Philips, and T. Lahey, Published by Addison/Wesley, ISBN 0201-54446-6.
- *Introduction to Fortran 90/95* by S. Chapman, Published by WCB McGraw-Hill, ISBN 0-07-011969-4.
- *Programmer's Guide to Fortran 90*, Second Edition by W. Brainerd, C. Goldberg, and J. Adams, Published by Unicomp, ISBN 0-07-000248-7.

For information on parallel programming using OpenMP, see the following:

- *Parallel Programming in OpenMP* by Rohit Chandra, Ramesh Menon, Leo Dagum, David Kohr, Dror Maydan, and Jeff MCDonald, Published by Morgan Kauffman, ISBN 1-55860-671-8.

For information on High Performance Fortran (HPF), see the following:

- *High Performance Fortran Language Specification, Version 2.0*. This specification is available on the web at:

  `http://www.crpc.rice.edu/HPFF/home.html`

3
Release Notes for Prior Version 5 Releases

This chapter contains the following sections:

- Section 3.1 (Product Terminology)
- Section 3.2 (New Features, Corrections, and Known Problems in Version 5.5)
- Section 3.3 (New Features, Corrections, and Known Problems in Version 5.4)
- Section 3.4 (New Features, Corrections, and Known Problems in Version 5.3)
- Section 3.5 (New Features, Corrections, and Known Problems in Version 5.2)
- Section 3.6 (High Performance Fortran (HPF) Support in Version 5.2)
- Section 3.7 (New Features and Corrections in Version 5.1)
- Section 3.8 (New Features and Corrections in Version 5.0)
- Section 3.9 (Additional Information)

3.1 Product Terminology

This document might use the following new or changed product names:

- “HP Fortran” was previously called “Compaq Fortran 90”.
- “Compaq Fortran” was previously called “DIGITAL Fortran 90”.
- “HP Fortran 77” was previously called “Compaq Fortran 77”.
- “Compaq Fortran 77” was previously called “DIGITAL Fortran 77”.
- “Compaq Fortran” refers to the combined packaging of the Compaq Fortran 90 and Compaq Fortran 77 products. It also refers to the Compaq Fortran language. It is used interchangeably with the name “HP Fortran”.
- Compaq Fortran 90 and Compaq Fortran 77 product version numbers are now the same.
- The operating system formerly known as “DEC OSF/1” and “DIGITAL UNIX” is now called “Compaq Tru64 UNIX”.
- The Compaq Extended Math Libraries {CXML} was previously called the DIGITAL Extended Math Libraries {DXML}.

3.2 New Features, Corrections, and Known Problems in Version 5.5

Version 5.5A is a minor release that includes corrections to problems discovered since Version 5.5 was released.

For the Version 5.5A subset numbers, see Section 1.3.

The following topics are discussed:

- Section 3.2.1 (Version 5.5A Corrections)
- Section 3.2.2 (Version 5.5A Known Problems)
- Section 3.2.4 (Version 5.5 ECO 1 Corrections)
- Section 3.2.5 (Version 5.5 New Features)
- Section 3.2.6 (Version 5.5 Important Information)
- Section 3.2.7 (Version 5.5 Corrections)
- Section 3.2.8 (Version 5.5 Known Problems)

3.2.1 Version 5.5A Corrections

From DFA551 X5.5-2602-48C8L to DFA55A V5.5A-3548-48D88, the following corrections have been made:

- Fix a number of bugs in UNPACK.
- Bug 2851. Note3136/CF90AU - Defer processing of character length expression on function definition if it has a forward reference to a currently unknown module procedure.
- Optimize import by ignoring 'uses' which occur in 'private' modules where all 'public' symbols belong to the modules itself.
- Change handling of mult_high intrinsic. Require its arguments to be INTEGER*8.
- Detect unknown, unprintable character in lexeme.
- Fix for allocation of module level symbols, particularly with equivalence.
- bug2853.f90 When context is cray fortran pointee, scalar or array - accept external function or subroutine.
- Fix for alignment of MODULE data.
- Suppress multiple non-standard tab formatting warnings in a program unit.
- Fix SIZE intrinsic when used with the optional DIM argument. Skip over those parts of array expressions that do not contribute to the SHAPE. Example: SIZE(A(2,4,6:10),1) must get the value 5. FORTRAN note 3134.1
- Bug2856: Accept !dec$ attributes no_arg_check in interface blocks and contained procedures. Disable check for scalar/array arg compatibility if arg has no_arg_check attribute.
- Bug2858. Prevent skipping to end of line on a '!' ikn C-style escape strings.
- Allow multiple incarnations of derived type/fields, i.e. for each module procedure.
- Allow KIND parameter of following intrinsics to be INTEGER*1 thru *8 instead of requiring default integer: aint, anint, ceiling, char, cmplx, floor, int, logical, nint, real.

- Bug2860. Derived-type/structure references with dots and percents.
- Improve performance of module import in interface blocks.
- Bug2862. Fix overloading: public sin; interface sin; function sin in module. Fix passing 'sin' as argument of call where 'sin' imported from above module and also from second module with specific having different characteristics.
- Fix another case where we CALL a FUNCTION (with awkward function return value).
- Note 3159. Bug2865. Fix bad code.
- Process optional KIND argument of ICHAR intrinsic.
- Bug2872. Exclude from exported module those imported symbols which are not used; are not in name_list; have no initial values; are not in only list; are not renamed. Problem is that when a module variable matches a dummy argument in the using code - there is not enough context to eliminate the import.
- Assign 'ANINT' as generic function associated with QNINT.
- FIX SIZE= with ADVAVCE='YES'.
- Modify text for message 563 (CVF19095). POINTER component following array component.
- We need to get upper bound information for expressions like: C(i:), especially when requesting bounds checking. This is when "C" is character*(*) dummy argument, but other cases are possible. More rare than it sounds! CVF18791.
- Fix various problems with array constructors with allocatable components. CVF18807.
- Allow .XOR. to be user (re)defined for two logical operands. bug2882.f90
- Fixed FORTRAN:3074 - Added support for 64-bit addressed strings.
- Bug2883 - force evaluation of triplet subscript components.
- Bug2869. Having found a valid module procedure reference, check to see if there is another entry by same name with the EXTERNAL attribute. This takes precedence.
- Bugs 2879 and bug2880 Issue standard warnings for too many continuations.
- Bug2881.f Issue non-standard diagnostics for array constructor character elements of differing lengths.
- Fix bug: data s.r2.r3.i /1/
- Bug2887. Allow constant array elements in array constructors: data a,b/x(1),p(3,4)/ where x and p are parameters.
- Bug2885.f90 Free form, character string continuation. Must skip over continuation character and not include it in string.
- bug2886.f Fix private.meas(1).gotit = 1 .
- bug2891 Relax requirement the format items in format list be separated by delimiters.
- Bug2892.f. Allow all combinations of "w.d" in format specs to be optional. Retain standard checking.

- Bug 2897 - note 3183. Change variable type from INTEGER*16 to *32 to accomodate indices >= 32767.
- Bug2895. When checking if symbol is in common, check also to see if it is equivalenced to common (named and blank).
- Improve MODULE debug information.
- Bug2899 - diagnose calls to non-pure intrinsics from pure procedure.
- bug2900 - diagnose dummy arguments of pure functions which have intent(out) or intent(inout) .
- bug2904 - include zero-size test when checking for missing length specification or '*' length.
- bug2907. Issue standard warning for recursive reference to pure function when no RESULT clause given.
- Bug2910 - Allow common block names, un-enclosed in slashes to be specified in ms_attributes C,ALIAS and DEFAULT directives.
- Bug1189. Allow name conflict between use_associated name and local name provided the procedure does not reference the name.
- bug1024.f90 - Diagnose references to hpf library routines on all platforms other than alpha unix - message 1877
- Allow cray pointers/pointees to be made public/private 81/41 =A; 82/41 =A . Bug2913.
- Fix grammar in message 1875
- Bug2921.f90 ; accept attributes alias,extern and attributes extern,alias.
- Bug 2917. Diagnose missing end on program/procedure.
- Issue non-standard warning for forward reference to derived type.
- bug2924. Recognize result value storage association for multiple entry points. No messages if result types match even if no explicit result assigned.
- Handle: real a(5); !dec$ attributes c,extern::a; end
- When checking for COMMON (to determine if shared) check common block (which is stGLOBAL) as well as variables which are in COMMON. Also - check for data with initial values.
- bug2926 - Allocate blank common variables in program unit following imported blank common. Day 1 bug. If a use statement imported blank common, the local blank common was simply ignored.
- Bug2925 - Diagnose use of statement function in specification expression. Add message 1884. Bug2925. Diagnose use of statement function in specification expression.
- Bug2929. Fix the initialization of temporaries to store bounds expressions for multiple entry points.
- Fixed Fortran:3205.
- Fix problem with a type constructor with a character array argument whose elements are different lengths than those required by the TYPE. Example is id=id_data((/"x","y","z"/)) where id_data has a character array where each element is length 20. From nagf95-f74.f90.

- Bug2931. Diagnose missing private attribute on derived type which has a user-type component which is private. (Standard switch only). Add message 1885.
- bug2932 Diagnose reference to array names or cross sections as operands of UNIT or FILE arguments of OPEN and CLOSE statements.
- Change message 1886. Bug2932. "UNIT and FILE arguments to OPEN and CLOSE statements must be scalars and not arrays".
- Bug2935. Be more rigorous in diagnosing/confirming types which are implicitly declared on parameter statements and then explicitly declared with type statements.
- Bug2941. Allow parenthesized integer expression as unit specifier for I/O statements.
- Return value for selected_int_kind(i) should be one for i <= 2 (all negative values).
- Fix offset problem with structure constructor. This comes from Fortran note 3228.
- Fix various problems with DO loop variables not getting made PRIVATE inside parallel regions.
- Do not assert when array length is zero (testing for overlap).
- Check if repeat count is non-zero before issuing diagnostic about having too many data values. Issue standard warning for multiple/overlapping data initialization of same location.
- bug2947. Diagnose AGOTO reference to format label.
- bug2794.f90 Diagnose parenthesized format specifiers in print and accept statements.
- Numerous fixes for an allocatable dummy argument which is intent(in), especially where the array is type character(*) (bug2944.f90).)
- Bug2951 - Diagnose the passing of an procedure argument, not specified as PURE, to a procedure requiring it to be PURE.
- Bug 2954. Issue diagnostic for use of 'record/type' name in context which makes it look like a function or array reference.
- Bug2955. FIX ".not..and." problem.
- Note3248. Bug2957.f90 . Replace empty argument (originating from adjacent commas) by integer 0. (VAX fortran feature).
- Bug2958 Allow '0' in column 6, fixed form dir, omp, dec directives.
- bug2962 - Issue non-standard warnings for allocatable functions and dummies. Add error messages 1893 and 1894.
- bug2963. Fixed form ends in col 72 or 132, not 1 before.
- bug2961 - Eliminate useless reference to standard rule in message 9. (R730.4 - R730 is 'initialization-expr')
- Reword message 371 to say 'may have' vs 'has' occurred when issuing diagnostic for a reference to a label to within an 'if' or 'do' block.

- Report on disallowed directives uniformly, i.e. with an informational message rather than a warning or error.
- The HPF man pages were updated to reflect its usage with MPI instead of PSE.
- Disallowed directives now get informational message rather than a warning or an error.
- Bug2965 - Imported blank common blocks are not extended by local blank common, nor do they extend local blank common.
- Bug2966 - Recognize 'time' as keyword for arg of intrinsic procedure CPU_ TIME(time=real).
- Fortran note 3259 - When the "-D" command-line option was used, the text of the option was being saved incorrectly.
- Allow -Dfoo from the command line, meaning "define foo to be the null string".
- Fortran note 3262 - Fix a bug in USE processing.
- Allow -DFOO=letters as a valid macro definition. This is the same as -DFOO="letters" but without the quotes.
- When promoting parameter argument do not change the type of symbols which have an explicit kind.
- Fix parsing bug that appeared when '0' found in column 6 of an OpenMP directive.
- Fortran note 3252 - Fix optimizer error: Compiler internal error
- Fortran note 3253 - Fix optimizer error: Wrong answers with -transform_ loops

3.2.2 Version 5.5A Known Problems

- Fortran note 3264 - The compiler dies at compile-time with the message "**Internal compiler error: internal abort**" in some codes that use MODULEs with PUBLIC and PRIVATE data.

3.2.3 Known Limitations

The following limitations exist in Version 5.5:

- When using the `-omp` or `-mp` options, if you declare a parallel DO loop which accesses an INTEGER*1 array (one element at a time), the results may not be correct unless you also specify `-granularity byte`. In general, if a parallel loop tries to store things of "small" size into packed spaces, the granularity needs to be set to that size. However, while this "fixes" the program to do the right thing, it is a dreadful performance slowdown. It would be better to execute such code serially.
- If you specify the `-hpf` (or `-wsf`) option to request parallel processing, the following Compaq Fortran language features are *not* supported:
  - REAL (KIND=16) (same as REAL*16) data type
  - Compaq Fortran 77 implementation of CRAY-style pointers

– Variable format expressions (VFEs). For example:

```
FORMAT(<N+1>I4)
```

– Initialization of Compaq Fortran 77 structures. For example:

```
STRUCTURE /S/
INTEGER I / 100 /
END STRUCTURE
```

3.2.4 Version 5.5 ECO 1 Corrections

Capturing error numbers after OPEN has some complications. The recommended method is to use IOSTAT= and ERR= on the OPEN. ERRSNS returns the UNIX error number after OPEN. IERRNO returns some butg not all UNIX error numbers, depending on the failure. IOSTAT= is the more reliable way to diagnose an error on OPEN.

From DFA550 V5.5-1877-48BBF to DFA551 X5.5-2602-48C8L, the following corrections have been made:

- If -nowarn, suppress informationals too.
- bug2699.f90 Another structure 'dot' problem where rewriting tree in correct precedence had a bug.
- Bug2710.f IMPLICIT($) conflict.
- When diagnosing unprintable strings, use '?'.
- Detect unknown, unprintable character in lexeme.
- Bug2713 - Allow '* -' adjacent operators with -std switch.
- bug2715. Honor '-names as_is' option when initializing character table. Honor '-names as_is' when deriving implicit type of character.
- Bug2716. Elim diag for use of boz constants as initializers in type statements; issue warning if -std switch set.
- Bug2717. Diagnose mismatch between scalar actual arg and array vector arg.
- At routine exit be more selective with deleting allocatable record components. In particular, avoid records declared SAVE, STATIC, or module record variables. CVF16613.
- Avoid Integrity check when passing a large (more than IMAX) structure by value (when the POINTER to the structure is passed, actually).
- Spell allocated correctly in error message.
- Allow ASSIGN 10 to FNAME where FNAME is the current function name.
- Bug2718.f90 ; message 1837
- Bug2725 Unary minus(kind constant).
- Promote arguments to KISHFTC to integer*8. CVF16757
- Fix ISHC with integer*8 arguments. CVF16757
- Allow bktest to be passed as actual argument.
- bug2731. Sort out multiple use of identifiers as vax-structures/f90-derived_ types and other use with and without standard switch.
- bug2727.f90 Associate 'type' determined from assignment context to typeless parameter constant.

- Although LOG and LOG10 are NOT specific intrinsic names, allow them to be passed as actual arguments. This is equivalent to passing ALOG and ALOG10, respectively.
- Explicitly encode required type for second argument of DCMPLX and QCMPLX . bug2747.
- Correct wording of diagnostic 1203. Add new diagnostic 1839 for bug2746. Diagnose (error vs warning) case ranges where low not\ less than high in low:high construct.
- Make ANINT generic; make ANINT and DNINT specifics of ANINT. Formerly ANINT and DNINT were incorrectly specifics of NINT.
- Replace 'syntax error' message by 'FROM and TO arguments of MVBITS must have the same type and kind parameters'.
- Bug2752. Diagnose invalid use of character string or hollerith string in arithmetic context if standard switch is set.
- When an actual array argument needs to be copied (to make it contiguous), and when the user has requested /check:arg_temp_created, the message that the user sees may be corrupted OR the compilation may mysteriously\ die. The attempt to put out a zero character after the name of the called routine ends up dropping the zero one character too far, just BEYOND the string. This causes memory corruption at compile time, and (at least) one EXTRA character after the name of the called routine in the warning message.
- Bug2758 - allow the use of a kind_constant parameter in statement function expressions... i.e. standard rule 1226.5 does not apply.
- Get better locator information for ASSIGNED GOTO variable.
- Fix bug with not-present optional argument used in pure specification function. bug2764
- bug2762. Issue standard warning for 'comma' between read/write i/O specs and iolist.
- When calling an intrinsic that needs to know the size of the default integer kind, make sure that we look at the effects of !DEC$ INTEGER and/or OPTIONS /[NO]I4. BUG2766.
- Never automatically deallocate allocatable components of dummy arguments upon routine exit, even when they are automatically "initialized" upon entry (OUT, but not IN intent). CVF17609.
- Several fixes for allocatable dummy arguments, mostly for INTENT(OUT). The fixes are: DEALLOCATE an ALLOCATABLE dummy argument (if allocated, and INTENT(OUT)) in the prolog of the routine; Allow ALLOCATABLE dummy arguments in ENTRY points (same treatment as for routines); Only do this deallocation for routines/entry points where the argument is present.
- Shorten long created external names, giving a new warning. This comes up most easily with a long Subroutine/Function name within a long Module name. Notice that the maximum name length depends upon the target platform (linker). bug2410.
- Bug2793. Allow pointer components of derived types to be initialized with NULL( ).

- Bug2795. Fix typo in analysis of MATMUL arguments. Pick up left argument, then right argument and not right argument twice.
- Bug2794. Disallow parens in print statement, ala: print (...) .
- Major work on ASSOCIATED. Handle strings of zero length, avoid integrity checks, etc.
- Prevent I/O loop collapsing when there are VFEs in the FORMAT.
- Bug 2796. No statments, including FORMAT and DATA statements, may appear between SELECT statement first CASE statement.
- Bug2803.f90 Elim standard check for what is an unconditional error, i.e. Data x /O"5"*100/ - 'An integer data type is requird in this context'.
- Bug 2810. Allow KIQINT and KIQNNT to be passed as arguments.
- bug2807. Generate error 1847: 'Allocatable fields of derived types are non-standard'.
- Bug2808.f90 . Issue non-standard warning for use of a non-intrinsic elemental procedure as an actual argument. New diagnostic. 1848 'A non-intrinsic elemental procedure shall not be used as an actual argument'.
- Bug2819.f Do not promote function with explicit INTRINSIC to external; diagnose and disallow if referenced with incorrect arg type(s).
- Bug2817. With standard checking impose restriction that length for statement function character dummies be initialization expressions.
- Bug2798, bug2105 and all recent analysis dealing with initialization/ specification expressions and standard enforcement.
- Bug2822. Diagnose use of optional dummy argument as actual DIM argument to ubound, lbound, all, any and count intrinsics.
- Fixed OMP loop index privitization bug. FORTRAN Note 3093.
- Bug2844. Diag 1851 - If dummy arg is allocatable, actual arg must be a whole array and not a section.
- Bug2847. Diag 1853 - An assumed-size array shall not be written as a whole array reference except as an actual argument in a procedure reference for which the shape is not required. Also - rewrite diag 364 .
- Bug2846. Diag 1852 - The use of a scalar in a structure constructor for an allocatable array component is not allowed.
- Bug2846. Diagnose the use of a scalar constant in a structure constructor for an allocatable array component.

3.2.5 Version 5.5 New Features

The following new Compaq Fortran features are now supported:

- The following new features are now supported:
  - f90 V5.5 contains the initial work to support allocatable components of derived types.
  - OpenMP enhancements: support for nested OpenMP parallel regions, and added support for the num_threads clause which selects the number of threads a parallel region will use based on a run time expression.

- Notice a difference between the Fortran 95 standard feature of initialized derived types and the VAX Fortran extension of initialized STRUCTUREs: initialized derived types are not SAVEd by default; initialized STRUCTUREs are.
- Improved algorithm for making inlining decisions that will tend to do more inlining, especially when many small routines are inlined into one another.
- Improvements to general optimizer: prefetching, software pipelining, strength, pulling branches out of loops, allowing more existing optimizations to fire.
- Parallel programs running on Non-Uniform Memory Access (NUMA) machines using the following directives, compiler options, and environment variables is available but unsupported:
  * !DEC$ MIGRATE_NEXT_TOUCH
  * !DEC$ MIGRATE_NEXT_TOUCH_NOPRESERVE
  * !DEC$ OMP NUMA
  * !DEC$ DISTRIBUTE - same as !HPF$ DISTRIBUTE
  * !DEC$ ALIGN - same as !HPF$ ALIGN

```
 -numa
Fortran 90 only.  Enables NUMA parallel processing and the NUMA
command line options -numa_memories and -numa_tpm.  NUMA paral-
lel processing is indicated by inserting certain !DEC$ direc-
tives in your source code.  Option -omp must also be specified.
The default is -nonuma.

 -numa_memories num
Fortran 90 only.  Specifies the number of memories (RADs) to be
used for NUMA parallel processing.  Option -numa must also be
specified.  If -numa_memories does not appear on the command
line or if num is 0, the number of memories will be chosen at
run-time either from the NUMA_MEMORIES environment variable (if
it is set) or by the system.

 -numa_tpm num
Fortran 90 only.  Specifies the number of threads per memory to
be used for NUMA parallel processing.  Option -numa must also
be specified. num is the number of threads per physical memory
that will execute NUMA parallel features in the program.  If
-numa_tpm does not appear on the command line or if num is 0,
the number of threads per memory will be chosen at run-time
either from the NUMA_TPM environment variable (if it is set) or
by the system.
```

  The "Parallel Processing on Tru64 UNIX Systems" manual that describes NUMA parallel processing in Compaq Fortran is available after installation in /usr/lib/cmplrs/fort90/fort55_parallel_manual.ps .

- The following new `f90` command options are now supported. See the f90(1) man page for details.
  - Nothing new in this version.
  - `-hpf_target pse` is no longer supported.

- The following new run-time features are now supported:
  - Nothing new in this version.
- See the new features listed in Section 3.3.1 (Version v5.4A ECO #1 New Features)
- The Compaq Extended Math Library (CXML) routines are unchanged in the Compaq Fortran kit. See the CXML release notes in:

  /usr/opt/XMDCOM410/docs/XMD410_release_note.txt

3.2.6 Version 5.5 Important Information

Some important information to note about this release:

- As of Compaq Fortran V5.3, the `f77` command executes the Compaq Fortran 90 compiler instead of the Compaq Fortran 77 compiler. Use `f77 -old_f77` to execute the Compaq Fortran 77 compiler.
- Object files created by Fortran 95 contain information about when they were compiled and by what version of the compiler. Use the command

  ```
  strings -a xxx.o | grep "@(.)"
  ```

  where `xxx.o` is the name of the object file. The strings generated by the compiler are
  - "(#)tttt" is a what(1) string with the text from a cDEC$ IDENT directive if present.
  - "@(c)Compaq Fortran Vn.n-eeee" is the version number of the Fortran 95 compiler that generated this object file.
  - "@(m)xxx" is the name of the first program unit in the source used to produce this object file.
  - "@(d)dd-mmm-yyyy hh:mm:tt" is the date and time this object file was created.

3.2.7 Version 5.5 Corrections

From DFA542 ECO #1 X5.4A-1684-46B5P to DFA550 FT1 T5.5-1736-48B88, the following corrections have been made:

- Fix UNIX driver to diagnose -fpscomp with no argument.
- Allow 'null( )' in structure constructors to initialize allocatable fields.
- Accept and promote SUBRNAME/FUNCNAME symbols previously defined in interface blocks as module procedures.
- Reset NUM_THREADS to NULL when processing a !$OMP PARALLEL directive.
- Allow NUM_THREADS clause on an !$OMP PARALLEL SECTIONS directive.
- Resolve references to derived type fields by combining visibility of various pieces of the hierarchy from different modules.
- Treat hex/oct - typeless constants as int8 when used as args of call on 64-bit processors; when typeless constant parameters are used, force to int8 and not to default integer type.

- Handle initialization of fields of structures of the form: type :: variable = constant
- Diagnose attempt to initialize a scalar field with an array.
- Give a fatal error when NUM_THREADS is a constant value .le. zero.
- Add error message 1830: "Multiple OMP NUM_THREADS clauses in a parallel region are not allowed".
- Possible to get wrong answer when MAX or MIN references an optional argument.
- Begin generating MODULE debug information.
- Significantly improve compilation time for certain large (and usually machine-generated) programs when uninitialized variable detection is not selected and optimization level is 0.

From DFA550 FT1 T5.5-1736-48B88 to DFA550 FT2 T5.5-1775-48B9D, the following corrections have been made:

- Additional (re)initialization to fix internal compiler errors when compiling with /separate_compilation on vms.
- Additional (re)initialization to fix internal compiler errors when the same file appears twice on command line.
- Correctly associate defined assignment generic procedures.
- Give -std warning for adjacent operators even if operand is numeric.
- When terminating a compilation because of too many diagnostics of a given severity, msg_report would try to append an "s" to a string in read-only memory. Manipulate a local character buffer instead.
- Add more module debugging information.
- Resolve one more conflict resulting from overloading of 'dot' as field separator and as dotted_operator delimiter, when the field name matches operator name, e.g. 'LT'.
- Treat 'optional' arguments as 'matching' arguments when comparing generic specifications.
- We need to mark variables in a CRITICAL directive (both -mp and -omp) as VOLATILE_READS and VOLATILE_WRITES.
- Add new error messages: 1831 - 'List directed ENCODE/DECODE not supported'; and 1832 - 'Greater than 7 dimensions is non-standard'
- With standard switch - warn about use of greater than 7 dimensions.
- Disallow list directed ENCODE/DECODE.
- Enhance debug output for automatic interface generation.

From DFA550 FT2 T5.5-1775-48B9D to DFA550 V5.5-1877-48BBF, the following corrections have been made:

- Fix CVF15588. ICE when passing a non-contiguous array with -check bounds.
- Fix BUG2658. Nested WHERE with different data types can get ICE when we put out an IAND without converting one of the data types. The bug does NOT happen with a WHERE in a FORALL.

- Note 2953. Bug2661.f90. Correct alignment computations for unions. Align each 'map' component to most demanding component.
- Change message about unused statement functions from warning to info. Note 2954. Bug 2662.
- CVF15596. Determine correctly the length of a character function that uses intrinsics.
- Bug2663. Note 15459. Fix incorrect handling of intrinsics in SUBSTRS. Recognize possibility of null string in array constructors.
- CVF 15569. Make sure record/structure types of common variables are exported {recursively}.
- Bug2665.f bug bug2666.f90 When dummy argument of procedure in an interface block is an undetermined procedure (subroutine or function) - issue diagnostic if the procedure is referenced with an actual array arg.
- Bug2664.f90 - diagnose name which is both a pointee and a common variable, regardless of which specification comes first.
- Bug2668.f90, Note 2959. Resolve symbols in bounds expressions even if PRIVATE.
- Implement feature, under switch control, which causes all quoted strings to be interpreted as 'c-strings', i.e. having trailing binary null character; and which may contain escape sequences.
- Bug2667. Issue diagnostic for reference to function result field outside its host associated scope: 'Use of this function name is not valid in this context'.
- Bug2672.f90 Detect 'rename' when resolving intrinsic function references.
- Issue message indicating conflict between identical names from different modules - one generic and one specific.
- Cvf15747. Another place where "." was not equivalent to "%" in a field reference.
- Bug2673. Elim erroneous diag of common equivalences.
- -fast => -assume cc_omp (unix driver only). DFB3612. {should only happen if -omp is present}
- Bug2678.f90 . Recognize the fact that a record constructor does not contain only constants but contains variable(s) as well.
- Support allocatable components of records. The changes support assignment, automatic deallocation of locals at end of routine, structure constructors, et. al. The biggest piece of code relates to assignment, where we do nested deallocation/ allocation on where the LHS contains allocatable components which are either not yet allocated or allocated to a different size from the RHS.
- FORTRAN Note 2977. Enable code which inhibits reload of imported module which has neither ONLYs nor renames.
- Allow inquire(iolength=l) for pointer and allocatable array fields.
- Improve handling of bad source line, free form, having only a number of 10-digit integers. Use max of 5 digits in label construction.
- Refine test for complex number.

- Make initialized VAX structures SAVE but initialized DT not SAVEd.
- bug2680.f90 - Walk components of array constructor and if typeless and untyped, attach type to components and to constructor. Inhibit generation of diag for use of typeless constant in an array constructor where there is one or more elements which are not typeless.
- When importing generic names and looking to see if name matches an intrinsic, used renamed string, if any. bug2690.
- bug2691.f One more conflict resulting from overloading of 'dot' as field separator and as dotted_operator delimeter.
- Bug2688.f90 For associated intrinsic - check attributes of component of derived_type/vax_structure for pointer/target properties, as well as of simple operand.
- Prevent using declaring module field as in if null, as for blank common block entries. Bug2695
- Bug2689. Add diagnostic 1835: "Record fields or array elements or sections of pointers are not themselves pointers." For associated intrinsic - disallow use of array elements or array cross-sections or fields of pointer records as 'pointer' operand. Issue diagnostics.
- bug2691.f Maintain left to right precedence in trees originating from overloaded use of dot operator
- bug2697.f Compute array descriptors and record sizes as function of platform.
- -fast => -assume cc_omp only when -omp seen. See FORTRAN 2999.

3.2.8 Version 5.5 Known Problems

The following known problems exist with Compaq Fortran Version 5.5:

- The following is a list of known problems for `-omp` and `-mp` parallel support in Version 5.5:
  - Global variables referenced by statement functions inside parallel regions should not reference local instances of those variable names. The following example should print 42.0 {10 times} instead of 0.0:

```
    real x,y(10)
    statement(a) = a + x

    x = 41.0
!$par parallel local(x)
    x = -1.0
!$par pdo local(i)
    do i=1,10
       y(i) = statement(1.0)
    end do
!$par end parallel
    type *,y
    end
```

- Please note that `-warn decl` gives an error level message, not a warning level message.

- When using Ladebug with certain versions of the UNIX operating system, be aware that a trailing underscore may be needed to display module variables. For example, to display variable X in module MOD, if typing `print $MOD$X$` does not display the variable's value, type:

  ```
  print $MOD$X$_
  ```

3.3 New Features, Corrections, and Known Problems in Version 5.4

Version v5.4A ECO #1 is a minor release that includes corrections to problems discovered since Version 5.4A was released and certain new features.

The following topics are discussed:

- Section 3.3.1 (Version v5.4A ECO #1 New Features)
- Section 3.3.2 (Version 5.4A New Features)
- Section 3.3.3 (Version 5.4 New Features)
- Section 3.3.4 (Version 5.4 Important Information)
- Section 3.3.5 (Version 5.4 Corrections)
- Section 3.3.6 (Version 5.4 Known Problems)

3.3.1 Version v5.4A ECO #1 New Features

The following new Compaq Fortran features are now supported:

- The following new features are now supported:
  - Modify the way that complex (and double complex) arguments are passed (and received) by value. This is an INCOMPATIBLE CHANGE. Old objects/libraries will NOT work with new. The end result is that on ALL platforms we will pass a complex (or double complex) as two real (or double) values when the user requests passing by value.
  - An undefined compile time variable in an expression now has the value 0.
  - Initialized records are re-initialized at every entry point.
  - COUNT and ZEXT now allow optional 'KIND' argument.
  - The IF DEFINED directive is documented in the LRM so that IF .NOT. DEFINED(...) isn't allowed; but it is. DEFINED is like a logical function and can be part of the expr in IF (expr)... .

From DFA541 V5.4A-1472-46B2F to DFA542 X5.4A-1684-46B5P, the following corrections have been made:

- Fix incorrect processing of %fill(lb:ub).
- For otherwise unattributed parameter, assign to it the type of its value (where, typeless constant value defaults to default-type-integer).
- Fix problem which occurs when a symbol which exists in a public list, is visible in one 'use' chain, and is mistakenly treated as 'public' when imported from a second use chain where it is private and invisible.
- Do platform-specific diagnostics for open keyword specifiers.

- Issue diagnostic for an actual argument which has a vector subscript if dummy has intent(out) or intent(inout). Do this even if arg of defined_ assignment subroutine.
- Fix problem that resulted in incorrect warnings about RESHAPE function and the size of the ORDER argument.
- Accept 'character*(length) %fill' without '::' separating type/entity.
- In the Unix driver, don't link against libexc if the user is compiling multi-threaded. When they compile multi-threaded, cc automatically adds it in the correct place in the "lineup".
- Implement ALIGN=xx values consistently. Also, support ALIGN=PAGE on all platforms (which is new) and set the alignment to whatever a "page" means on that platform.
- Generate better debug information for dummy arguments which are subroutines or functions. In particular, make a dummy argument which is a subroutine look like a subroutine instead of a function. In addition, if there is an INTERFACE specification for the dummy routine, put out information about the arguments. For example:

```
SUBROUTINE FOO(BAR)
INTERFACE FUNCTION BAR(X,I)
DOUBLE PRECISION X
INTEGER I
END FUNCTION
END INTERFACE
...
END
```

We will now provide debug information for the arguments X and I.

- Collect ALL generic entries for a name from the set generated during import. Multiple modules may validly contain generic specifications for a single symbol.
- Process fpscomp options general and ldio_spacing in a left-to-right order.
- Fix SIZE(A,DIM) where DIM is an optional argument.
- Correctly diagnose an incorrect number of arguments in a structure constructor.
- Allow MAX and MIN to have optional arguments.
- Optimize module importing when 'use' occurs within interface blocks.
- Modify the way that complex (and double complex) arguments are passed (and received) by value. This is an INCOMPATIBLE CHANGE. Old objects/libraries will NOT work with new. The end result is that on ALL platforms we will pass a complex (or double complex) as two real (or double) values when the user requests passing by value. This will also happen when the user requests that the routine have a "C" calling standard. There are two motivations for this incompatible change: the Alpha Calling Standard changed and the C++ standard is going to introduce a COMPLEX data type, which will be implemented according to the Alpha Calling Standard.
- Mark the array descriptor for an explicitly shaped array function result as quadword aligned.
- Ignore migrate directives if -omp and -numa are not set.

- Treat use of undefined compile time variable in expression as integer with value 0. Do not diagnose.
- Get debugging information put out at all (appropriate) ENTRY points. In particular, put out prolog code for dynamic array bounds for array dummy arguments and for f90 automatic arrays. Before this edit, we were only putting out prolog code at the FIRST entry point referencing the array as an argument (or the subroutine/function for the case of an f90 automatic array).
- Handle generic defined operator, merging specifics from multiple modules; honoring 'only' defined operator.
- VAX structures like F90 defined types.
- Fix bug in WHERE in FORALL where WHERE mask contains a call to an elemental function.
- Initialized records need to be re-initialized at every entry point. With edit 1213 we changed our handling of initialized records to give them the SAVE attribute. This was in keeping with long-standing f77 practice and documentation. However, Fortran-95 prescribes that local initialized defined-types (records) be initialized every time into a routine. This requires a documentation change (to SAVE).
- Fix generated code for optional arguments to SELECTED_REAL_KIND.
- Recognize match between actual 'procedure' argument with pointer attribute and dummy 'procedure' argument with pointer attribute.
- Accept a VAX scalar field name reference (x.y) where x currently known as parameter object of structure type.
- Recognize renamed generic in 'use' where there also is an 'only' clause.
- COUNT and ZEXT now allow optional 'KIND' argument.
- Get the LOCATOR correct for all ENTRY SYMBOLs. Debugging problem.
- Do NOT evaluate the argument to any numeric inquiry function (like HUGE).
- Make actual argument which is a function returning a scalar pointer work correctly.
- Fix access to module from path specified on command line.
- Array passing optimization was not working correctly for assumed shape arrays being passed as explicit arrays.

3.3.2 Version 5.4A New Features

The following new Compaq Fortran features are now supported:

- The following new features are now supported:
  - When `-fast -std` is specified {using any of the `-std*` options, the `-align` options `dcommon` and `sequence` will not be set.
  - `-hpf n` is now the preferred spelling for `-wsf n`.

From DFA540 V5.4-1265-46ABA to DFA541 V5.4A-1472-46B2F, the following corrections have been made:

- Allow Sequence/Nosequence directives on fields of derived types with -hpf, whether or not numa switch is set.

- Add diag :"The NOSEQUENCE directive may not be specified for a component of a derived type which has the SEQUENCE attribute.",
- Issue diag if NOSEQUENCE directives are applied to fields of 'sequence' derived types.
- Use new RTL routine, for_check_mult_overflow( ), to calculate the size of an array to be allocated, passing flag returned to for_allocate( ).
- Fix obscure bug with OPTIONAL mask for intrinsics MAXLOC et al. The mask has an expression for the dimension bound(s).
- Allow, without assertion violation - subroutines generated by transform to be ignored during export.
- When a psect becomes too big, give a better error message: ME_STOREQEXC "Psect !AD is too large. Reduce array sizes or make arrays AUTOMATIC or ALLOCATABLE."
- Allow character substring assignment in forall.
- Export intrinsic function which is a module generic if it is referenced.
- Make -fast set the align_dcommons bit. The effect is ONLY on the listing (the code was already doing correct alignment; only the listing was incorrect).
- Make much better locators when debugging programs with adjustably dimensioned arrays.
- Deal with PRIVATE fields of a type that is not PRIVATE.
- Fix several problems with nested modules with rename and only.
- When exporting function/subroutines, walk 'used' parameter list and collect those derived types which are needed, i.e. add to export list. (Similar to collecting types of dummy args for export).
- Fix regression in inline_sizeof.
- Call foo(NINT) should pass JNINT instead of ININT.
- Add support for passing KIDNNT, KNINT and IQNINT as actual arguments on Unix platforms.
- Passing IDINT should use integer*4 instead of integer*2 routine names.
- Issue diagnostic for character string as arg to RECL= in OPEN statement.
- If substring index in a FORALL assignment stmt uses one of the indexing variables, need to calculate size differently, otherwise get used before defined problems.
- Walk dummy arguments as well function result, parameters and globals used as bound variables to pick up for export 'record types' which are not otherwise used in module procedure.
- Fix problem with -warn truncated.
- If the argument to DFOR$PREFETCHxxx is any real or complex type, use that type on the PREFETCH tuple. All other types continue to use INT32.
- Diagnose prefetch arguments which are not variables, array element references or structure component references.
- Diagnose missing intent attributes on pure (explicitly or implicitly if elemental) subroutine args which are neither pointers nor procedures.

- Diagnose mask expressions with incompatible shapes in where constructs and contained where/elsewheres.
- Relax strict statement/token construction if errors encountered while processing cdec directive 'if' or 'elseif' .
- When terminating nested do loops/constructs, recognize and process OMP END DO.
- Impose standard restriction that defined-operator names consist only of letters.
- Correctly pass complex by value on Alpha platforms. This fix makes us follow the Alpha calling standard. Note: This is an INCOMPATIBLE change.
- Allow -fpscomp [no]ldio_spacing.
- -fpscomp:general => -fpscomp:ldio_spacing.
- Handle leading 0s in subscript constants.
- Fix problem with dot as field separator.
- For default complex, special case ccos, cexp, clog, csin, csqrt to be type of arg; similarly for default real type, special case alog, alog10
- Fix inquire with IOLENGTH=FIELD%NAME or array element.
- Array in derived type and explicit-shaped array slices now avoids temp creation and copy-in/copy-out when passing arrays.
- Do not diagnose fields of private types as inaccessible within the defining module, just outside the module.
- Disallow ALLOW_NULL/REFERENCE attributes with assumed shape arrays.
- Make fields having a PRIVATE derived type invisible.
- Allow dummy args to be used as args to inquiry intrinsics in specification statements without requiring intent IN. Force private module variables used in specification statements of a procedure, even as args to inquiry intrinsics, to be exported.
- Allow NUM_THREADS clause in a PARALLEL directive. This already was handled properly in a PARALLELDO directive.
- Fix problem with FORALL masks with -integer_size 64.
- Alphabetize annotations, fpscomp options.
- When putting out a null for a not present optional argument, get the arg position correct.
- Fix code in forall/elsewhere processing.
- Alphabetize show, warnings options. Restrict -warn hpf to unix.
- If /fpscomp:general, and STATUS omitted from OPEN, supply "default" code rather than "new" or "unknown", because FPS had its own rule for this that doesn't match what
- If not 'renamed' force the loading of generic/specific declaration record whose generic name appears in source; resulting in the collecting of individual specifics from different modules.

- Detect and diagnose illegal value in octal/hex/binary constant when used as operand to unary operator
- Accept '$' at the end of a range of letters in the implicit statement.
- Allow -hpf [n] on unix; treat like -wsf [n].
- Speed up compiling DATA initialization statements.
- For -assume dummy_alias, make all COMMON block variables and Module variables always work.
- Change error code for edit 1396. Make 1813. 'ALLOW_NULL and REFERENCE attribute pair is disallowed with assumed shape arrays.'
- When detecting end of statement function acceptability, and sitting on array element/section assignment, conclude function result-variable processing to make variable available to statement in progress.
- Consider duplicate if 'optional arg' specs match and if all non-optional arg specs match.
- Fix PACK when the MASK is .FALSE. (or when the MASK is a scalar expression whose value is not known at compile time).
- Fix "mapped" pointee checking problem. The code which attempts to deliver an error when a pointee is HPF "mapped" cannot handle array pointees.
- Fix INTEGER x(10), y(10); DATA x(1:10) /1/; DATA y/10*0/; END
- Integer*8 fixes for selected_xx_kind, passing integers, and added for_kdate entry point.
- Within a procedure in a 'contain' scope, identify <type-function-name> strings as type statements and not as function definitions.
- To conditionally diagnose non-standard, calculable, recursive use in type-parameter specification, of symbol being defined . CHARACTER :: C(10)*(SIZE(C,1))
- Detect and diagnose the assignment to a function result of external procedure, and not to function result of current
- Diagnose instead of generating assertion violation, non-calculable, recursive use in bounds specification, of symbol being defined. INTEGER :: A(KIND(A)),G(BIT_SIZE(G)),I(DIGITS(I)).
- Allow recursive reference to parameter by inquiry functions in initialization expression if query is about known property. TYPE PARAMETER :: EP = BIT_SIZE(EP), HP = DIGITS(HP), XP = EPSILON(XP) .
- Cause front end to generate warning diagnostics for integer overflow, i.e. when I4 computation exceeds I4 capacity.
- Issue error for previously undiagnosed VOLATILE/PARAMETER declaration.
- If -fast is set *and* -standard (however it's spelled) is set, don't set the non-standard switches such as -align dcommons and -align sequence.
- Improve instruction scheduling {SLOT4}.

3.3.3 Version 5.4 New Features

The following new Compaq Fortran features are now supported:

- The following new features are now supported:
  - `-pipeline` is now the default at optimization level `-O4` (the default).
  - Nested parallel regions are now supported by `-omp`.
  - Parallel programs running on Non-Uniform Memory Access (NUMA) machines have initial support using the following directives, compiler options, and environment variables:
    * !DEC$ MIGRATE_NEXT_TOUCH
    * !DEC$ MIGRATE_NEXT_TOUCH_NOPRESERVE
    * !DEC$ OMP NUMA
    * !DEC$ DISTRIBUTE - same as !HPF$ DISTRIBUTE
    * !DEC$ ALIGN - same as !HPF$ ALIGN

```
-numa       Fortran 90 only. Enables NUMA parallel processing and the NUMA
     command line options -numa_memories and -numa_tpm.  NUMA paral-
     lel processing is indicated by inserting certain !DEC$ direc-
     tives in your source code.  Option -omp must also be specified.
     The default is -nonuma.

-numa_memories num
     Fortran 90 only. Specifies the number of memories (RADs) to be
     used for NUMA parallel processing.  Option -numa must also be
     specified.  If -numa_memories does not appear on the command
     line or if num is 0, the number of memories will be chosen at
     run-time either from the NUMA_MEMORIES environment variable (if
     it is set) or by the system.

-numa_tpm num
     Fortran 90 only. Specifies the number of threads per memory to
     be used for NUMA parallel processing.  Option -numa must also
     be specified. num is the number of threads per physical memory
     that will execute NUMA parallel features in the program. If
     -numa_tpm does not appear on the command line or if num is 0,
     the number of threads per memory will be chosen at run-time
     either from the NUMA_TPM environment variable (if it is set) or
     by the system.
```

- The following new `f90` command options are now supported. See the f90(1) man page for details.
  - `-arch ev67`
  - `-ccdefault fortran | list | none | default`
  - `-annotations`
  - `-assume noprotect_constants`
  - `-check arg_temp_created`
- The following new run-time features are now supported:
  - The Fortran RTL has been changed so that it now processes the [.m] (the minimum number of digits) portion of the edit descriptor when w (the width field) is Zero for I, B, O, and Z editing.

- An end-of-file or end-of-record status is no longer treated as an error status. This change was done to adhere to the Fortran 90 language standard. Prior to this change, if there was an ERR= specified and no END= | EOR= for a READ operation and the READ encountered an end-of-file or end-of-record situation, the ERR= path would be followed. With this change, this situation would now result in a fatal message being generated.
- The runtime library now contains support for a new environment variable, FORT_CONVERT_ext that allows a user to associate a foreign data conversion option with files of a particular file extension. The values of FORT_CONVERT_ext are the same as FOR_CONVERTn.
- The runtime library has been changed to perform more thorough edit checking on list directed input. Previously, the RTL was liberal in what it accepted for input to integer and real values. In accordance with the F95 Standard, the RTL no longer accepts "+", "-", ".", "D", "E", or "Q" without expressing at least 1 digit. For example, the RTL used to allow a single "+" to convert to a 0, but now the RTL will return a FOR$IOS_LISIO_SYN error. In addition, ambiguous expressions such as "+-" and "–" will be rejected.
- Support for the -fpscomp options are now implemented in the Fortran RTL on UNIX:

  * -fpscomp all
  * -fpscomp general
  * -fpscomp ioformat
  * -fpscomp logicals
  * -fpscomp filesfromcmd

  This support includes the six Microsoft PowerStation compatible file types as described in the Visual Fortran Programmer's Guide:

  * MS Unformatted Sequential Access
  * MS Unformatted Direct Access
  * MS Formatted Sequential Access
  * MS Formatted Direct Access
  * MS Binary Sequential Access
  * MS Binary Direct Access

  This includes all fpscomp behavior that does not involve Windows-specific features. There is no QuickWin-like support. The 'Filenames from Command Line' option looks in the command line arguments for unspecified filenames in an OPEN(...,FILE='',...) statement and will also prompt for filenames at the terminal, but does not implement the equivalent of a Windows Dialog box, like a QuickWin application would in the PC Windows environment. Windows-specific physical device names used as filenames are given no special consideration or handling.
- Support for reading nondelimited character strings as input for character NAMELIST items has been added.

A character string does not need delimiting apostrophes or quotation marks if the corresponding NAMELIST item is of type default character, and the following is true:

* The character string does not contain a blank, comma, slash, exclamation point (!), ampersand (&), dollar sign ($), left parenthesis, equal sign (=), percent sign (%), or period (.).
* The character string is not continued across a record boundary.
* The first nonblank character in the string is not an apostrophe or a quotation mark.
* The leading character is not a string of digits followed by an asterisk.

A nondelimited character string is terminated by the first blank, comma, slash, end-of-record, exclamation point, ampersand, or dollar sign encountered. Apostrophes and quotation marks within nondelimited character strings are transferred as is.

Should an equal sign, percent sign, or period be encountered while scanning for a nondelimited character string, the string will be treated as a variable name (or part thereof) and not as a nondelimited character string.

Be forewarned that nondelimited character strings that are written out by using a NAMELIST write may not be read in as expected by a corresponding NAMELIST read.

Given the following example code:

```
NAMELIST/TEST/ CHARR
CHARACTER*3 CHARR(4)
DATA CHARR/'AAA', 'BBB', 'CCC', 'DDD'/
OPEN (UNIT=1, FILE='NMLTEST.DAT')
WRITE (1, NML=TEST)
END
```

The output file NMLTEST.DAT will contain:

```
 &TEST
  CHARR   = AAABBBCCCDDD
 /
```

Should an attempt be made to read the data in NMLTEST.DAT back in with a NAMELIST read using nondelimited character strings:

```
NAMELIST/TEST/ CHARR
CHARACTER*3 CHARR(4)
DATA CHARR/4*'   '/
OPEN (UNIT=1, FILE='NMLTEST.DAT')
READ (1, NML=TEST)
  PRINT *, 'CHARR read in >', CHARR(1),'<  >',CHARR(2),'<  >',
1 CHARR(3), '<  >', CHARR(4), '<'
END
```

This will result in:

```
 CHARR read in >AAA<  >   <  >   <  >   <
```

– New run-time jacket routines provided for Fortran 3f lgamma, erf, erfc, short and long functions in libUfor. See the appropriate 3f man page for details.

- The run-time now supports unlimited record sizes for writes, reads, backspaces of variable length unformatted files. Records greater than 2.1 giga-bytes use a new scheme that may not be portable to another vendor's Fortran.
- When the DATE, TIME, or ZONE arguments to the DATE_AND_TIME intrinsic routine are not large enough to hold the required information, a fatal run-time error will now be generated.
- The run-time support was enhanced to allow a REWIND operation to be performed on a direct access file. This is allowed without having to specify any command line options.
- New bits were defined in the for_set_fpe routine interface to allow the compiler to specify counting and messages for floating point inexact traps (fpe:6). The runtime handler is updated to provide this service.
- New run-time jacket routines provided for Fortran 3f lgamma, erf, erfc, short and long functions in libUfor. New entry point interfaces are:

```
float lgamma_ ( float *x ) ;
double dlgamma_ ( double *x ) ;
float erf_ ( float *x ) ;
double derf_ ( double *x ) ;
float erfc_ ( float *x ) ;
double derfc_ ( double *x ) ;
short short_ ( int *x ) ;
int long_ ( short *x ) ;
```

- Support for generating traceback information has been added to the run-time support. Traceback will be produced for all severe errors. Traceback output can be disabled by setting the environment variable FOR_DISABLE_STACK_TRACE to true. All diagnostic output, which includes traceback, can be redirected to a file defined by the FORT0 environment variable, which was already supported.
- Namelist input was not handling slices and strides of arrays, array segments with zero and negative positions, and character substrings of arrays. Note: this implementation adheres to the F90 Standard, so nested array slices are illegal.
- Support for the environment variable FORT_BUFFERED was added. When it is set to TRUE, the run-time library will assume that buffered I/O will be used for output to all I/O units, except those whose output is to the terminal. This provides a run-time mechanism to support the behavior enabled by the `-assume buffered_io` option.

- See the new features listed in Section 3.4.2 (Version 5.3 ECO 01 New Features)
- The Compaq Extended Math Library (CXML) routines are updated in the Compaq Fortran kit. See the CXML release notes in:

  /usr/opt/XMDCOM410/docs/XMD410_release_note.txt

3.3.4 Version 5.4 Important Information

Some important information to note about this release:

- As of Compaq Fortran V5.3, the `f77` command executes the Compaq Fortran 90 compiler instead of the Compaq Fortran 77 compiler. Use `f77 -old_f77` to execute the Compaq Fortran 77 compiler.
- Object files created by Fortran 95 contain information about when they were compiled and by what version of the compiler. Use the command

  ```
  strings -a xxx.o | grep "@(.)"
  ```

  where `xxx.o` is the name of the object file. The strings generated by the compiler are

  - "(#)tttt" is a what(1) string with the text from a cDEC$ IDENT directive if present.
  - "@(c)Compaq Fortran Vn.n-eeee" is the version number of the Fortran 95 compiler that generated this object file.
  - "@(m)xxx" is the name of the first program unit in the source used to produce this object file.
  - "@(d)dd-mmm-yyyy hh:mm:tt" is the date and time this object file was created.

3.3.5 Version 5.4 Corrections

From DFA531 ECO 01 final X5.3-1120 -44A7B to DFA540 FT1 T5.4-1130-46A7R, the following corrections have been made:

- Eliminate internal compiler error for character function whose length expression includes LEN_TRIM of a dummy argument.
- Eliminate internal compiler error for SIZE of a derived type component pointer array.
- Software pipelining is now enabled at the default optimization level (-O4)
- Eliminate more unnecessary copies of contiguous array arguments.
- Correct parsing error for certain record component references using dots.
- Eliminate internal compiler error for particular use of RESHAPE.
- Do not give an unused variable warning where the only reference to a variable is as the argument to LOC.
- Eliminate internal compiler error for particular use of nested STRUCTURE.
- Disallow illegal ALLOCATED(A(:))
- Allow, as an extension for compatibility with our Fortran 77 compilers, a LOGICAL value as an argument to intrinsics which accept INTEGER, such as ABS.
- Diagnose ambiguous generic routine reference.
- When an integer constant is assigned to a variable whose KIND is smaller, don't change the KIND of the constant for future references.
- Allow IMPLICIT REAL with $.
- Eliminate internal compiler error for RESHAPE of array constructor

From DFA540 FT1 T5.4-1130-46A7R to DFA540 FT2 T5.4-1170-46A97, the following corrections have been made:

- Speed up processing of EQUIVALENCE groups in COMMON blocks.
- Properly handle SIZE(A(:)(I:J))
- Implement INT8 intrinsic (already documented)
- Implement !DEC$ PSECT NOWRT for COMMON blocks on UNIX.
- Allow omitted OPTIONAL arguments to be passed as optional arguments to intrinsics.
- If "too many errors", suppress informational/warning messages as well.
- Allow keyword specification of arguments A10, A20, etc. to MAX/MIN.
- Eliminate internal compiler error for case of ADJUSTL with array argument.
- Implement nested LASTPRIVATE for COMMON.
- Where DATA initializations initialize the same array element (not allowed according to the standard, but supported by Compaq Fortran), preserve the order in which initializations were specified, so that the last one takes precedence.
- Fix parsing error for typed function declaration with MS-style argument passing specification in argument list.
- Resolve incorrect generic resolution between subroutine with no arguments (but ( ) specified as argument list) and routine with one required argument.
- Allow named constant to be used in specification expression later in same statement where it was defined.
- Give error instead of compiler abort for IF (expr) SELECT CASE
- For F90 standards checking, warn about I0, etc. in FORMAT.
- Warn about variables in BLOCK DATA subprogram that are not in a COMMON.
- Fix problem with continuation in free-form source when OpenMP conditional compilation is used.
- Eliminate internal compiler failure for incorrect ". AND." in free-form source.
- Fix a case where two NaNs sometimes compared as equal.
- Fix assertions caused by instruction scheduling at -fpe1.

From DFA540 FT2 T5.4-1170-46A97 to DF540 FT3 T5.4-1195-46AAC, the following corrections have been made:

- Support the full range of format width specifiers as documented.
- Correct problem with page-alignment for POINTER objects.
- Improve detection of contiguous array slices.
- Allow EOSHIFT to work on REAL*16 and COMPLEX*32 arrays.
- Correct support for SIZE with argument being array with vector subscripts.
- Eliminate internal compiler failure for incorrect ". AND." in free-form source.
- Correct problem with fixed form continuation with -assume cc_omp.

- Improve handling of continued C-string escape sequences.
- Preserve order of data initializations where the same location is initialized multiple times. While this is not allowed by the standard, we do support it.
- Eliminate spurious warnings for use of some intrinsics when -double_size 128 is used.
- Correct problem with module visibility and ONLY.
- Eliminate internal compiler error for certain use of POINTER and nested structures.
- Eliminate inappropriate error about multiple matching procedures when generic operators are used.
- Correct parsing error of certain variable names in an assignment following a logical IF.
- Eliminate inappropriate shape mismatch for certain use of RESHAPE in a PARAMETER initialization expression.

From DFA540 FT3 T5.4-1195-46AAC to DFA540 V5.4-1265-46ABA, the following corrections have been made:

- Add support for -mixed_str_len_arg switch on UNIX, which puts character argument lengths next to their addresses in the argument list (this is the default Windows method). Add support for [NO]MIXED_STR_LEN_ARG attribute in !DEC$ ATTRIBUTES.
- Eliminate spurious error for declaration of RECORD array in a COMMON block in a MODULE.
- Correct handling of generics where several routines have an optional argument at the same position.
- In C strings, allow octals of the form \ooo to have 1, 2 or 3 octal digits and hex objects of the form \Xxx to have 1 or 2 hexadecimal characters.
- Add support for -check arg_temp_created.
- Annotations now are displayed on a per-routine basis.
- Put out type information to the debugger for module functions

From DFA540 V5.4-1265-46ABA to DFA540 V5.4-1283-46ABA, the following corrections have been made:

- Allow Sequence/Nosequence directives on fields of derived types with -hpf, whether or not numa switch is set.
- Add diagmostic "The NOSEQUENCE directive may not be specified for a component of a derived type which has the SEQUENCE attribute.".
- Issue diagnostic if NOSEQUENCE directives are applied to fields of 'sequence' derived types.
- When calculating the size of an array to be allocated, check the size computation for integer overflow.
- Fix obscure bug with OPTIONAL mask for intrinsics MAXLOC et al. The mask has an expression for the dimension bound(s).
- When a psect becomes too big, give a better error message: "Psect xxx is too large. Reduce array sizes or make arrays AUTOMATIC or ALLOCATABLE."

3.3.6 Version 5.4 Known Problems

The following known problems exist with Compaq Fortran Version 5.4:

- The following is a list of known problems for `-omp` and `-mp` parallel support in Version 5.4:
  - Global variables referenced by statement functions inside parallel regions should not reference local instances of those variable names. The following example should print 42.0 {10 times} instead of 0.0:

```
    real x,y(10)
    statement(a) = a + x

    x = 41.0
!$par parallel local(x)
    x = -1.0
!$par pdo local(i)
    do i=1,10
       y(i) = statement(1.0)
    end do
!$par end parallel
    type *,y
    end
```

- Please note that `-warn decl` gives an error level message, not a warning level message.
- When using Ladebug with certain versions of the UNIX operating system, be aware that a trailing underscore may be needed to display module variables. For example, to display variable X in module MOD, if typing `print $MOD$X$` does not display the variable's value, type:

```
print $MOD$X$_
```

3.4 New Features, Corrections, and Known Problems in Version 5.3

Version 5.3 is a minor release that includes corrections to problems discovered since Version 5.2 was released and certain new features.

The following topics are discussed:

- Section 3.4.1 (Version 5.3 ECO 02 New Features)
- Section 3.4.2 (Version 5.3 ECO 01 New Features)
- Section 3.4.3 (Version 5.3 ECO 01 HPF New Features)
- Section 3.4.4 (Version 5.3 New Features)
- Section 3.4.5 (Version 5.3 Important Information)
- Section 3.4.6 (Version 5.3 Corrections)
- Section 3.4.7 (HPF in Compaq Fortran Version 5.3)
- Section 3.4.8 (Version 5.3 Known Problems)

3.4.1 Version 5.3 ECO 02 New Features

The following new Compaq Fortran features are now supported:

- COMMON blocks can be marked read-only using the NOWRT PSECT attribute.

From DFA531 ECO 01 X5.3-1120-44A7B to DFA532 ECO 02 X5.3-1155-44A8I, the following corrections have been made:

- Eliminate internal compiler error for SIZE of a derived type component pointer array.
- Eliminate more unnecessary copies of contiguous array arguments.
- Speed up processing of EQUIVALENCE groups in COMMON blocks.
- Properly handle SIZE(A(:)(I:J))
- Implement INT8 intrinsic (already documented) - UNIX)
- Implement !DEC$ PSECT NOWRT for COMMON blocks on UNIX.
- Allow omitted OPTIONAL arguments to be passed as optional arguments to intrinsics.
- If "too many errors", suppress informational/warning messages as well.
- Allow keyword specification of arguments A10, A20, etc. to MAX/MIN.
- Eliminate internal compiler error for case of ADJUSTL with array argument.
- Eliminate internal compiler error for particular use of nested STRUCTURE.
- Disallow illegal ALLOCATED(A(:))
- Allow, as an extension for compatibility with our Fortran 77 compilers, a LOGICAL value as an argument to intrinsics which accept INTEGER, such as ABS.
- Diagnose ambiguous generic routine reference.
- When an integer constant is assigned to a variable whose KIND is smaller, don't change the KIND of the constant for future references.
- Allow IMPLICIT REAL with $.
- Eliminate internal compiler error for RESHAPE of array constructor.
- Fix parsing error for typed function declaration with MS-style argument passing specification in argument list.
- Resolve incorrect generic resolution between subroutine with no arguments (but ( ) specified as argument list) and routine with one required argument.
- Allow named constant to be used in specification expression later in same statement where it was defined.
- Give error instead of compiler abort for IF (expr) SELECT CASE
- For F90 standards checking, warn about I0, etc. in FORMAT.
- Warn about variables in BLOCK DATA subprogram that are not in a COMMON.
- Fix several compiler assertions including one from instruction scheduling at -fpe1.

3.4.2 Version 5.3 ECO 01 New Features

The following new Compaq Fortran features are now supported:

- The new INT_PTR_KIND( ) intrinsic returns the kind of an integer pointer (ie, 8 for Tru64 UNIX/Alpha).
- An optional KIND argument is now allowed on the intrinsics LEN, SHAPE, SIZE, UBOUND, LBOUND, MAXLOC, MINLOC, INDEX, LEN_TRIM, SCAN, and VERIFY. This allows these intrinsics to return a result that is other than default integer kind.

The following new `f90` command options are now supported:

- `-wsf_target` for use with `-wsf` .

  Notice the change in spelling of both `-wsf_target` and the environment variable `DECF90_WSF_TARGET`.
- `-assume [no]cc_omp` [dis]allows the use of OpenMP conditional compilation separate from `-omp`.
- Compiling `-omp -non_shared` now adds `-qlpset_r -lpset` to the link line.

Some important information to note about this release:

- An end-of-file condition on READ no longer triggers an ERR= branch - this is to conform with clearer wording in the recent standard. If an EOF condition occurs and there is no END= or IOSTAT=, an error is signalled.
- Add a NUL to the end of non-C character literals. This will not be reflected in the "length" of the constant.
- %VAL/%REF now overrides any mechanism specified in an explicit interface.
- The Compaq Extended Math Library (CXML) routines are not updated in this ECO 01 Compaq Fortran kit.

From released version V5.3-915-449BB to ECO 01 FT1 X5.3-953-44A17, the following corrections have been made:

- Improve compile time performance when USEing a module which contains a large COMMON block with many (thousands) of EQUIVALENCEd variables.
- Allow general CHARACTER expressions for the MOLD argument of the TRANSFER intrinsic.
- Correct problem that prevented scalar numeric variables and array elements which appeared in the iolist of an output statement (WRITE, etc.) from participating in uninitialized variable analysis.
- Add support for -ccdefault switch.
- In OPEN statement, if STATUS is not specified, default to UNKNOWN unless -f66 is specified, in which case default to NEW.
- Eliminate internal compiler error for case involving EQUIVALENCEd POINTER variables not in COMMON.
- Missing !DEC$ ENDIF no longer causes compiler abort.
- Correct rules for when SAVE in a PURE procedure is allowed or not.
- Correct parsing of assignment to field of RECORD whose name begins with TYPE.

- Eliminate E-level error when a BLOCK DATA subprogram name is the same as that of a COMMON block. A future revision will cause an appropriate diagnostic to appear.
- Issue clearer error message when a module name is repeated in a USE statement.
- Eliminate problem where UBOUND gave wrong value in certain cases.
- Allow substrings in left hand side of FORALL.
- Give proper error when a PARAMETER constant is CALLed.
- Give proper error when a variable is CALLed.
- In assignment statements, make sure that real constants are evaluated using the precision appropriate for their syntax (single/double/quad).

From DFA531 ECO 01 FT1 X5.3-953-44A17 to FT2 T5.3-1034-44A3V, the following corrections have been made:

- Fix problem which could cause incorrect code to be generated for certain uses of PACK, RESHAPE and UNPACK, primarily with CHARACTER arguments.
- Correct f77 driver problems, such as invalid syntax passed to cpp when a .F file was compiled.
- Speed up compilation of initialized nested structures.
- Fix problem with incorrect code generated for use of nested SPREAD intrinsic.
- Driver now understands shorter "-ieee" switch name.
- Eliminate internal compiler errors for use of defined assignment in FORALL.
- Speed up compilation where very large MODULEs are repeatedly used. Further work in this area remains.
- Correct problem with PACK intrinsic where array is array of structures with an array field with constant subscript/substring values.
- Improve generated code for SPREAD intrinsic.
- Improve generated code for array reductions.
- Improve generated code for UBOUND, LBOUND, SELECTED_INT_KIND, and SELECTED_REAL_KIND when the argument(s) are constants.
- Improve generated code for SIZEOF when bounds are constants.
- Fix UNIX driver to pass ld's position-specific qualifier within lib_list rather than cc_list.
- Eliminate internal compiler error for certain cases of integer (not F90) POINTER as a module variable.
- Reduce severity of "variable has not been used" diagnostic to "informational".
- Improve generated code for MINVAL/MAXVAL.
- Improve generated code for SIZE(A(:)).
- Improve generated code for SIZE with array argument that has vector subscripts.
- Add new INT_PTR_KIND( ) intrinsic which returns the kind of an integer pointer (either 4 or 8).

- Eliminate internal compiler error for use of allocatable array reference in a variable format expression.
- Improve generated code for ILEN.
- An end-of-file condition on READ no longer triggers an ERR= branch - this is to conform with clearer wording in the recent standard. If an EOF condition occurs and there is no END= or IOSTAT=, an error is signalled.
- Add a NUL to the end of non-C character literals. This will not be reflected in the "length" of the constant. This matches the Compaq Fortran 77 behavior, which was undocumented.
- %VAL/%REF now overrides any mechanism specified in an explicit interface.
- Generate much better code for certain array constructors (such as (/0,I=1,500000/) and allow lowerbound, upperbound and stride to have different KINDs.
- If -warn declarations is specified, do not issue diagnostic for use of IMPLICIT NONE.
- Eliminate internal compiler error for EOSHIFT with constant array argument.
- Eliminate internal compiler error for program fragment that is "ifdef-ed out".
- Report error for kind mismatch where an ac-value in an array constructor is an expression of a different kind than the other literal ac-values.
- Report error for assumed-size array passed as actual to deferred-shape array.
- Eliminate compiletime error for use of AIMAG in initialization expression when -real_size 128 is specified.
- Eliminate internal compiler error for format in a PRINT statement being an expression with concatenation.
- DFOR$PREFETCH, DFOR$PREFETCH_MODIFY, and DFOR$PREFETCH_EVICT_NEXT intrinsics now supported for Alpha processors.
- Generate correct values for a PARAMETER array whose element values are computed in an implied DO loop involving indexing into another PARAMETER array.
- Correct bad parse of .NE. as record field name.
- Allow a dummy argument that has an INTENT attribute specified to be specified in a NAMELIST list.
- Give standards warning for kind mismatch between source and pad arguments in RESHAPE.
- Long source lines are now correctly compiled when standards checking is requested (warning is still given).
- Correct problem with incorrect error given for a particular complex module renaming case.
- Allow as an extension (as does Compaq Fortran 77) a LOGICAL argument to an intrinsic which expects INTEGER.
- Correctly parse format which contains ">" characters and a variable format expression.

- Eliminate internal compiler error for a particularly complex and deeply nested structure reference in an IF.
- Don't give error for passing section of an assumed size array to a deferred shape array.
- Improve compilation speed for certain large DATA statements.
- Eliminate internal compiler error for a certain complicated FORALL.
- Correct problem with PUBLIC/PRIVATE attributes of a COMMON block in a MODULE.
- Allow (A .EQ. .NOT. B) (an extension).
- Correct problem with some COMPLEX*32 initialization expressions.
- Eliminate spurious unused variable diagnostic for variable used in pointer assignment.
- Correct problem where asking for standards checking disabled -D (/define)
- Fix a case where two NaNs sometimes compared as equal.

From DFA531 ECO 01 FT2 T5.3-1034-44A3V to final X5.3-1120 -44A7B, the following corrections have been made:

- Eliminate many unnecessary copies of assumed-shape arrays, when passed as arguments to explicit shape arrays, by keeping track of whether or not the array is known to be contiguous.
- Automatically force the alignment of a COMMON block to be at least as large as that required by the widest variable in the COMMON block.
- Fix a number of problems with WHERE inside of FORALL.
- Make defined assignment in FORALL work properly.
- Generate correct code for CSHIFT of a non-contiguous array slice.
- Allow user-defined types to be named BYTE and DOUBLECOMPLEX.
- Improve generated code for mixed COMPLEX-REAL multiplication and division.
- In array constructors with only one implied-DO, and nothing else, avoid creating an unnecessary temp.
- Allow SIZEOF(allocatable-array)
- Improve generated code for SIZEOF(array)
- Allow directory names specified with -module to be longer than 23 characters.
- Prevent incorrect collapsing of implied-DO loop in an I/O statement where there are nested loops and the implied-DO variable of one loop is used as a bound of an inner loop.
- When the error limit has been exceeded (default 30), simply suppress further messages rather than exiting the compiler. This means that the object file will get deleted appropriately, the listing file created, and subsequent source files on the command line will be processed.
- Re-allow complex constants to be passed by value. The real and imaginary parts are passed as separate arguments.
- Allow character array valued function as a format specifier.

- Allow debugging of a character function whose length is computed based on the length of a passed-length character argument.
- Solve a set of problems where the length of a character function is computed using a dummy array argument.
- The use of an INTENT(OUT) argument with LOC is now considered a "definition" for the purpose of uninitialized variable checking. Also, the use of LOC(ARRAY(N)) is now considered a "use" of ARRAY for unused variable checking.
- Eliminate internal compiler error for structure with %FILL component in module procedure.
- When standards checking is requested, do not give a warning for fixed-form source line exactly 72 columns long.
- Eliminate internal compiler error for assignment statement with variable whose name starts with an underscore. (Such names are not allowed, but a reasonable error should have been given.)
- Correct the problem where if a program unit contains two internal subroutines which both use host-association to access the same variable, the second one gets an inappropriate error.
- Eliminate internal compiler error for declaration of a derived type array with an initializer that is an implied-DO array constructor.
- Eliminate inappropriate error message for initialization expression of an implied-DO array constructor of length zero.
- When standards checking is enabled, give one warning, not three, for a !DEC$ ATTRIBUTES directive.
- Generate correct code for certain cases involving overloading of the .AND. and .OR. operators.
- Allow Variable Format Expression in a character literal when the I/O list has a subscripted array element.
- Eliminate compiler abort with incorrect program that names the enclosing program unit in an ONLY clause.
- Allow the extension syntax '101'B for a binary literal.
- Correctly handle whitespace in -omp conditional compilation lines.
- Fix problem where $INTEGER directive was not being properly handled.
- Add support for KIND= keyword in MINLOC/MAXLOC.
- Add suppprt for KIND= keyword in various string intrinsics.
- Prevent compiler abort for incorrect attempt to pass unsupported data types by value.
- Properly report invalid declaration EXTERNAL, INTEGER and recover so that remainder of program is properly parsed.
- Give standards warning for non-standard placement of NAMELIST.
- Eliminate internal compiler error for particular type of concatenation of substrings when Fortran 95 standards checking is requested.

- When converting a negative REAL value to COMPLEX, use +0.0 as the imaginary part rather than -0.0.
- Allow PARAMETER constant in ALIGN= specification of !DEC$ PSECT.
- Don't give shape mismatch for correct-shape RESHAPE in initialization expression.
- Don't give inappropriate alignment warnings for certain convoluted EQUIVALENCE/COMMON combinations.
- Eliminate internal compiler error for initialization expression which contains a constant expression in a structure constructor.
- Allow EXTERNAL :: FOO

3.4.3 Version 5.3 ECO 01 HPF New Features

The following information pertains to HPF using MPI.

Overview of HPF and MPI

The Compaq Fortran compiler now generates code that uses MPI as its message-passing library instead of PSE's HPF-specific support. The compiler provides a choice of three different variants of MPI: one for Compaq's SC supercomputer systems, one that supports shared-memory and Memory Channel interconnects, and public domain MPI for other interconnects that include Ethernet and FDDI.

It is now possible to write HPF programs that also call or use MPI (such as distributed-memory libraries that invoke MPI). The compiler's MPI runtime library uses its own private MPI "communicator" so it won't interfere with other MPI code. A new example program, /usr/examples/hpf/call_mpi.f90, illustrates this.

You enable the new MPI-based runtime library, that supports Compaq Fortran's HPF directives, by adding the `-wsf_target` option. This option, which requires an argument, belongs in the compilation and link commands.

Compiling HPF Programs for MPI

You must now specify which variant of MPI support you wish to use for HPF programs by including the option `-wsf_target` with an MPI selection (argument *target*) in the command to the f90 compiler. An example is next that selects Compaq MPI.

```
% f90 -wsf 2 -wsf_target cmpi -c lu.f90
```

An expansion of this example is next that invokes both the compiler and linker.

```
% f90 -wsf 2 -wsf_target cmpi -o lu lu.f90
```

The values of *target* in the option `-wsf_target` *target* appear next with their explanations.

target	**Explanation**
`smpi`	SC (Quadrics) MPI This MPI comes installed on SC-series systems. It works with the SC's RMS software that provides a set of commands for launching MPI jobs, scheduling these jobs on SC clusters, and performing other miscellaneous tasks.

target	**Explanation**
cmpi	Compaq MPI This MPI is a version that is specifically tuned for Alpha systems. It is distributed as a Compaq layered product. Compaq MPI supports only Memory Channel clusters and shared-memory (SMP) machines.
gmpi	Generic MPI This target is for use with MPICH V1.2.0 or other compatible libraries. MPICH is a public domain implementation of the MPI specification that is available for many platforms. You can obtain this implementation from http://www-unix.mcs.anl.gov/mpi/mpich/. MPICH V1.2.0 supports many interconnection networks including Ethernet, FDDI, and other hardware. Using Compaq Fortran and HPF with this MPI is, officially, not supported. Compaq does not guarantee support of problems caused by specifying -wsf_target gmpi. However, Compaq remains quite interested in receiving problem reports and will attempt to respond to them.

If the command to the f90 compiler includes -wsf_target *target*, then the command must also include -wsf.

Another way of specifying the version of MPI to the compiler, instead of using the option -wsf_target, is to set the environment variable DECF90_WSF_TARGET to a value in the first column of the previous table. For example, the command

```
% f90 -wsf 2 -wsf_target cmpi -c lu.f90
```

is equivalent to the commands

```
% setenv DECF90_WSF_TARGET cmpi
% f90 -wsf 2 -c lu.f90
```

If an f90 command contains -wsf_target with a value (such as cmpi) and environment variable DECF90_WSF_TARGET is set to a different value, then the value in the f90 command overrides the value of the environment variable.

Using the environment variable to select the desired MPI variant is the recommended method. This will require the fewest changes to existing scripts for building HPF programs, and will allow users generating code for more than one MPI variant to do so more easily. Compaq additionally recommends setting the environment variable in your shell initialization file (e.g. .cshrc if you use 'csh'), particularly if you usually use only one MPI variant.

A table, showing all changes to HPF-related compiler options between Fortran V5.3 and V5.3 ECO 01, is next.

Fortran V5.3	**Fortran V5.3 ECO 01**
-assume bigarrays	No change
-assume nozsize	No change
-hpf_matmul	Deleted
-nearest_neighbor	No change
-nowsf_main	No change (but currently does not work)
-pprof	Use only with -wsf_target pse
-show hpf*	No change
-show wsfinfo	No change
-wsf	No change

Fortran V5.3	Fortran V5.3 ECO 01
—	-wsf_target *target*

Linking HPF Programs with MPI

You must now specify which variant of MPI support you wish to use for HPF programs by including the option -wsf_target with an MPI selection (argument *target*) in the link command. An example is next.

```
% f90 -wsf 2 -wsf_target cmpi -o lu lu.o
```

The values of *target* come from the table in the section "Compiling HPF Programs for MPI".

If you specified generic MPI at compilation time, either by including the -wsf_target gmpi option or by setting the environment variable DECF90_WSF_TARGET to gmpi, you must specify a path to the desired generic MPI library during linking. Do this in one of these ways:

- Set the environment variable DECF90_GMPILIB to the path of the desired generic MPI library to link with
- In the link command line, include -l (possibly along with -L) with the path of the desired generic MPI library to link with. Or, explicitly add the library to the link command line.

An example of a link command for a generic MPI library is next.

```
% f90 -wsf 2 -wsf_target gmpi -o lu lu.o /usr/users/me/libmpich.a
```

In addition, you must have the Developer's Tool Kit software installed on your computer to link properly with the option -wsf_target gmpi.

Finally, programs linked with -wsf_target and an MPI target must be linked with -call_shared (which is the default); the -non_shared option does not link correctly.

Running HPF Programs Linked with MPI

The dmpirun command executes program files created with the -wsf_target cmpi option. Include the -n *n* option in the command line where *n* is the same value of -wsf *n* in the compilation command line. Or, if no value was given with the -wsf option, then set n to the desired number of peers. Also include the name of the program file.

An example is next where the compilation command line included -wsf 4 and the name of the program file is heat8.

```
% dmpirun -n 4 heat8
```

If your AlphaServer SC system is running with Revision A of the Quadrics switch, your boot log will contain the message:

```
elan0: Rev A Elite network detected - disabling adaptive routing (1)
```

To make MPI programs (including HPF programs generated with the "-wsf_ target smpi" option) run properly with Revision A hardware, you need to set the LIBELAN_GROUP_HWBCAST environment variable to DISABLE; for example, from csh:

```
% setenv LIBELAN_GROUP_HWBCAST DISABLE
```

The manpage dmpirun contains a full description of this command.

The prun command executes program files created with the -wsf_target smpi option. Include the -n *n* option in the command line where *n* is the same value of -wsf *n* in the compilation command line. Or, if no value was given with the -wsf option, then set n to the desired number of peers. Also include the name of the program file.

An example is next where the compilation command line included -wsf 4 and the name of the program file is heat8.

```
% prun -n 4 -N 4 heat8
```

The mpirun command executes program files created with the -wsf_target gmpi option. Include the -np *n* option in the command line where *n* is the same value of -wsf *n* in the compilation command line. Also include the name of the program file. The mpirun command varies according to where you installed the generic MPI.

An example is next where the compilation command line included -wsf 4 and the name of the program file is heat8.

```
% /usr/users/me/mpirun -np 4 heat8
```

In the /usr/examples/hpf directory, there is a sample script that will launch an HPF program for any variant of MPI. This script, called "hpfrun", will even determine the number of processors a source program was compiled for (if that was specified at compile time), and invoke the proper MPI run command with the number of processors specified. Portions of the script, or the entire script, may be useful for users automating the building and running of HPF programs.

Cleaning up After Running HPF Programs Linked with MPI

Execution of the dmpirun command (but not the prun and mpirun commands) may leave various system resources allocated after the program has completed. To free them, give the mpiclean with no arguments. An example is next.

```
% mpiclean
```

Changing HPF Programs for MPI

There two changes you should make to Fortran source files before compiling them for MPI. If a module contains an EXTRINSIC (HPF_LOCAL) statement and it executes on a system different from peer 0, then its output intended for stdout may (depending on the variant of MPI used) go instead to /dev/null. Change such modules or your execution commands to have the extrinsic subroutine do input/output only from peer 0.

In addition, the ability to call parallel HPF subprograms from non-parallel (Fortran or non-Fortran) main programs, is not supported in this release. For more information, see Chapter 6 of the DIGITAL High Performance Fortran 90 HPF and PSE Manual.

3.4.4 Version 5.3 New Features

The following new Compaq Fortran features are now supported:

- The following new features are now supported:
  - You can now CALL a function. In other words, a routine that is declared to be a FUNCTION can be invoked by a CALL statement. The function's return value is discarded.

- Compaq Fortran now supports COMPLEX(KIND=16), also spelled COMPLEX*32. This is a complex number composed of two 128-bit extended floating point numbers (ie, REAL(KIND=16)). Complete documentation is in the updated Compaq Fortran Language Reference Manual as well as the /usr/lib/cmplrs/fort90/decfortran90.hlp help file. Here are some highlights:
  - * COMPLEX*32 or COMPLEX(KIND=16) declares a pair of REAL*16 128-bit reals as a complex pair. It is 32 bytes big.
  - * COMPLEX*32 constants are (x,y) where at least one of x and y is a REAL*16 constant, eg, (1,2Q0).
  - * COMPLEX arithmetic supports + - * / ** . Mixed type arithmetic converts everything up to COMPLEX*32 since COMPLEX*32 is the biggest.
  - * COMPLEX*32 can be read and written in all I/O forms.
  - * Command line option "-real_size 128" forces "COMPLEX" to be COMPLEX*32 and DOUBLE COMPLEX to be COMPLEX*32. "-double_size 128" forces DOUBLE COMPLEX to be COMPLEX*32.
  - * Intrinsic generic functions that take COMPLEX now take COMPLEX*32. New specific intrinsic functions for COMPLEX*32 are CQABS, QIMAG, QCONJG, CQCOS, CQEXP, CQLOG, QREAL, CQSIN, CQSQRT, QCMPLX.
  - * Operations involving a COMPLEX*16 and a REAL*16 now produce a COMPLEX*32 result. These used to produce a COMPLEX*16 result.
- The BUFFERED= keyword has been added to the OPEN and INQUIRE statements. The default is BUFFERED='NO' for all I/O, in which case the RTL empties its internal buffer for each WRITE. If BUFFERED='YES' is specified and the device is a disk, the internal buffer will be filled, possibly by many WRITE statements, before it is emptied.

  If the OPEN has BUFFERCOUNT and BLOCKSIZE arguments, their product is the size in bytes of the internal buffer. If these are not specified, the default size is 8192 bytes. This internal buffer will grow to hold the largest single record but will never shrink.
- Character vector constructors may now have unequal length elements. The length of each element is the maximum of the element lengths. For example,

```
(/ 'ab', 'abc', 'a' /) == (/ 'ab ', 'abc', 'a  ' /)
```

- The Compaq Extended Math Library (CXML) routines are updated in the Compaq Fortran kit. See the CXML release notes in:

  /usr/opt/XMDCOM360/docs/XMD360_release_note.txt
- The following new `f90` command options are now supported:
  - `-arch ev67` and `-tune ev67` now provide instruction set support and performance tuning for the ev67 processor (21264A chip), which adds the count extension (CIX) instructions POPCNT, LEADZ, and TRAILZ.

- -align sequence allows the components of a SEQUENCEd derived type to be aligned according to the alignment rules set by the user. The default alignment rules are to align components on natural boundaries. The default is -align nosequence which means components of a SEQUENCEd derived type will be packed, regardless of the current alignment rules set by the user.
- -fast now sets -align sequence so that SEQUENCEd derived type components can be naturally aligned for improved performance.
- -fast now sets -arch host -tune host.
- -assume buffered_io turns on buffered I/O for all Fortran logical units opened for sequential writing. The default is -assume nobuffered_io.
- -Dname=value now allows a quoted string as value. For example, -DDATE="Nov 20, 1999" passes the character string Nov 20, 1999 as the value of DATE to cpp(1) and to the Compaq Fortran 90 compiler.
- -warn hpf tells the compiler to do both syntactic and semantics checking on HPF directives. The default is -warn nohpf unless -wsf is specified, in which case -warn hpf is assumed.
- -f77rtl tells the compiler to use the run-time behavior of Compaq Fortran 77 instead of Compaq Fortran 90. For example, this affects the output form for NAMELIST. The default is -nof77rtl.
- -mixed_str_len_arg tells the compiler that the hidden length passed for a character argument is to be placed immediately after its corresponding character argument in the argument list. The default is -nomixed_str_len_arg, which places the hidden lengths in sequential order at the end of the argument list.
- The file suffix .F90 now tells the driver that the file contains Fortran 90 free-form source that must be preprocessed by cpp(1). cpp(1) produces an intermediate .i90 file that is then compiled.

3.4.5 Version 5.3 Important Information

Some important information to note about this release:

- As of Compaq Fortran V5.3, the f77 command executes the Compaq Fortran 90 compiler instead of the Compaq Fortran 77 compiler. Use f77 -old_f77 to execute the Compaq Fortran 77 compiler.
- There are four INCLUDE files in /usr/include that give definitions of DFAO RTL symbols:
  - for_fpe_flags.f - flags for for_set/get_fpe(3f)
  - fordef.f - return values for the fp_class intrinsic
  - foriosdef.f - values for STAT= IO status results
  - forompdef.f - interface blocks to the omp_* routines
  - forreent.f - flags for for_set_reentrancy(3f)
- PARAMETER constants ae now alloacted in a read-only PSECT.

- Files that contain declarations that will be INCLUDEd into source code should declare data fully so that command line options used to compile the source code do not unexpectedly affect the INCLUDEd declarations. For example, if I is declared INTEGER, then using the `-i2` changes I from INTEGER*4 to INTEGER*2. If I is declared INTEGER*4, then its definition is not affected by `-i2`.

3.4.6 Version 5.3 Corrections

From version X5.2-829-4296F ECO 01 to FT1 T5.3-860-4498G, the following corrections have been made:

- Fix problem with wrong generated code if an OPTIONAL and omitted descriptor-based dummy argument is passed as an actual argument to a routine which declares that argument as OPTIONAL.
- Fix problem where ASSOCIATED did not always return the correct result for a pointer component that was transferred via pointer assignment.
- Enable display of array bounds larger than 32 bits in listing summary.
- Fix internal compiler error for certain uses of defined assignment where multiple defined operators appeared in the right-hand side of the assignment.
- Add /ALIGN=SEQUENCE (/ALIGN:SEQUENCE, -align sequence) which specifies that SEQUENCE types may be padded for alignment.
- Make the default for BLANK= in OPEN match the documentation when the -f66 (/NOF77) switch is specified, which is to default to BLANK='ZERO'. Previously, BLANK='NULL' was used regardless.
- Allow array constructors to have scalar CHARACTER source elements of varying size.
- Correct problem where a call to a routine with the C and VARYING attributes generates incorrect code.
- Make sure that -g3 does not turn off optimization.
- Fix internal compiler error for statement function which uses the function return variable of the host function.
- Fix internal compiler error for incorrect program which uses an component of a derived type variable in an automatic array bounds expression, the derived type is undefined and IMPLICIT NONE is used.
- Fix internal compiler error when RESULT variable has same name as a previously seen FUNCTION.
- Fix problem with PUBLIC/PRIVATE attributes in a particular complicated module usage.
- Eliminate spurious error message for valid generic procedure reference.
- Fix problem with DATA initialization of zero-origin arrays.
- Fix problem where compiler would not allow "# linenum" to appear in a source file if a !DEC$ or !MS$ directive was seen.
- Don't give "unused" warning for EQUIVALENCEd variable.
- Properly treat INT(n,KIND=) in an array constructor.
- Don't disable type checking for %LOC.

- Properly parse generic INTERFACE whose name begins with TO.
- When -align dcommons is used, make sure that POINTER objects in COMMON are aligned on quadword boundaries.
- Correctly parse program with IF construct whose name begins with IF.
- Fix a case where two NaNs sometimes compared as equal.
- If an attempt is made to DEALLOCATE an item which is not DEALLOCATEable, such as an array slice, a run-time error is now given. Previously, the results were unpredictable.

From version FT1 T5.3-860-4498G to FT2 T5.3-893-4499U, the following corrections have been made:

- Allocate all PARAMETER constants in a read-only PSECT.
- Ensure that locally-allocated derived-type arrays are naturally aligned.
- Generate correct code for pointer assignment of an array generated from a section of a derived type.
- Eliminate internal compiler error in certain cases with dummy argument that has OPTIONAL and INTENT(OUT) attributes.
- Flag square-bracket array constructor syntax as an extension.
- Eliminate internal compiler error for certain uses of TRANSFER.
- Properly detect ambiguous generic reference when all distinguishing arguments are OPTIONAL.
- Eliminate internal compiler error for a case involving a PRIVATE POINTER in a module.
- Eliminate spurious "this name has already been used as an external procedure" error for recursive function which returns a derived type.
- "Directive not supported on this platform" diagnostic is now informational, not warning severity.
- Allow array sections in DATA statement variable list.

From version FT2 T5.3-893-4499U to V5.3-915-449BB, the following corrections have been made:

- Eliminate access violation on some platforms for ALLOCATE of pointer in a derived type.
- Correct problem where compiler could omit putting out declaration for a routine symbol.
- Handle non-present, optional dummy arguments as third argument to INDEX, SPAN, and VERIFY.
- Generate correct code when passing character array slices as arguments.
- Fix case of contiguous array slice as first argument to TRANSFER.
- Fix INQUIRE by IOLIST of ALLOCATABLE arrays.
- Correct problem involving pointer assignment with sections of a derived type.
- Eliminate inappropriate error messages when overloading SIGN intrinsic.

- Eliminate internal compiler error when "-" defined as both unary and binary operators in separate modules.
- Eliminate spurious unused warning for pointer target.
- Implement OMP interpretation regarding DEFAULT(NONE).
- Eliminate spurious standards diagnostic for !DEC$ UNROLL.
- Correct problem with accessibility of NAMELIST names from module.
- When -real_size 64 and -double_size 128 are used, make sure DOUBLE PRECISION gets REAL*16.
- Correct evaluation of FLOAT intrinsic with -real_size 64.
- Correct problem with array constructors in format expressions.

3.4.7 HPF in Compaq Fortran Version 5.3

As in Fortran 90 Version 5.2, the HPFLIBS subset replaces the old PSESHPF subset. If you previously installed the PSESHPF subset you do not need to delete it. If you choose to delete it, delete it before you install the Fortran 90 V5.3 HPFLIBS170 subset. If you delete the PSESHPF subset after you install the Fortran HPFLIBS170 subset, you need to delete the HPFLIBS170 subset and then reinstall it. For information on using the `setld` command to check for and delete subsets, see the *Compaq Fortran Installation Guide for Tru64 UNIX Systems*.

To execute HPF programs compiled with the `-wsf` switch you must have both PSE160 and Fortran 90 Version 5.3 with the HPFLIBS170 subset installed. For this release the order of the installation is important. You must first install PSE160 and then install Fortran 90 Version 5.3 with the HPFLIBS170 subset. The HPFLIBS170 subset must be installed last. If you do this it will be properly installed.

If you also need to use the latest versions of MPI and PVM, you must install PSE180. PSE180 contains only MPI and PVM support. The support for HPF programs compiled with the `-wsf` option is only found in PSE160. Therefore you must install both versions of PSE and you must install PSE180 after PSE160.

To install Compaq Fortran with HPF and MPI and PVM, install them in the following order. The order is very important.

1. Delete any old versions that you wish to delete.
2. Install PSE160.
3. Install Compaq Fortran Version 5.3 including the HPFLIBS170 subset.
4. Install PSE180.

The HPF runtime libraries in Compaq Fortran Version 5.3 are only compatible with PSE Version 1.6. Programs compiled with this version will not run correctly with older versions of PSE. In addition, programs compiled with older compilers will no longer run correctly when linked with programs compiled with this version. Relinking is not sufficient; programs must be recompiled and relinked.

If you cannot install these in the order described, follow these directions to correct the installation:

- If you have installed Fortran Version 5.3 but are missing PSE160, then install PSE160. Delete the HPFLIBS170 subset of Fortran V5.3 and then reinstall the HPFLIBS170 subset.
- If you installed Fortran Version 5.3 first and then PSE160, then delete the HPFLIBS170 subset of Fortran V5.3. Next, reinstall the HPFLIBS170 subset.
- If you already have Fortran Version 5.3 and PSE160 installed but did not install the HPFLIBS170 subset of Fortran V5.3, then simply install the HPFLIBS170 subset.
- If you deleted any old PSESHPF subset after installing Fortran V5.3, this will also cause problems. In this case delete the HPFLIBS170 subset of Fortran Version 5.3 and then reinstall the HPFLIBS170 subset.
- If you installed PSE180 before PSE160, then delete PSE180 and reinstall it now.

For more information about installing PSE160, see the *Compaq Parallel Software Environment Release Notes*, Version 1.6.

For more information about installing PSE180, see the *Compaq Parallel Software Environment Release Notes*, Version 1.8.

3.4.8 Version 5.3 Known Problems

The following known problems exist with Compaq Fortran Version 5.3:

- The following is a list of known problems for `-omp` parallel support in Version 5.3:
  - Nested parallel regions are not supported by `-omp`. A program that contains nested parallel regions will cause the compiler to fail with an internal error.

3.5 New Features, Corrections, and Known Problems in Version 5.2

Version 5.2 is a minor release that includes corrections to problems discovered since Version 5.1 was released and certain new features.

The following topics are discussed:

- Section 3.5.1 (Version 5.2 ECO 01 New Features)
- Section 3.5.2 (Version 5.2 New Features)
- Section 3.5.3 (Version 5.2 Important Information)
- Section 3.5.4 (Version 5.2 Corrections)

3.5.1 Version 5.2 ECO 01 New Features

The following new Compaq Fortran (DIGITAL Fortran 90) features are now supported:

- IVDEP Directive

  The IVDEP directive assists the compiler's dependence analysis. It can also be specified as INIT_DEP_FWD (INITialize DEPendences ForWarD). The IVDEP directive takes the following form:

```
cDEC$ IVDEP
c  Is one of the following: C (or c), !, or *.
```

  The IVDEP directive is an assertion to the compiler's optimizer about the order of memory references inside a DO loop.

  The IVDEP directive tells the compiler to begin dependence analysis by assuming all dependences occur in the same forward direction as their appearance in the normal scalar execution order. This contrasts with normal compiler behavior, which is for the dependence analysis to make no initial assumptions about the direction of a dependence.

  The IVDEP directive must precede the DO statement for each DO loop it affects. No source code lines, other than the following, can be placed between the IVDEP directive statement and the DO statement:

  - An UNROLL directive
  - A PARALLEL DO directive (TU*X only)
  - A PDO directive (TU*X only)
  - Placeholder lines
  - Comment lines
  - Blank lines

  The IVDEP directive is applied to a DO loop in which you know that dependences are in lexical order. For example, if two memory references in the loop touch the same memory location and one of them modifies the memory location, then the first reference to touch the location has to be the one that appears earlier lexically in the program source code. This assumes that the right-hand side of an assignment statement is "earlier" than the left-hand side.

  The IVDEP directive informs the compiler that the program would behave correctly if the statements were executed in certain orders other than the sequential execution order, such as executing the first statement or block to completion for all iterations, then the next statement or block for all iterations, and so forth. The optimizer can use this information, along with whatever else it can prove about the dependences, to choose other execution orders.

  Example

  In the following example, the IVDEP directive provides more information about the dependences within the loop, which may enable loop transformations to occur:

```
!DEC$ IVDEP
      DO I=1, N
         A(INDARR(I)) = A(INDARR(I)) + B(I)
      END DO
```

In this case, the scalar execution order follows:

1. Retrieve INDARR(I).
2. Use the result from step 1 to retrieve A(INDARR(I)).
3. Retrieve B(I).
4. Add the results from steps 2 and 3.
5. Store the results from step 4 into the location indicated by A(INDARR(I)) from step 1.

IVDEP directs the compiler to initially assume that when steps 1 and 5 access a common memory location, step 1 always accesses the location first because step 1 occurs earlier in the execution sequence. This approach lets the compiler reorder instructions, as long as it chooses an instruction schedule that maintains the relative order of the array references.

- UNROLL Directive

  The UNROLL directive tells the compiler's optimizer how many times to unroll a DO loop. It takes the following form:

```
cDEC$ UNROLL [(n)]

  c  Is one of the following: C (or c), !, or *.
  n  Is an integer constant.  The range of "n" is 0 through 255.
```

  The UNROLL directive must precede the DO statement for each DO loop it affects. No source code lines, other than the following, can be placed between the UNROLL directive statement and the DO statement:

  - An IVDEP directive
  - A PARALLEL DO directive (TU*X only)
  - A PDO directive (TU*X only)
  - Placeholder lines
  - Comment lines
  - Blank lines

  If "n" is specified, the optimizer unrolls the loop "n" times. If "n" is omitted, or if it is outside the allowed range, the optimizer picks the number of times to unroll the loop.

  The UNROLL directive overrides any setting of loop unrolling from the command line.

Some important information to note about this release:

- -fast now implies "-arch host -tune host" as defaults. These can be overridden with explicit options. Note that this has an impact on redistributed programs - if they are to run on older generation processors than the compiling host, -arch, at least, must be overridden.

- The command line option "-source_listing" is not documented but it produces a listing file with a file extension of ".lis" {as opposed to "-V" which produces a .l listing file}.
- This ECO release includes the two subsets XMDLOA351 (DXML serial libraries) and XMDPLL351 (DXML parallel libraries).
- Note that there is an installation order issue with PSE: PSE160 should be installed BEFORE Fortran, since the Fortran kit has newer HPF libraries. If you are also using MPI and/or PVM, then there is also a dependency with the latest MPI/PVM kits, which are in PSE V1.8 (PSE180): the installation order needs to be PSE160 then Fortran then PSE180 OR PSE160 then PSE180 then Fortran.

From version V5.2-705-428BH to X5.2-829-4296F, the following corrections have been made:

- Correct a problem with PACK when the first argument is a two-dimensional slice of a three-dimensional array.
- Correct problem with ADJUSTL, ADJUSTR and COTAN with array element arguments.
- Fix internal compiler error for certain uses of LL* intrinsics.
- Prevent internal compiler error when the size of a return value is based on a call to a pure function with the argument to this function.
- Correct problems with nested uses of SPREAD intrinsic.
- Make ASSOCIATED return the correct result when the target is an element of a deferred-shape array.
- Correct a problem with a USE...ONLY of some symbols from an EQUIVALENCE group in a module. Previously, the compiler might generate an external reference to the wrong symbol.
- Correct a problem with EOSHIFT of a structure array with a multidimensional structure component.
- Eliminate the unnecessary use of temporary array copies in many cases.
- Add support for specific names IMVBITS, JMVBITS and KMVBITS (already documented).
- Correct a problem where calling an ELEMENTAL routine with a pointer array may give incorrect results.
- Fix transfer intrinsic where the MOLD is a character substring with non-zero base, e.g., TRANSFER(X, CH(I1:I2)).
- Fix problem where CSHIFT of an array of derived type generated bad code.
- Correct problem with pointer assignment when the right-hand-side is an array of derived types.
- Correct problems involving function return value whose size depends on properties of input arguments.
- Fix problem that caused internal compiler error with RESHAPE.
- Fix problem where IBCLR of mixed-kind arguments gave wrong answer.
- When fpp is invoked, have it also look in the current directory for include files.

- Correct problem with I/O of a slice of an assumed-size array.
- Issue error message for lexically nested parallel regions.
- In listing summary, list zero-length COMMON PSECTs.
- Eliminate spurious warning when passing a POINTER or assumed-shape array in COMMON to a routine with a compatible dummy argument declaration.
- Fix internal compiler error involving array-valued functions with entry points.
- Generate correct code for unusual (and non-standard) dummy aliasing case involving an EQUIVALENCEd variable passed as an argument.
- Fix problem with incorrect code for a call to ALLOCATE or DEALLOCATE where STAT= is specified using an array element.
- -fast now implies -arch host -tune host as defaults. These can be overridden with explicit options. Note that this has an impact on redistributed programs - if they are to run on older generation processors than the compiling host, -arch, at least, must be overridden.
- Fix internal compiler error for certain programs which CALL a function.
- Correct compiler abort with ASSOCIATED (X,(Y))
- Don’t give standards warning for ELEMENTAL PURE.
- Consider FORALL index variables "used" for -warn unused purposes.
- Disallow leading underscore in identifiers, as documented.
- Correct problem with implied DO loop in non-INTEGER array constructors in initialization expressions.
- Allow expression involving array constructors in an initialization expression.
- %LOC is treated the same as LOC for type checking purposes.
- Correct problem involving generic routine resolution.
- SEQUENCE now byte-packs fields, as the documentation says.
- Correct compiler abort with RESHAPE in initialization expression.
- Correct compiler abort for case with defined operators.
- Correct compiler abort for syntax error X(;,:)
- Give appropriate error if DO loop variable is too small for range.
- Correct compiler abort for LEN_TRIM(array) in initialization expression.
- Correct compiler abort for SIZE(non-array).
- Correct problems with ISHFT(array) in initialization expression.
- Allow SHAPE in initialization expression.
- Don’t give standards warning for use of INDEX in initialization expression.
- Consider statement function dummy argument "used" for /warn=unused.
- Correct compiler abort for invalid syntax in a Variable Format Expression (VFE).
- Correct compiler abort for module procedure with ENTRY.

- Allow full set of F95-permitted intrinsic functions in specification expressions.
- Correct compiler abort with invalid VFE in FORMAT.
- Correct problem with accessibility of MODULE symbols when two modules define the symbol but one has marked it PRIVATE.
- Correct compiler abort for certain programs when -i8 and -wsf specified.
- Correct problem with missing and duplicate alignment warnings.
- Allow repeated NULL( ) in DATA initialization when variables have different types.
- Correct spurious "shapes do not conform" error.
- Correct compiler abort for invalid program using wrong component in ASSOCIATED.
- When -names as_is specified, don't make IMPLICIT case-sensitive.
- Give standards warning for Q exponent letter in floating literals.
- Generate correct code for generic which replaces MIN or MAX.
- Give more reasonable error message when variable used as control construct name.
- Eliminate spurious message for vector-valued subscript in defined assignment.
- Give error if INTENT not properly specified for defined assignment.
- Correct internal compiler error for overloaded MAX.
- Eliminate spurious warning for FORALL.
- Give warning when INTENT(IN) argument modified in PURE FUNCTION.
- Eliminate spurious error for valid DATA with array subscript.
- Allow ORDER in RESHAPE to be non-constant.
- Fix compiler abort with RESHAPE.
- Don't give unused warning for TARGET argument used in pointer assignment.
- Properly distinguish STRUCTUREs with the same name in different CONTAINed routines.
- Allow NULL( ) to initialize a pointer to derived type.
- Incorrect warning for variable IF when -omp specified.
- Don't give unused warning for array constructor implied-DO variable.
- Allow INTRINSIC :: name (new in F95).
- Eliminate spurious standards warning for certain obscure uses of UNPACK.
- Eliminate compiler abort when transformational intrinsic used (illegally) in statement function.
- Raise limit of number of items in a FORMAT from 200 to 2048.
- Disallow invalid INTENT keywords.
- Allow CALL of a typed identifier (Compaq Fortran 77 extension).

- Correct problem where USE-associated identifiers aren't seen in certain cases involving renaming.
- Correctly evaluate CEIL intrinsic when used in a specification expression.
- Allow SIZE intrinsic to be overloaded.
- Don't issue spurious "function value has not been defined" warning for case involving ENTRY and RESULT.
- Fix internal compiler error involving defined assignment.
- Fix problem with incorrect CHARACTER initialization values and CHAR function.
- Disallow array constructor being used to initialize a scalar.
- Allow ALLOCATE/DEALLOCATE of argument to PURE SUBROUTINE.
- Fix problem for certain uses of period separators for derived type fields.
- Eliminate spurious syntax error for use-associated variable in NAMELIST.
- Eliminate spurious syntax error for certain uses of variable format expression in FMT=.
- Allow as an extension the use of a name previously seen in a CALL statement as an actual argument without an EXTERNAL statement or explicit interface.
- Eliminate spurious overflow message for MS-style base-2 constant.
- Correct problem with generic routine matching.
- Correct internal compiler error when function return value used in statement function expression.

3.5.2 Version 5.2 New Features

Version 5.2 supports the following new features :

- The following new features are now supported:
  - The f90 compiler now gives "uninitialized variable" warnings at optimization levels lower than -O4.
  - The RTL now has support for handling units *, 5 and 6 as separate units. Use of this feature, requires both RTL and compiler support. Programs must be compiled with a version of the compiler that implements this support and linked with or use a shareable RTL that implements the support. Older existing images will continue to work with the newer RTL. As a consequence of separating the units: if you were to connect unit 6 to a file, and then write to unit * - that write would produce output to the console (or stdout device). Previous to this, a write to unit * would go to the same file connected to unit 6. This new behavior is consistent to that of VMS and MS-FPS.
  - For F90, a NAMELIST input group can start with either an ampersand (&) or dollar sign ($) in any column and can be terminated by one of a slash (/), an ampersand (&) or a dollar sign($) in any column.
- The DIGITAL Extended Math Library (DXML) routines are now included in the Compaq (DIGITAL) Fortran kit.

- The following new `f90` command options are now supported:
  - `-assume gfullpath` causes the full source file path to be included in the debug information. The default is `-assume nogfullpath`.
  - `-assume [no]pthreads_lock` lets you select the kind of locking used for an unnamed critical section (when parallel processing is requested with `-mp` or `-omp`). Using the default, `-assume nopthreads_lock`, provides the fastest performance by providing a single lock for all unnamed critical sections (but does *not* lock out other process threads).

    To request more restrictive locking, specify `-assume pthreads_lock`. This locks out all other process threads in addition to all critical sections, which slows application performance.

    When using `-assume nopthreads_lock` (default), enter critical is used with the _OtsGlobalLock argument. With `-assume pthreads_lock`, enter critical is used with the _OtsPthreadLock argument.
  - `-arch ev6` generates instructions for ev6 processors (21264 chips). This option permits the compiler to generate any EV6 instruction, including instructions contained in the BWX (Byte/Word manipulation instructions) or MAX (Multimedia instructions) extension, square root and floating-point convert, and count extension. Applications compiled with this option may incur emulation overhead on ev4, ev5, ev56, and pca56 processors, but will still run correctly.

3.5.3 Version 5.2 Important Information

Some important information to note about this release:

- UNIX Virtual Memory from the Compaq Tru64 UNIX docset

  There is a new manual in V4.0D of the docset: "System Configuration and Tuning". Section 4.7.3 from that book is "Increasing the Available Address Space".

  If your applications are memory-intensive, you may want to increase the available address space. Increasing the address space will cause only a small increase in the demand for memory. However, you may not want to increase the address space if your applications use many forked processes.

  The following attributes determine the available address space for processes:

  vm-maxvas

  This attribute controls the maximum amount of virtual address space available to a process. The default value is 1 GB (1073741824). For Internet servers, you may want to increase this value to 10 GB.

  per-proc-address-space
  max-per-proc-address-size

  These attributes control the maximum amount of user process address space, which is the maximum number of valid virtual regions. The default value for both attributes is 1 GB.

  per-proc-stack-size
  max-per-proc-stack-size

These attributes control the maximum size of a user process stack. The default value of the per-proc-stack-size attribute is 2097152 bytes. The default value of the max-per-proc-stack-size attribute is 33554432 bytes. You may need to increase these values if you receive cannot grow stack messages.

per-proc-data-size
max-per-proc-data-size

These attributes control the maximum size of a user process data segment. The default value of the per-proc-data-size attribute is 134217728 bytes. The default value of the max-per-proc-data-size is 1 GB. You can use the setrlimit function to control the consumption of system resources by a parent process and its child processes. See setrlimit(2) for information.

- If you try to link `-non_shared` a parallel application that uses `-mp` or `-omp`, you must explicitly add `-lpset` in addition to the libraries `f90` links in.
- The `-noD` command switch is now available to allow symbol definitions (using `-D`) to be passed to `fpp` but not to be passed to the conditional compilation facilty inside the `f90` compiler.
- When `-arch ev6` is used, the `f90` driver will add `-qlm_ev6` before `-lm` on the `cc` command so `ld` will look for the EV6-tuned math library.
- Please note the behavior of NOWAIT reductions: each thread contributes its part, and proceeds without waiting for the final value of the reduction variable. The reduction variable's value is undefined until a synchronization operation has occurred, or the parallel region is left.
- UNIX v4.0D contains ld options to restrict library searches to shared and archived libraries. See `-no_so`, `-no_archive`, and `-so_archive` in the `ld(1)` man page.
- Use the `setld -D` option to install the software to another root directory. Everything in the installation then hangs off that root. Commands like f90 can be pointed to by PATH, the DECF90 environment variable can point to where the compiler is, `-L` can tell f90 where the RTL is, and the LD_ LIBRARY_PATH environment variable can be used to ensure that the desired version of shareable libraries are picked up at run time.

3.5.4 Version 5.2 Corrections

From version V5.1-594-3882K to FT1 T5.2-682-4289P, the following corrections have been made:

- Don't create stack temporary for character operands to ALL except when absolutely necessary.
- Add -warn argument_checking warning for mismatch between INTEGER kinds with explicit interface.
- Add -warn argument_checking warning for insufficent arguments.
- Improve display of various diagnostic messages so that the "pointer" is more appropriate.
- Fix internal compiler error when compiling a -mp or -omp program with any COMMON or EQUIVALENCED data declared in a PRIVATE, LASTPRIVATE, FIRSTPRIVATE, or REDUCTION list.

- Fix problem with TRANSFER of CHARACTER items using non-1 substring offset.
- Don't give use-before-defined warning for pointer structure assignment.
- Allow LOC(intrinsic_name).
- Allow RECORDs of empty STRUCTUREs.
- Allow repeat counts in FORMATs to be up to 2147483647.
- Always quadword-align EQUIVALENCE groups.
- Prevent internal compiler error with very long list of -D definitions.
- Correct problem relating to use of an AUTOMATIC array in a parallel region.
- Allow contained function result to have dimension bounds depend upon size of one of its array arguments.
- Eliminate inappropriate argument mismatch warning with record structures when -wsf is specified. Add support for -assume gfullpath, which causes the full source file path to be included in the debug information.
- If -check bounds is in effect, don't optimize implied-DO in I/O as this can prevent bounds checking from occurring.
- Eliminate inappropriate use-before-defined warnings when passing array slices.
- Improve generated code when calling routines with INTENT(IN). Prevent an output statement (WRITE, etc.) from inhibiting use-before-defined warnings.
- Improve generated code when calling intrinsic functions.
- -fast or -math_library fast implies -check nopower.
- Fortran 90 interpretation 100 - ASSOCIATED of two zero-sized arrays always returns .FALSE..
- Eliminate internal compiler error for LOC(character-parameter-constant)
- Eliminate "text handle table overflow" errors for certain programs that had very large and complicated single statements (e.g., DATA).
- Allow structure field names which are the same as relational operators.
- In pointer assignment, where the right-hand-side is a structure constructor, enforce the standard's requirement that the constructor expression be an allowable target.
- Allow a module procedure as an actual argument.
- Eliminate inappropriate error about use of PRIVATE type declared later in the module.
- Eliminate parsing error where a KIND specifier is continued across multiple source lines.
- Eliminate parsing error involving an assignment to a variable whose name begins with "PARAMETER".
- When passing an element of a named array constant as an actual argument, make sure that sequence association works as if it had been a variable.
- Correct problem with visibility of inherited identifier.

- Eliminate internal compiler error for PARAMETER declaration where the constant value is an undefined identifier.
- Eliminate internal compiler error involving a statement function having the same name as another routine in the same compilation.
- Make severity of -warnings declarations diagnostics warning instead of error.
- Eliminate internal compiler error when all source is conditionalized away.
- Eliminate internal compiler error for certain programs which use TRANSFER in a PARAMETER declaration.
- Allow a tab character in a FORMAT.
- Assume INTEGER type for bit constants where required.
- Don't sign extend result of ICHAR in a PARAMETER definition.
- Eliminate internal compiler error for certain programs using functions with mask arguments.
- Make !DEC$ATTRIBUTES (no space) work in any column in fixed-form.
- Give proper error instead of internal compiler error when QFLOAT used on platforms that don't support REAL*16.
- Don't consider a DECODE to modify the buffer argument for purposes of INTENT.
- Eliminate internal compiler error for certain programs when -assume dummy_aliases is in effect.
- Correct problem with certain programs using STRUCTUREs with %FILL fields.
- When -real_size 64 is in effect, intrinsics with explicitly REAL*4 or COMPLEX*8 arguments are no longer inappropriately promoted to REAL*8/COMPLEX*16.
- Do not cause internal compiler error for reference to undefined user operator.
- Allow use of an array-constructor's implied DO variable in a specification expression.
- Allow SIZE argument to be omitted to IISHFTC, JISHFTC, KISHFTC.
- Make result type of IBSET, IBCLR, IBITS, etc. be type of the first argument.
- Allow up to 256 arguments to an intrinsic function (e.g., MAX, MIN) in a specification expression - the previous limit was 8.
- Give error for passing an array section with vector subscript to INTENT(INOUT) or INTENT(OUT) argument.
- Fix internal compiler error for use in the length specification expression for a function LEN(concatenation) where one of the concatenation arguments is a passed-length argument to the function being declared.
- Fix internal compiler error for use in the length specification expression for a function LEN(TRIM(arg)) where arg is a passed-length argument to the function being declared.
- Treat a negative declared length for a CHARACTER variable as if it were zero.

- Properly parse "ELSE IFCONSTRUCT" where CONSTRUCT is a construct name.
- Give an error when an AUTOMATIC variable is DATA initialized.
- Properly propagate (or not) PRIVATE attribute for nested USE.
- Eliminate undeserved argument conformance error in certain cases involving WHERE masks.
- Ensure that the return kind of ICHAR is "default integer", no matter what kind that is (due to integer_size switch).
- Fix internal compiler error for type constructor with string argument for numeric element.
- Fix internal compiler error when an INTERFACE TO block has certain syntax errors.
- Correctly parse non-standard 'n syntax for REC= in I/O statement when the I/O list contains a quoted literal.
- Fix problem relating to ONLY and nested USE.
- Make variables whose names begin with $ have implicit INTEGER type.
- Allow $ in the range for IMPLICIT (sorts after Z).
- If a program has multiple USE statements where the module files cannot be found, give error messages for each of them.
- Allow SIZEOF in EQUIVALENCE array index.
- Fix internal compiler error with certain array initializers containing an implied DO.
- Accept F95-style reference to MAXVAL, MINVAL, MAXLOC, MINLOC with a mask as a second non-keyword argument.
- Accept F95-style reference to PRODUCT and SUM with a mask as a second non-keyword argument.
- Don't give inappropriate alignment warnings for REAL*16 variables in COMMON.
- Don't give error message for empty FORALL statement body.
- Allow FORALL to be nested 7 deep (previous limit was 3).
- Correctly parse certain complex instances of named FORALL.
- Allow RESULT of ENTRY to have same name as host FUNCTION.
- Demote diagnostic for not using all active combinations of FORALL index names from error to warning.
- Eliminate inappropriate error for certain uses of intrinsic functions in a specification expression.
- Eliminate internal compiler error for a peculiar (and erroneous) case of a USE of a NAMELIST whose group contains a variable inherited from another module but which isn't visible due to an ONLY list.
- Make OPTIONS /EXTEND_SOURCE persistent across an INCLUDE.
- Add support for defined assignment statement from within a WHERE statement.

- Allow a function result length to be computed using a field of an array element, where the array is a derived type passed as a dummy argument.
- Fix problem with functions returning complex/doublecomplex.

From version FT1 T5.2-682-4289P to FT2 T5.2-695-428AU, the following corrections have been made:

- Allow an ALLOCATABLE variable to be PRIVATE in a parallel scope.
- Support ISHC for INTEGER*8.
- Correct problem with overlapping CHARACTER assignment in FORALL.
- Correct debug information for CHARACTER POINTERs.
- Correct problems with ISHFTC which can cause alignment errors.
- Correct problem with FORALL and WHERE with non-default integer size.
- Don't issue spurious UNUSED warning for argument whose interface comes from a MODULE.
- Fix internal compiler error for invalid IMPLICIT syntax.
- Eliminate inappropriate type mismatch error for certain cases of references to a generic procedure with a procedure argument.
- Allow use of . field separator in addition to % in ALLOCATE/DEALLOCATE.
- Give warning of unused variable in module procedure when appropriate.
- Do not allow a non-integer/logical expression in a logical IF.
- Fix another case of recognizing a RECORD field that has the same name as a relational operator.
- Correct compiler failure for CMPLX(R8,R8) when real_size=64 is in effect.
- Allow gaps in keyword names in MAX/MIN, for example MAX(A1=x,A4=y).
- Correct compiler failure when a COMPLEX array is initialized with a REAL array constructor.
- Correct compiler failure when the CHAR intrinsic is used in an initialization expression.
- Correct compiler failure ("possible out of order or missing USE") in certain uses of nested MODULEs and ONLY.
- Show correct source pointer for syntax error in declaration.

From version FT2 T5.2-695-428AU to V5.2-705-428BH, the following corrections have been made:

- The compiler now accepts a new DEFAULT keyword on the !DEC$ ATTRIBUTES directive. This tells the compiler to ignore any compiler options that change external routine or COMMON block naming or argument passing conventions, and uses just the other attributes specified (if any). The options which this affects are -names and -assume underscore.
- Avoid giving a spurious "Inconsistent THREADPRIVATE declaration of common block" error if one COMMON block has a name which is an initial substring of another and one of them is named in a THREADPRIVATE directive.
- Prevent FUSE XREF from dying when !DEC$ ATTRIBUTES is used.

- Add support for -source_listing option. The listing file has the extension .lis.
- The f66 option now establishes OPEN defaults of STATUS='NEW' and BLANK='ZERO'.
- Correct compiler failure with RESHAPE and SHAPE used in an initialization expression.
- Eliminate spurious error when a defined operator is used in a specification expression
- Correct compiler failure when undefined user-defined operator is seen.
- Eliminate spurious error when component of derived type named constant is used in a context where a constant is required.
- Correct problem with host association and contained procedure.
- Correct compiler failure with WHERE when non-default integer_size is in effect.

3.6 High Performance Fortran (HPF) Support in Version 5.2

Compaq Fortran (DIGITAL Fortran 90) Version 5.2 supports the entire High Performance Fortran (HPF) Version 2.0 specification with the following exceptions:

- Nested FORALL statements
- WHERE statements within FORALL statements
- Passing CYCLIC(N) arguments to EXTRINSIC (HPF_LOCAL) routines. See Section 3.6.5.3.
- Accessing non-local data (other than arguments) within PURE functions in FORALL statements
- SORT_UP library procedure
- SORT_DOWN library procedure

In addition, the compiler supports many HPF Version 2.0 approved extensions including:

- Extrinsic (HPF_LOCAL) routines
- Extrinsic (HPF_SERIAL) routines
- Mapping of derived type components
- Pointers to mapped objects
- Shadow-width declarations
- All HPF_LOCAL_LIBRARY routines (except LOCAL_BLKCNT, LOCAL_ LINDEX, and LOCAL_UINDEX). Other exceptions are the approved extensions to HPF_LOCAL_LIBRARY routines.
- ON directive within INDEPENDENT loops
- RESIDENT directive used with INDEPENDENT loops

3.6.1 Optimization

This section contains release notes relevant to increasing code performance. You should also refer to Chapter 7 of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual* for more detail.

3.6.1.1 The -fast Compile-Time Option

To get optimal performance from the compiler, use the `-fast` option if possible.

Use of the `-fast` option is not permitted in certain cases, such as programs with zero-sized data objects or with very small nearest-neighbor arrays.

For More Information:

- On the cases where use of `-fast` is not permitted, see the "Optimizing" and "Compiling" chapters of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*.

3.6.1.2 Non-Parallel Execution of Code

The following constructs are not handled in parallel:

- Reductions with non-constant DIM argument.
- CSHIFT, EOSHIFT and SPREAD with non-constant DIM argument.
- Some array-constructors
- PACK, UNPACK, RESHAPE
- *xxx*_PREFIX, *xxx*_SUFFIX, GRADE_UP, GRADE_DOWN
- In the current implementation of Compaq Fortran 95/90, all I/O operations are serialized through a single processor; see Chapter 7 of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual* for more details
- Date and time intrinsics, including DATE_AND_TIME, SYSTEM_CLOCK, DATE, IDATE, TIME, and SECNDS

If an expression contains a non-parallel construct, the entire statement containing the expression is executed in a nonparallel fashion. The use of such constructs can cause degradation of performance. Compaq recommends avoiding the use of constructs to which the above conditions apply in the computationally intensive kernel of a routine or program.

3.6.1.3 INDEPENDENT DO Loops Currently Parallelized

Not all INDEPENDENT DO loops are currently parallelized. It is important to use the `-show hpf` or `-show hpf_indep` compile-time option, which will give a message whenever a loop marked INDEPENDENT is not parallelized.

Currently, a nest of INDEPENDENT DO loops is parallelized whenever the following conditions are met:

- When INDEPENDENT DO loops are nested, the NEW keyword must be used to assert that all loop variables (except the outer loop variable) are NEW. It is recommended that the outer DO loop variable be in the NEW list, as well.
- The loop does not contain any of the constructs listed in Section 3.6.1.2 that cause non-parallel execution.
- Each subscript of each array reference must either
  - contain no references to INDEPENDENT DO loop variables, or

- contain one reference to an INDEPENDENT DO loop variable and the subscript expression is an affine function of that DO loop variable.

- At least one array reference must reference all the independent loops in a nest of independent loops.
- The compiler must be able to prove that loop nest either
  - requires no inter-processor communication, or
  - can be made to require no inter-processor communication with compiler-generated copyin/copyout code around the loop nest.
- Any reductions in an interior (i.e. any but the outer) loop may use an INDEPENDENT DO index as a subscript only if that index represents a serially distributed dimension of the array. An exception to this is the index of the outermost DO loop, which may be used as a subscript even if it represents a non-serially distributed array dimension.
- There must not be any assignments to scalars, except for NEW or reduction variables.
- Any procedure call inside an INDEPENDENT DO loop must either be PURE, or be encapsulated in an ON HOME RESIDENT region (see Section 3.6.5.6).

When the entire loop nest is encapsulated in an ON HOME RESIDENT region, then only the first two restrictions apply.

For More Information:

- On enclosing INDEPENDENT DO loops in an ON HOME RESIDENT region, see Section 3.6.5.6

3.6.1.4 Nearest-Neighbor Optimization

The following is a list of conditions that must be satisfied in an array assignment, FORALL statement, or INDEPENDENT DO loop in order to take advantage of the nearest-neighbor optimization:

- Relevant arrays with the POINTER or TARGET attributes must have shadow edges explicitly declared with the SHADOW directive.
- The arrays involved in the nearest-neighbor style assignment statements should not be module variables or variables assigned by USE association. However, if both the actual and all associated dummies are assigned a shadow-edge width with the SHADOW directive, this restriction is lifted.
- A value must be specified for the `-wsf` option on the command line.
- Some interprocessor communication must be necessary in the statement.
- Corresponding dimensions of an array must be distributed in the same way (though they can be offset using an ALIGN directive). If the `-nearest_neighbor` flag's optional *nn* field is used to specify a maximum shadow-edge width, only constructs with a subscript difference (adjusted for any ALIGN offset) less than or equal to the value specified by *nn* will be recognized as nearest neighbor. For example, the assignment statement `(FORALL (i=1:n) A(i) = B(i-3))` has a subscript difference of `3`. In a program compiled with the flag `-nearest_neighbor 2`, this assignment statement would not be eligible for the nearest neighbor optimization.
- The left-hand side array must be distributed BLOCK in at least one dimension.

- The arrays must not have complicated subscripts (no vector-valued subscripts, and any subscripts containing a FORALL index must be affine functions of *one* FORALL index; further, that FORALL index must not be repeated in any other subscript of a particular array reference).
- Statements with scalar subscripts are eligible only if that array dimension is (effectively) mapped serially.
- Subscript triplet strides must be known at compile time and be greater than 0.
- The arrays must be distributed BLOCK or serial (`*`) in each dimension.

Compile with the `-show hpf` or `-show hpf_nearest` switch to see which lines are treated as nearest-neighbor.

Nearest-neighbor communications are not profiled by the `pprof` profiler. See the section about the `pprof` Profile Analysis Tool in the Parallel Software Environment (PSE) Version 1.6 release notes.

For More Information:

- On profiling nearest-neighbor computations, see the section about the `pprof` Profile Analysis Tool in the Parallel Software Environment (PSE) Version 1.6 release notes.
- On using EOSHIFT for nearest-neighbor computations, see Section 3.6.1.6

3.6.1.5 Widths Given with the SHADOW Directive Agree with Automatically Generated Widths

When compiler-determined shadow widths don't agree with the widths given with the SHADOW directive, less efficient code will usually be generated.

To avoid this problem, create a version of your program without the SHADOW directive, and compile with the `-show hpf` or `-show hpf_near` option. The compiler will generate messages that include the sizes of the compiler-determined shadow widths. Make sure that any widths you specify with the SHADOW directive match the compiler-generated widths.

3.6.1.6 Using EOSHIFT Intrinsic for Nearest Neighbor Calculations

In the current compiler version, the compiler does not always recognize nearest-neighbor calculations coded using EOSHIFT. Also, EOSHIFT is sometimes converted into a series of statements, only some of which may be eligible for the nearest neighbor optimization.

To avoid these problems, Compaq recommends using CSHIFT or FORALL instead of EOSHIFT if these alternatives meet the needs of your program.

3.6.2 New Features

This section describes the new HPF features in this release of Compaq Fortran.

3.6.2.1 RANDOM_NUMBER Executes in Parallel

The RANDOM_NUMBER intrinsic subroutine now executes in parallel for mapped data. The result is a significant decrease in execution time.

3.6.2.2 Improved Performance of TRANSPOSE Intrinsic

The TRANSPOSE intrinsic will execute faster for most arrays that are mapped either * or BLOCK in all dimensions.

3.6.2.3 Improved Performance of DO Loops Marked as INDEPENDENT

Certain induction variables are now recognized as affine functions of the INDEPENDENT DO loop indices, thus meeting the requirements listed in Section 3.6.1.3. Now, the compiler can parallelize array references containing such variables as subscripts. An example is next.

```
!     Compiler now recognizes a loop as INDEPENDENT because it
!        knows that variable k1 is k+1.
      PROGRAM gauss
      INTEGER, PARAMETER    :: n = 1024
      REAL, DIMENSION (n,n) :: A
      !HPF$ DISTRIBUTE A(*,CYCLIC)

      DO k = 1, n-1
         k1 = k+1
         !HPF$ INDEPENDENT, NEW(i)
         DO j = k1, n
            DO i = k1, n
               A(i,j) = A(i,j) - A(i,k) * A(k,j)
            ENDDO
         ENDDO
      ENDDO
      END PROGRAM gauss
```

3.6.3 Corrections

This section lists problems in previous versions that have been fixed in this version.

- In programs compiled with the `-wsf` option, pointer assignments inside a FORALL did not work reliably. In many cases, incorrect program results occurred.
- The ASSOCIATED intrinisc sometimes returned incorrect results in programs compiled with the `-wsf` compile-time option.
- GRADE_UP and GRADE_DOWN were not stable sorts.

3.6.4 Known Problems

3.6.4.1 "Variable used before its value has been defined" Warning

The compiler may inappropriately issue a "Variable is used before its value has been defined" warning. If the variable named in the warning does not appear in your program (e.g. var$0354), you should ignore the warning.

3.6.4.2 Mask Expressions Referencing Multiple FORALL Indices

FORALL statements containing mask expressions referencing more than seven FORALL indices do not work properly.

3.6.5 Unsupported Features

This section lists unsupported features in this release of Compaq Fortran.

3.6.5.1 SMP Decomposition (OpenMP) not Currently Compatible with HPF

Manual decomposition directives for SMP (such as the OpenMP directives enabled with the `-omp` option, or the directives enabled with the `-mp` option) are not currently compatible with the `-wsf` option.

3.6.5.2 Command Line Options not Compatible with the -wsf Option

The following command line options may not be used with the `-wsf` option:

- The -feedback and -cord options are not compatible, since they require the use of -p, which is not compatible with `-wsf`.
- -double_size 128
- -gen_feedback
- -p, -p1, -pg (use -pprof instead)
- -fpe1, -fpe2, -fpe3, -fpe4
- -om
- -mp
- -omp

3.6.5.3 HPF_LOCAL Routines

Arguments passed to HPF_LOCAL procedures cannot be distributed CYCLIC(*n*). Furthermore, they can have neither the inherit attribute nor a transcriptive distribution.

Also, the following procedures in the HPF Local Routine Library are not supported in the current release:

- ACTIVE_NUM_PROCS
- ACTIVE_PROCS_SHAPE
- HPF_MAP_ARRAY
- HPF_NUMBER_MAPPED
- LOCAL_BLKCNT
- LOCAL_LINDEX
- LOCAL_UINDEX

3.6.5.4 SORT_UP and SORT_DOWN Functions

The SORT_UP and SORT_DOWN HPF library procedures are not supported. Instead, use GRADE_UP and GRADE_DOWN, respectively.

3.6.5.5 Restricted Definition of PURE

In addition to the restrictions on PURE functions listed in the Fortran 95 language standard and in the *High Performance Fortran Language Specification*, Compaq Fortran adds the additional restriction that PURE functions must be *resident*. "Resident" means that the function can execute on each processor without reading or writing any data that is not local to that processor.

Non-resident PURE functions are not handled. They will probably cause failure of the executable at run-time if used in FORALLs or in INDEPENDENT DO loops.

3.6.5.6 Restrictions on Procedure Calls in INDEPENDENT DO and FORALL

In order to execute in parallel, procedure calls from FORALL and DO INDEPENDENT constructs must be *resident*. "Resident" means that the function can execute on each processor without reading or writing any data that is not local to that processor. The compiler requires an explicit assertion that all procedure calls are resident. You can make this assertion in one of two ways:

1. by labeling every procedure called by the FORALL or INDEPENDENT DO loop as PURE
2. by encapsulating the entire body of the loop in an ON HOME RESIDENT region.

Because of the restricted definition of PURE in Compaq Fortran (see Section 3.6.5.5), the compiler interprets PURE as an assertion by the program that a procedure is resident.

Unlike procedures called from inside FORALLs, procedures called from inside INDEPENDENT DO loops are not required to be PURE. To assert to the compiler that any non-PURE procedures called from the loop are resident, you can encapsulate the entire body of the loop in an ON HOME RESIDENT region.

If you incorrectly assert that a procedure is resident (using either PURE or ON HOME RESIDENT), the program will either fail at run time, or produce incorrect program results.

Here is an example of an INDEPENDENT DO loop containing an ON HOME RESIDENT directive and a procedure call:

```
!HPF$ INDEPENDENT
DO i = 1, 10
   !HPF$ ON HOME (B(i)), RESIDENT  BEGIN
   A(i) = addone(B(i))
   !HPF$ END ON
END DO
.
.
.

CONTAINS
  FUNCTION addone(x)
    INTEGER, INTENT(IN) :: x
    INTEGER addone
    addone = x + 1
  END FUNCTION addone
```

The ON HOME RESIDENT region does not impose any syntactic restrictions. It is merely an assertion that inter-processor communication will not actually be required at run time.

For More Information:

- On the requirements for parallel execution of INDEPENDENT DO loops, see Section 3.6.1.3

3.6.5.7 Restrictions on Routines Compiled with -nowsf_main

The following are restrictions on dummy arguments to routines compiled with the `-nowsf_main` compile-time option:

- The dummy must not be assumed-size
- The dummy must not be of type CHARACTER*(*)
- The dummy must not have the POINTER attribute
- %LOC must not be applied to distributed arguments

Failure to adhere to these restrictions may result in program failure, or incorrect program results.

3.6.5.8 RAN and SECNDS Are Not PURE

The intrinsic functions RAN and SECNDS are serialized (not executed in parallel). As a result, they are not PURE functions, and cannot be used within a FORALL construct or statement.

3.6.5.9 Nonadvancing I/O on stdin and stdout

Nonadvancing I/O does not work correctly on `stdin` and `stdout`. For example, this program is supposed to print the prompt ending with the colon and keep the cursor on that line. Unfortunately, the prompt does not appear until after the input is entered.

```
PROGRAM SIMPLE

        INTEGER STOCKPRICE

        WRITE (6,'(A)',ADVANCE='NO') 'Stock price1   : '
        READ  (5, *) STOCKPRICE

        WRITE (6,200) 'The number you entered was ', STOCKPRICE
200     FORMAT(A,I)

END PROGRAM SIMPLE
```

The work-around for this bug is to insert a CLOSE statement after the WRITE to `stdout`. This effectively flushes the buffer.

```
PROGRAM SIMPLE

        INTEGER STOCKPRICE

        WRITE (6,'(A)',ADVANCE='NO') 'Stock price1   : '
        CLOSE (6)                       ! Add close to get around bug
        READ  (5, *) STOCKPRICE

        WRITE (6,200) 'The number you entered was ', STOCKPRICE
200     FORMAT(A,I)

END PROGRAM SIMPLE
```

3.6.5.10 WHERE and Nested FORALL

The following statements are not currently supported:

- WHERE statements inside FORALLs
- FORALLs inside WHEREs
- Nested FORALL statements

When nested DO loops are converted into FORALLs, nesting is ordinarily not necessary. For example,

```
DO x=1, 6
  DO y=1, 6
    A(x, y) = B(x) + C(y)
  END DO
END DO
```

can be converted into

```
FORALL (x=1:6, y=1:6)  A(x, y) = B(x) + C(y)
```

In this example, both indices (x and y) can be defined in a single FORALL statement that produces the same result as the nested DO loops.

In general, nested FORALLs are required only when the outer index is used in the definition of the inner index. For example, consider the following DO loop nest, which adds 3 to the elements in the upper triangle of a 6×6 array:

```
DO x=1, 6
  DO y=x, 6
    A(x, y) = A(x, y) + 3
  END DO
END DO
```

In Fortran 95/90, this DO loop nest can be replaced with the following nest of FORALL structures:

```
FORALL (x=1:6)
  FORALL (y=x:6)
    A(x, y) = A(x, y) + 3
  END FORALL
END FORALL
```

However, nested FORALL is not currently supported in parallel (i.e. with the `-wsf` option).

A work-around is to use the INDEPENDENT directive:

```
      integer, parameter :: n=6
      integer, dimension (n,n) :: A
!hpf$ distribute A(block,block)

      A = 8

      !hpf$ independent, new(i)
      do j=1,n
         !hpf$ independent
         do i=j,n
            A(i,j) = A(i,j) + 3
         end do
      end do

      print "(6i3)", A

      end
```

All three of these code fragments would convert a matrix like this:

$$\begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 \\ 8 & 8 & 8 & 8 & 8 & 8 \end{bmatrix}$$

into this matrix:

$$\begin{bmatrix} 11 & 11 & 11 & 11 & 11 & 11 \\ 8 & 11 & 11 & 11 & 11 & 11 \\ 8 & 8 & 11 & 11 & 11 & 11 \\ 8 & 8 & 8 & 11 & 11 & 11 \\ 8 & 8 & 8 & 8 & 11 & 11 \\ 8 & 8 & 8 & 8 & 8 & 11 \end{bmatrix}$$

3.6.6 Obsolete Features Deleted

3.6.6.1 GLOBAL_TO_PHYSICAL and GLOBAL_LBOUNDS are Deleted

The following obsolete HPF Local Library routines have been deleted:

- GLOBAL_TO_PHYSICAL
- GLOBAL_LBOUNDS

3.6.7 Miscellaneous

This section contains miscellaneous release notes relevant to HPF.

3.6.7.1 What To Do When Encountering Unexpected Program Behavior

This section gives some guidelines about what to do when your program displays unexpected behavior at runtime. The two most common problems are incorrect programs that either segmentation fault or hang at runtime.

Before attempting to debug parallel HPF programs, it is important to verify first that the program runs correctly when compiled without the `-wsf` command line switch.

When the problem occurs only when compiled with the `-wsf` switch, the best way to debug these programs is to execute them with the `-debug` command line switch.

In addition, programs with zero sized arrays which were compiled with `-fast` or `-assume nozsize` may behave erratically or fail to execute.

3.6.7.1.1 Incompatible or Incomplete Libraries Installed If your program displays unexpected behavior at runtime, your system might have incomplete or incompatible libraries installed. You must have PSE160 installed on your system to execute programs compiled with the `-wsf` switch. PSE180 is not sufficient. In addition, for this release, you must have first installed PSE160. Then you must have installed Fortran V5.2, including the HPFLIBS170 subset.

Choose one of the following options to fix an incorrect installation:

- If you have installed Fortran V5.2 but are missing PSE160, then install PSE160. Delete the HPFLIBS170 subset of Fortran V5.2 and then reinstall the HPFLIBS170 subset.
- If you installed Fortran V5.2 first and then PSE160, then delete the HPFLIBS170 subset of Fortran V5.2. Next, reinstall the HPFLIBS170 subset.
- If you already have Fortran V5.2 and PSE160 installed but did not install the HPFLIBS170 subset of Fortran V5.2, then simply install the HPFLIBS170 subset.
- If you deleted any old PSESHPF subset after installing Fortran V5.2, this will also cause problems. In this case delete the HPFLIBS170 subset of Fortran V5.2 and then reinstall the HPFLIBS170 subset.

- If you have installed PSE180 but not PSE160, then begin to correct this situation by deleting PSE180. Install PSE160. Next, reinstall PSE180. You need both PSE160 and PSE180; PSE180 must be installed last. Finish by deleting the HPFLIBS170 subset of Fortran V5.2 and then reinstalling the HPFLIBS170 subset.

For more information about installing PSE160, see the *Compaq Parallel Software Environment Release Notes*, Version 1.6.

For more information about installing PSE180, see the *Compaq Parallel Software Environment Release Notes*, Version 1.8.

3.6.7.1.2 Segmentation Faults When a program segmentation faults at runtime it can be confusing because it may look like the program executed, even though no output is produced. The PSE does not always display an error message when the return status of the executed program is non zero. In particular, if the program segmentation faults it does not display an error message, the program just stops. In this example, program "bad" gets a segmentation fault at runtime.

```
# bad -peers 4
#
```

To see the execution status, type this `csh` command (other shells require different commands):

```
# echo $status
```

A status of `-117` indicates a segmentation fault. See the section about known problems in the Parallel Software Environment (PSE) Version 1.6 release notes.

Alternatively, you can run the program in the debugger. This is better because it shows what went wrong on each peer. To do this, use the `-debug` command line switch.

```
# bad -peers 4 -debug
```

See Chapter 9 of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual* for more information.

Note that some correct programs may segmentation fault at runtime due to lack of stack space and data space. See Section 3.6.7.2 for further details.

3.6.7.1.3 Programs that Hang If your program hangs at runtime, rerun it in the debugger. You can type `<CTRL>/c` in the debugger to get it to stop. Then look at the stack frames to determine where and why the program is hanging. Programs can hang for many reasons. Some of the more common reasons are:

- Incorrect or incorrectly-spelled HPF directives
- Incorrect usage of extrinsic routines
- Templates not large enough
- Incorrect interfaces
- Missing interface blocks
- Allocatables aligned incorrectly
- Arrays aligned outside of template bounds
- Exceeding the available stack or data space (see Section 3.6.7.2)

It is always best to compile, run, and debug the program without the -wsf switch first to verify program correctness. Since it is easier to debug scalar programs than parallel programs, this should always be done first.

3.6.7.1.4 Programs with Zero Sized Arrays Programs with zero sized arrays should not be compiled with the -fast or the -assume nozsize command line options; see Chapter 8 in the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*. If you incorrectly compile this way there are several different types of behavior that might occur. The program might return an error status of -122 or -177 or 64. It might also hang (or timeout when the -timeout switch is used). Try compiling the program without these options and execute it to see if it works correctly. If it does, there is most likely a zero-sized array in the program.

3.6.7.2 Stack and Data Space Usage

Exceeding the available stack or data space on a processor can cause the program execution to fail. The failure takes the form of a segmentation violation, which results in an error status of -117. (See the section about known problems in the Parallel Software Environment (PSE) Version 1.6 release notes.) This problem can often be corrected by increasing the stack and data space sizes or by reducing the stack and data requirements of the program. The following csh commands increase the sizes of the stack and data space up to system limits (other shells require different commands):

```
limit stacksize unlimited
limit datasize  unlimited
```

If your system limits are not sufficient, contact your system administrator, and request that maxdsiz (the data space limit) and/or maxssiz (the stack limit) be increased.

3.6.7.3 Non-"-wsf" main programs

The ability to call parallel HPF subprograms from non-parallel (Fortran or non-Fortran) main programs, is supported in this release. For more information, see Chapter 6 of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*.

3.6.7.4 Using "-std" Disables HPF Directive Checking

Normally, all HPF directives are checked for syntactic and semantic correctness regardless of whether or not the -wsf switch is specified. To disable this checking, specify the -std option.

3.6.7.5 Use the Extended Form of HPF_ALIGNMENT

Due to an anomaly in the *High Performance Fortran Language Specification*, the extended version of the HPF_ALIGNMENT library routine (*High Performance Fortran Language Specification* V.2 Section 12.2) is incompatible with the standard version (*High Performance Fortran Language Specification* V.2 Section 7.7).

In particular, the DYNAMIC argument, which is valid only in the extended version, is not the final argument in the argument list.

Because each compiler vendor must choose to implement only one version of this library routine, programs that use this routine are not portable from one compiler to another unless keywords are used for each of the optional arguments.

Compaq chooses to support the extended version of this library routine.

3.6.7.6 EXTRINSIC(SCALAR) Changed to EXTRINSIC(HPF_SERIAL)

EXTRINSIC(SCALAR) was renamed to EXTRINSIC(HPF_SERIAL) to be compatible with Versions 1.1 and later of the *High Performance Fortran Language Specification*. EXTRINSIC(SCALAR) continues to be supported in this release, but may not be supported in future releases.

3.6.8 Example Programs

The `/usr/examples/hpf` directory contains example Fortran programs. Most of these programs are referred to in the HPF Tutorial section of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*. Others are just there to show examples of HPF code and PVM code. The provided makefile can be used to compile all these programs.

- `heat_example.f90` solves a heat flow distribution problem. It is referred to by the Solving Nearest Neighbor Problems section of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*.
- `io_example.f90` implements a network striped file. It is referred to by the Network Striped Files chapter of the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*. This program is a good example of how to use EXTRINSIC(HPF_LOCAL) routines.
- `lu.f90` implements a LU Decomposition. It is referred to by the LU Decomposition chapter of *DIGITAL High Performance Fortran 90 HPF and PSE Manual*.
- `mandelbrot.f90` visualizes the Mandelbrot Set. It is referred to by the HPF Tutorial. This program uses the PURE attribute and non-Fortran subprograms within an HPF program. Mandelbrot also requires these files: `simpleX.h`, `simpleX.c`, and `dope.h`. Read the `README.mandelbrot` file to see how to compile and execute Mandelbrot.
- `pi_example.f90` calculates pi using four different Fortran 90 methods. This program contains a timing module which may be pulled out and used separately.
- `shallow.f90` is a optimized HPF version of the Shallow Water benchmark.
- `twin.f90` demonstrates Compaq Fortran's new non-wsf main program capability.
- `hpf_gexample.f` is a Fortran program with explicit calls to PVM. It demonstrates some group and reduction operations in PVM. You must have PVM installed to run this program.
- `hpf.tcl` is a TK-based HPF data distribution learning aid. It illustrates the data distribution patterns represented by various data distributions, such as (BLOCK, *), (*, CYCLIC), (BLOCK, CYCLIC), etc.
- `fft.f90` performs a fast Fourier transform, achieving parallelism by means of EXTRINSIC(HPF_LOCAL) routines.

3.7 New Features and Corrections in Version 5.1

Version 5.1 is a major release that includes corrections to problems discovered since Version 5.0 was released.

The following topics are discussed:

- Version 5.1 New Features
- Version 5.1 Corrections

3.7.1 Version 5.1 New Features

The following new Compaq Fortran (DIGITAL Fortran 90) features are now supported:

- DIGITAL Fortran 90 on UNIX contains full support for OpenMP. Here are some details from the OpenMP web site (http://www.openmp.org).

  The OpenMP application program interface (API) supports multi-platform shared-memory programming on UNIX platforms and Microsoft Windows NT architectures. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives shared-memory programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer.

  For more information on the OpenMP Fortran API, see the revised user manual.

  The following directives in OpenMP are supported by DIGITAL Fortran 90 Version 5.1:

  - !$OMP PARALLEL and !$OMP END PARALLEL
  - !$OMP DO and !$OMP END DO
  - !$OMP SECTIONS and !$OMP END SECTIONS and !$OMP SECTION
  - !$OMP SINGLE and !$OMP END SINGLE
  - !$OMP PARALLEL DO and !$OMP END PARALLEL DO
  - !$OMP PARALLEL SECTIONS and !$OMP END PARALLEL SECTIONS
  - !$OMP MASTER and !$OMP END MASTER
  - !$OMP CRITICAL [(name)] and !$OMP END CRITICAL [(name)]
  - !$OMP BARRIER
  - !$OMP ATOMIC
  - !$OMP FLUSH
  - !$OMP ORDERED and !$OMP END ORDERED
  - !$OMP THREADPRIVATE

  The following clauses on directives in OpenMP are supported by DIGITAL Fortran 90 Version 5.1:

  - IF (exp)
  - PRIVATE(list)
  - SHARED(list)
  - DEFAULT (PRIVATE | SHARED | NONE)

- FIRSTPRIVATE(list)
- LASTPRIVATE(list)
- REDUCTION({operator | intrinsic} : list)
- COPYIN(list)
- SCHEDULE(type[,chunksize])
- ORDERED

- The following new features are now supported:
  - As of DIGITAL UNIX v4.0, constants in Fortran code are placed in read-only memory. An attempt to modify a constant (as in the example below) has always been an error but will now cause the program to abort:

```
CALL F (1)
...
SUBROUTINE F (I)
I = 2
```

  - A change in the math library (`libm`) starting with DIGITAL UNIX v4.0B is that `MOD(3.0,0.0)` now returns "floating invalid"; it used to return "0".
  - Several new intrinsics are now available in DIGITAL Fortran 90. For more details, see the online Fortran 90 help file, located in:

    `/usr/lib/cmplrs/fort90/decfortran90.hlp`

    * ASM - execute in-line assembler code
    * LEADZ and TRAILZ - count leading and trailing 0 bits in an integer
    * POPCNT - count 1 bits in an integer
    * POPPAR - parity of the bits in an integer
    * MULT_HIGH - multiply two 64-bit unsigned integers
  - The X-Open Standard followed by DIGITAL UNIX made a change to its "pow" function that affects Fortran's "**" operation: (0.0)**Y for all negative values of Y {single or DOUBLE PRECISION} will now return "-Infinity" (using `-fpe3`); it used to return "+Infinity".

- The following new `f90` command options are now supported:
  - `-align rec`*N*`byte`

    Requests that fields of records and components of derived types be aligned on the smaller of:

    * The size byte boundary (*N*) specified (*N* is 1, 2, 4, or 8)
    * The boundary that will naturally align them

    Specifying `-align rec`*N*`byte` does not affect whether common blocks are naturally aligned or packed.
  - `-altparam`

    Specifies if the alternate form of parameter constant declarations (without parenthesis) is recognized. The default is `-altparam`.
  - `-assume minus0`

Tells compiler to use Fortran 95 standard semantics for the treatment of the IEEE® floating value -0.0 {of all KINDs}. There are two places where Fortran 95 defines behavior on -0.0:

* SIGN (data, -0.0) is "data" with a negative sign, whereas the Fortran 90 standard says SIGN (data, -0.0) is the same as SIGN (data, +0.0) which is "data" with a positive sign
* Fortran 95 says that -0.0 prints as "-0.0", whereas Fortran 90 says it prints as "0.0"

`-assume nominus0` is the default {this is a change from DIGITAL Fortran 90 V5.0} and means that SIGN (data, -0.0) is the same as SIGN (data, +0.0) {the Fortran 90 and FORTRAN 77 standard semantics}

The `f95` command driver adds "`-assume minus0`" to the compiler command line (before any other options) so the `f95` command will get the Fortran 95 standard semantics.

– `-check omp_bindings`

Provides run-time checking to enforce the OpenMP binding rules:

* It is an error to enter a DO, SINGLE, or SECTIONS if you are already in a work-sharing construct, a CRITICAL SECTION, or a MASTER.
* It is an error to attempt to execute a BARRIER if you are already in a work-sharing construct, a CRITICAL SECTION, or a MASTER.
* It is an error to attempt to execute a MASTER directive if you are already in a work-sharing construct.
* It is an error to execute an ORDERED directive if you are already in a CRITICAL SECTION.
* It is an error to execute an ORDERED directive unless you are already in an ORDERED DO.

The default is `-check noomp_bindings`:

* `-omp` implies `-check omp_bindings`
* `-fast -omp` implies `-check noomp_bindings`, regardless of the placement of `-fast`
* If the user wants the checking done on `-mp`, specify `-check omp_bindings` explicitly

At run-time, the errors in example program `t.f` trigger an ASSERTION error and the program aborts:

```
Example program t.f:

 real b(100)
        x = 0

!$omp paralleldo
 do i= 1, 100
           b(i) = i
!$omp single
           x = x + 1
!$omp end single
 end do

 print *, b, x
 end
```

```
> f90 -omp t.f
> a.out
forrtl: severe (145): assertion error
```

- `-module` *directory*

  Requests that the compiler create module files in the specified *directory* instead of the current directory.

- `-omp`

  Enables recognition of OpenMP directives.

- `-std95`

  Enables Fortran 95 standards checking (`-std90` or `-std` enable Fortran 90 standards checking).

- `-warn truncated_source`

  Requests that the compiler issues a warning diagnostic message when it reads a source line with a statement field that exceeds the maximum column width in fixed-format source files. The maximum column width for fixed-format files is column 72 or 132, depending whether the `-extend_source` option was specified.

  This option has no effect on truncation; lines that exceed the maximum column width are always truncated.

  This option does not apply to free-format source files. The default is `-warn notruncated_source`.

- Additional support has been provided for directed parallel processing using the `-omp` and `-mp` options.

  For more information on the parallel directives, see the *Compaq Fortran User Manual for Tru64 and Linux Alpha Systems*.

  To allow task-local thread storage, you must be using Version 4.0D (code name PTmin) of the DIGITAL UNIX operating system.

  The following problem in the use of `-omp` and `-mp` parallel directives should be noted:

  In the following example `test.f`, the user should add "firstprivate(k,m)" to initialize the private variables k and m for use in the loop control expressions.

```
Example test.f:
      dimension x(10)

      k = 1
      m = 10

!$omp parallel
!$omp do private(k,m)
      do i = k,m
         x(i) = i
      enddo
!$omp end parallel

      print *, x
      end
```

```
> f90 -omp test.f
f90: Warning: test.f, line 8: Variable K is used before its value
                              has been defined
      do i = k,m
------^
f90: Warning: test.f, line 8: Variable M is used before its value
                              has been defined
      do i = k,m
------^
```

- The following Fortran 95 features are have been implemented in Version 5.1:
  - Zero-length formats

    On output, when using I, B, O, Z, and F edit descriptors, the specified value of the field width can be zero. In such cases, the compiler selects the smallest possible positive actual field width that does not result in the field being filled with asterisks (*).
- The command-line options `-assume minus0` and `-std95` (described previously in this section).

3.7.2 Version 5.1 Corrections

Since Version 5.0, the following corrections have been made:

- Using ASSOCIATED with f90 pointer now gives correct answer.
- Using vector subscripts in MATMUL now gives correct answer.
- Passing %REF argument to a routine with explicit INTERFACE no longer gets an internal error.
- CSHIFT of an array pointer contained within a derived type no longer gets an internal error.
- Compiling files that contain very long routine names with `-V` no longer gets an internal error.
- Using assignments in a defined generic assignment subroutine when the subroutine is not RECURSIVE now gets an error.
- Parameter constant is allowed as argument of a LOC intrinsic.
- Using UNION within derived type now gets correct result.
- Having EQUIVALENCEd character array elements within a MODULE no longer gets internal error.
- Duplicate SAVE statement no longer gets an error.
- Parameter constant can be used as case-value in a SELECT CASE statement.
- ALIAS attribute can now be specified in a cDEC$ ATTRIBUTE directive for a variable that has not been declared EXTERNAL (EXTERNAL is assumed).
- Interface with optional function argument is now resolved properly.
- Using record fields in multiply and add operations now produces correct result.
- Using the following operators: ==, /=, <, >, <=, and >= no longer get non-standard conforming warnings.
- Extra trailing blanks are now allowed and ignored when used in specifier of OPEN statement, e.g., FORM='formatted '.

- Passing an array argument to a statement function now gets an error.
- INTEGER*2 array now gets correct result when compiled with `-integer_size 16`.
- Fix a bug related to module importing with modules that contain PRIVATE statement.
- Parameter constant defined in a MODULE is now imported when its use is only in a variable format expression.
- C attribute can be specified in a cDEC$ ATTRIBUTE directive for module variables.
- Parameter constants are allowed in a structure constructor.
- A derived type component having the same name as a common block no longer gets an internal error.
- IVDEP directive can be specified between PDO and DO statements.
- A non-standard warning is issued if the first argument of a GENERIC assignment procedure is not INTENT(OUT) or INTENT(INOUT).
- $PACK directive and the `-align recNbyte` option now affect alignment of data items in a SEQUENCE derived-type.
- Using a structure constructor to initialize a multi-dimensional array component of a derived-type no longer causes an internal error.
- Fix $omp parallel copyin (/common_block/) directive.
- The `-fpconstant` option now works correctly for assignment to double complex variables.
- Having a D line as the first non-comment line after a conditional $ENDIF directive no longer gets an error.
- No longer flag ES format as non-standard.
- Remove an internal limit on the number of entries of a NAMELIST
- Using substring of a character array as argument of ICHAR intrinsic no longer gets internal error
- Pointer assignment of an array of character substrings now gets correct result. For example: `p=>a(:)(2:4) )`
- Using array transformation intrinsics such as PACK, SPREAD, and RESHAPE with array of derived-type as argument in a PRINT statement now gets correct results
- Allow an array with a name of TYPE
- Specifying $NOFREEFORM in a .f90 file now sets the line size to 72 columns
- Remove a limit of 256 number of arguments for subroutines and functions
- An incorrect statement: "IF (ABS(I).GT 1) I=0" now gets an error message
- An incorrect statement: "CHARACTER(LEN=1), PARAMETER :: CPSCLR = ” ”" now gets an error message
- Using record fields in multiply and subtract operations now produces correct results

- Having a PRIVATE, EQUIVALENCE variable in a module no longer causes compile time segmentation violation
- Specifying ONLY on one variable in a COMMON block now only declares the one variable (not the entire variables) in the COMMON block
- Allow user-defined operator to be used as the format character expression in a PRINT or READ statement
- Using modules and the AUTOMATIC statement in the same routine no longer gets internal error
- Module variables with EXTERN attributes now work properly
- Increase an internal limit so that large programs no longer get the "text handle table overflow" message
- Using record fields in exponentiation and subtract operations now produce correct result
- Flag incorrect usage of an entry dummy argument in an executable statement before its declaration in the entry statement
- Disallow optional return dummy argument following other OPTIONAL dummy arguments
- An invalid WRITE statement no longer gets an internal error
- Allow passing NULL intrinsic function as argument to other routines
- Allow AUTOMATIC variables to be used in an EQUIVALENCE statement
- Using a structure constructor with scalar value to assign an array element now produces correct result
- Using an array constructor with integer value to initialize a real or complex array now produces correct result
- Flag common block name and routine name conflict
- Fix elemental character function with varying length
- Fix problem where `-assume dummy_aliases` wasn't being taken into account in checks for overlap in array assignment.
- If !DEC$ ATTRIBUTES C is specified in an INTERFACE block, and an argument is declared as having an array type, always pass that argument by reference even if the actual argument is a single array element.
- Allow a concatenation expression as the first argument to TRANSFER.
- Implement `-std90` and `-std95` options.
- A STRUCTURE with no fields no longer causes a compiler failure.
- No longer issue standards warning for certain cases with a relational operator followed by a unary minus.
- Do not give spurious "more variables than values" warning for certain cases of data initialization of multi-dimensional arrays.
- User-defined generic interface for IDATE no longer causes internal compiler error.

- LOC(funcname) as an actual argument when used inside function "funcname" now properly returns the address of the return value variable and not the entry point.
- The compiler no longer gives spurious errors (including internal compiler failures) in certain cases where a module file cannot be opened.
- Certain incorrect programs which include a reference to fields of an undeclared STRUCTURE no longer cause an internal compiler error.
- Give error when ALLOCATED is used on a non-ALLOCATABLE object.
- Allow a recursive function name to be passed as an actual argument inside the function.
- If an INCLUDE file includes a directive to change the source form, revert to the original setting after returning to the including source file.

  The following limitations were fixed in Version 2.0 or previous releases:

  - The `f90` command option `-wsf` and its related options are now supported.
  - Allocatable arrays that are allocated but not deallocated before routine exit are now deallocated upon exit from routine.
  - The `f90` command options `-cord` and `-feedback` described in the documentation have now been implemented.

3.7.3 HPF Version 5.1 New Features

3.7.3.1 SHADOW Directive Now Supported

The new SHADOW directive, as defined in Version 2.0 of the *High Performance Fortran Language Specification*, is now supported. SHADOW is now a separate HPF directive, rather than a keyword inside the DISTRIBUTE directive.

3.7.3.2 Pointers Now Handled in Parallel

Mapped variables with the POINTER attribute are now handled in parallel. This capability is an approved extension of the *High Performance Fortran Language Specification*.

3.7.3.3 SHADOW Directive Required for Nearest-Neighbor POINTER or TARGET Arrays

The compiler will *not* generate shadow edges automatically for arrays with the POINTER or TARGET attributes. In order to be eligible for the compiler's nearest-neighbor optimization, POINTER or TARGET arrays must explicitely be given shadow edges using the SHADOW directive. If pointer assignment is done, both the POINTER and the TARGET must have the same mapping, including shadow edges.

For More Information:

- On the conditions that must be satisfied for a statement to be eligible for the nearest-neighbor optimization, see Section 3.6.1.4 of these Release Notes.

3.7.3.4 Descriptive Mapping Directives are Now Obsolescent

In Version 1 of the HPF Language Specification, a special form of the DISTRIBUTE and ALIGN directives was used in interfaces and procedures when mapped arrays were passed to a procedure. Known as *descriptive* mapping, it was specified by an asterisk (*) appearing before the left parenthesis "(" in a DISTRIBUTE directive, or after the WITH in an ALIGN directive. For example,

```
!HPF$ DISTRIBUTE R*(BLOCK, BLOCK)
!HPF$ ALIGN S WITH *R
```

Beginning with version 2.0 of the *High Performance Fortran Language Specification* (DIGITAL Fortran 90 Version 5.0), the meaning of descriptive syntax has changed. Descriptive mapping is now a weak assertion that the programmer believes that no data communication is required at the procedure interface. If this assertion is wrong, the data communication will in fact occur.

Although there is now no semantic difference between the descriptive form and the ordinary prescriptive form, there is still some benefit in using the descriptive form. Compaq Fortran generates informational messages when a descriptive directive is specified if the compiler is unable to confirm that there will in fact be no communication. These messages can uncover subtle programming mistakes that cause performance degradation.

Existing programs with descriptive mapping directives will continue to compile and run with no modification.

In the future, DIGITAL may provide a command-line option that specifies that descriptive directives be treated as strong assertions that data communication will not be necessary at the procedure interface. This would allow the compiler to omit checking whether the mappings of the actual and dummy agree, leading to performance improvement in some cases.

3.7.3.5 New support for HPF Local Library Routines GLOBAL_LBOUND and GLOBAL_UBOUND

The following HPF Local Library routines are now supported:

- GLOBAL_LBOUND
- GLOBAL_UBOUND

3.7.3.6 REDUCTION Clause in INDEPENDENT Directives

The REDUCTION clause in INDEPENDENT directives is now supported.

3.7.3.7 HPF_SERIAL Restriction Lifted for Procedures Called from INDEPENDENT DO Loops

Previous versions required procedures called from inside INDEPENDENT DO loops to HPF_SERIAL in order to obtain parallel execution. This restriction is now lifted.

For More Information:

- On the requirements for parallel execution of INDEPENDENT DO loops containing procedure calls, see Section 3.6.5.6 of these Release Notes.

3.7.4 HPF Version 5.1 Corrections

This section lists problems in previous versions that have been fixed in this version.

- Some bugs in implementing whole structure references in IO and assignment were fixed.
- Aligning components of derived types is now supported.
- The restriction that statements with scalar subscripts are not eligible for the nearest-neighbor optimization is now removed. Statements with scalar subscripts may now be eligible for the nearest-neighbor optimization if that array dimension is (effectively) mapped serially.
- Nearest-neighbor assignments with derived types are now eligible for the nearest-neighbor optimization.

3.8 New Features and Corrections in Version 5.0

Version 5.0 is a major release that also includes corrections to problems discovered since Version 4.1 was released.

The following topics are discussed:

- Version 5.0 New Features
- Version 5.0 Corrections

3.8.1 Version 5.0 New Features

The following new Compaq Fortran (DIGITAL Fortran 90) features are now supported:

- The `-mp` compiler option enables parallel processing using directed decomposition. Parallel processing is directed by inserting !$PAR directives in your source code. This type of parallel processing is for shared memory multiprocessor systems.

  To enable parallel processing across clusters of servers or workstations with !HPF$ directives, continue to use the `-wsf` compiler option.

  On a shared memory system, you can use both the `-mp` and the `-wsf` options for the same program. This combination provides improved performance for certain programs running mostly on shared memory systems.

  The new parallel directives:

  - Use the !$PAR prefix
  - Are recognized only if compiled with the `-mp` option
  - Some of the parallel !$PAR directives include:

    PARALLEL and END PARALLEL
    PDO
    PARALLEL DO
    PSECTIONS
    CRITICAL SECTION
    TASK COMMON

  - Environment variables control the run-time behavior. For example, MP_ THREAD_COUNT specifies how many threads to create.

To allow task-local thread storage, you must be using Version 4.0D (code name PTmin) of the DIGITAL UNIX operating system.

For more information on these directives, see the *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*.

- The `-warning_severity` *keyword* compiler option allows you to:
  - Specify `-warning_severity errors` to make all compiler warning messages error-level instead of warning-level messages.
  - Specify `-warning_severity stderrors` to make standards checking compiler warning messages (`-std` option) error-level instead of warning-level messages.
- The `-warn nogranularity` compiler option allows you to suppress the NONGRNACC warning message: `Unable to generate code for requested granularity`.
- An internal procedure can now be used as an actual argument.
- Support for certain new language extensions for compatibility with Compaq Visual Fortran (and Microsoft® Fortran PowerStation). These features include the following:
  - # Constants–constants using other than base 10
  - C Strings–NULL terminated strings
  - Conditional Compilation And Metacommand Expressions ($define, $undefine, $if, $elseif, $else, $endif)
  - $FREEFORM, $NOFREEFORM, $FIXEDFORM–source file format
  - $INTEGER, $REAL–selects size
  - $FIXEDFORMLINESIZE–line length for fixed form source
  - $STRICT, $NOSTRICT–F90 conformance
  - $PACK–structure packing
  - $ATTRIBUTES ALIAS–external name for a subprogram or common block
  - $ATTRIBUTES C, STDCALL–calling and naming conventions
  - $ATTRIBUTES VALUE, REFERENCE–calling conventions
  - \ Descriptor–prevents writing an end-of-record mark
  - Ew.dDe and Gw.dDe Edit Descriptors–similar to Ew.dEe and Gw.dEe
  - 7200 Character Statement Length
  - Free form infinite line length
  - $DECLARE and $NODECLARE == IMPLICIT NONE
  - $ATTRIBUTES EXTERN–variable allocated in another source file
  - $ATTRIBUTES VARYING–variable number of arguments
  - $ATTRIBUTES ALLOCATABLE–allocatable array
  - Mixing Subroutines/Functions in Generic Interfaces
  - $MESSAGE–output message during compilation
  - $LINE == C's #line

- INT1 converts to one byte integer by truncating
- INT2 converts to two byte integer by truncating
- INT4 converts to four byte integer by truncating
- COTAN returns cotangent
- DCOTAN returns double precision cotangent
- IMAG returns the imaginary part of complex number
- IBCHNG reverses value of bit
- ISHA shifts arithmetically left or right
- ISHC performs a circular shift
- ISHL shifts logically left or right

The following new High Performance Fortran (HPF) features and corrections have been added for DIGITAL Fortran Version 5.0:

- The new SHADOW directive, as defined in Version 2.0 of the HPF specification, is now supported. SHADOW is now a separate HPF directive, rather than a keyword inside the DISTRIBUTE directive.
- Mapped variables with the POINTER attribute are now handled in parallel. This capability is an approved extension of the HPF specification.
- Beginning with version 2.0 of the HPF specification (DIGITAL Fortran Version 5.0), the meaning of descriptive syntax has changed. Descriptive mapping is now a weak assertion that the programmer believes that no data communication is required at the procedure interface. If this assertion is wrong, the data communication will in fact occur.

  Existing programs with descriptive mapping directives will continue to compile and run with no modification and no performance penalty.
- The following HPF Local Library routines are now supported:
  - GLOBAL_LBOUND
  - GLOBAL_UBOUND
- The REDUCTION clause in INDEPENDENT directives is now supported.
- Previous versions required procedures called from inside INDEPENDENT DO loops to HPF_SERIAL in order to obtain parallel execution. This restriction is now lifted.
- Some bugs in implementing whole structure references in IO and assignment were fixed.
- Aligning components of derived types is now supported.
- The restriction that statements with scalar subscripts are not eligible for the nearest-neighbor optimization is now removed. Statements with scalar subscripts may now be eligible for the nearest-neighbor optimization if that array dimension is (effectively) mapped serially.
- Nearest-neighbor assignments with derived types are now eligible for the nearest-neighbor optimization.

3.8.2 Version 5.0 Corrections

Since Version 4.1, the following corrections have been made:

- Fix SIGN intrinsic to handle –0.
- Fix LOC intrinsic and %LOC of a derived type field.
- Fixed debug information for dynamic character variable (such as `character*(i) c`).
- Add debugging support for integer (Cray) pointers.
- Fix storing to the return value of a function returning character in a containing internal routine.
- Fix Nullification of a character*n pointer argument.
- Fix using passed length argument in a containing internal routine.
- Fix compiler abort when a source line is longer than 1024 characters in freeform source file.
- Fix using IOLENGTH in a INQUIRE statement.
- Fix FORALL problem of the form "`X(X(I)) =`."
- Fix contained functions returning an implicitly initialized derived-type.
- Better diagnostics for invalid programs.
- Fix compiler abort when using Nullification of a pointer in a MODULE.
- Fix a certain type of USE of a MODULE with rename list.
- Fix using `-extend_source:80` and `-pad_source`.
- Fix compiler abort when using do-loop style implicitly initialized derived-types in a MODULE.
- Sign-extending INTEGER*2 parameter constants.
- Flag invalid nested internal procedures.
- Fix compiler abort of USE of a MODULE with namelist variables in rename list.
- Issue a warning message for a intrinsic with wrong argument type and treat it as an external.
- Issue a warning message for having a SAVE common block data object.
- Fix compiler abort of USE of a MODULE with namelists.
- Fix using SIZEOF(common_block_array) in a PARAMETER statement.
- Fix using READONLY keyword as first keyword in an OPEN statement.
- Allow record name to be the same as a structure name.
- Fix parameter character constant with embedded NULL character.
- Fix compiler abort when same name used as a structure and derived type.
- Allow BLOCKSIZE keyword in an INQUIRE statement.
- Allow a record in a SAVE statement.
- Allow a module to have the name "procedures".

- Do not flag IABS intrinsic function as nonstandard.
- Do not flag DOUBLE COMPLEX as nonstandard.
- Treat POPCNT, POPPAR, LEADZ as external functions.
- Put out an error message for `ptr => pack(...)`.
- Treat C$DOACROSS as a comment.
- Issue an error message for invalid derived type parameter constant.
- Fix compiler abort when passing an array constructor as an actual argument.
- Fix using derived-type components that are same as intrinsic names.
- Fix an incorrect warning about "explicit-shaped array is being passed to a routine that expects a pointer or assumed-shape array".
- Fix a problem with `-warn:errors` and `-stand:f90` options. Nonstandard messages should be error messages.
- Fix incorrect results when compiled a program with `-assume:dummy_aliasing`.
- Do not flag BOZ constant as nonstandard.
- Do not flag Z format as nonstandard.
- Allow 511 continuation lines.
- Put out a standard warning for using character constant in DATA statement.
- Fix using TRANSFER in initialization.
- Fix a problem with user defined assignment statement.
- Issue an error message when passing or receiving an optional argument by value.
- Fix an invalid message about return value of a function is not defined when the function returns an initialized derived type.
- Fix a compiler abort with "text handle table overflow" message.
- Fix a compiler abort using a SAVE statement.
- Fix a problem when an existing operator is overloaded.
- Fix argument checking of intrinsic subroutines.
- Fix generic interface of elemental functions.
- Issue an argument mismatch warning message for using an integer with a statement function that takes real argument.
- Fix compiler directives processing.
- Fix a compiler abort using an invalid PARAMETER array.
- Issue an error message for SAVE of an ENTRY result variable.
- Fix using UNION within derive type.
- Fix a compiler internal error when using C and REFERENCE attributes on a function name.
- Fix a compiler internal error when using ASSOCIATED of a function returning a pointer.
- Add support for passing complex by value.

- Fix pointer assignment with a character substring.
- Allow using ICHAR in an array constructor within the initialization part of an array declaration.
- Fix a problem with using UNION declaration within the derived type.
- Allow exporting of a module procedure which has a name that is the same as a generic name.
- Fix a problem with using user defined assignment operation.
- Allow specifying NaNs in the PARAMETER statement.
- Allow D source line to continue a non-D source line.
- Fix a problem in array initialization processing.
- Cray pointees that were being allocated statically are now correctly given no storage class.
- Using assume shape array within a contained routine no longer produces an internal compiler error.
- An error message is now given for invalid keyword values given for an I/O statement's keyword.
- Declarations of the type "character, allocatable :: field*7(:)", in which the array shape specifier comes after the length specification in a deferred-shape character array no longer produces an internal compiler error.
- When assigning a derived type variable with a structure constructor, if a character scalar is supplied to an character array component, every elements of the array is assigned with the character scalar value.
- The MVBITS intrinsic now gives correct result if its 4th argument is a non-lowerbound subscripted array element.
- Reference of a character function, where the length of its return value is dependent on one or more of its arguments, no longer produces an internal compiler error.
- Pointer assignment should now work properly when the target is a component of an allocatable array with a lower bound different of 1.
- Long NAMELISTs no longer causes a compiler internal error.
- The compiler now prints out better error messages for the PDONE directive.
- When initializing a derived type variable with a structure constructor, if a scalar is supplied to an array component, every elements of the array is initialized with the scalar value.
- Allow %fill in STRUCTURE declarations using F90 syntax, such as: `integer :: %fill`.
- Using unary "-" operator with record fields should now give correct results.
- Use of NEAREST with two different KIND REAL arguments no longer gets a nonstandard warning.
- Allow SIZEOF of assumed size record field.
- Module importing has been improved. If a module is USEd in both the host and its internal procedure, the module is now only imported once by the compiler.

- A module that contains "PRIVATE" followed by "PUBLIC variable" no longer gets incorrect error message.
- Optional comma in DO with label, such as: `label: DO, JJ = 1, N, 1` no longer gets incorrect syntax error.
- Allow a dummy argument to have the same name as a structure field.
- A module that contains `USE module, ONLY : var` no longer gets internal compiler error.
- A component of a derived-type with a name that starts with FILL no longer gets a syntax error.
- A PDONE directive in a statically-scheduled PDO loop no longer gets an error message.
- Allow a variable with '_' in its name to be used in a variable format expression.
- Fix for array references in CRITICAL SECTION directive.
- Set NOWAIT for paralleldo directive (the region waits, so do NOT make the loop wait as well).
- Allow variables in COMMON/EQUIVALENCE on local, lastlocal, and reduction lists. Create and use new local variables in parallel do scopes for these variables.
- Allow c$copyin of EQUIVALENCE/COMMON variables.
- Fix `-mp` (ordered) bug.
- No longer import any definitions if a module is used with the "USE module_ name, ONLY:" statement.
- Fix a compile time stack overflow problem.
- Fix a "$IF DEFINED( )" problem when a routine is defined between the conditional compilation.
- Put out error message for an invalid use of "TYPE (type_name)" statement.
- Allow RECORD in a NAMELIST.
- Fix using "# line_number" in DATA statement.
- The `-pad_source` option now properly pads Hollerith literals that are continued across source records.
- Add standards warning for using two consecutive operators in an expression.
- Allow POINTER attribute for character entities whose length is specified by a variable expression or an * (assumed length character).
- Do not flag as nonstandard when all of the objects in the EQUIVALENCE set are of type default integer, default real, double precision, default complex, default logical, or numeric sequence type.
- Add standards warning for assignment to the host associated variable in a PURE function.
- Add standards warning for using a dummy argument in a specification-expr as the argument of one of the intrinsic functions BIT_SIZE, KIND, LEN, or the numeric inquiry functions.

- Compiling BLOCK DATA with `-recursive` no longer causes a compiler internal error.
- A special usage of equivalenced variables in parallel no longer causes a compiler internal error.
- DO loop variables are now set to be PRIVATE by default in a parallel region.
- The scope of C$CHUNK and C$MP_SCHEDTYPE directives is restricted to one program unit.
- Fix a bug in a special usage of passing internal procedure as argument.

3.9 Additional Information

This section contains information that supplements the HP Fortran documentation.

3.9.1 HP Fortran Home Page

The HP Fortran Web site is located at:

`http://www.hp.com/software/fortran`

3.9.2 Support for the Fortran 95 Standard Features

This section briefly describes the Fortran 95 language features that have been added to Compaq Fortran:

- The FORALL statement and construct

  In Fortran 95/90, you could build array values element-by-element by using array constructors and the RESHAPE and SPREAD intrinsics. The Fortran 95 FORALL statement and construct offer an alternative method.

  FORALL allows array elements, array sections, character substrings, or pointer targets to be explicitly specified as a function of the element subscripts. A FORALL construct allows several array assignments to share the same element subscript control.

  FORALL is a generalization of WHERE. They both allow masked array assignment, but FORALL uses element subscripts, while WHERE uses the whole array.

  Compaq Fortran previously provided the FORALL statement and construct as language extensions.

- PURE procedures

  Pure user-defined procedures do not have side effects, such as changing the value of a variable in a common block. To specify a pure procedure, use the PURE prefix in the function or subroutine statement. Pure functions are allowed in specification statements.

  Compaq Fortran previously allowed pure procedures as a language extension.

- ELEMENTAL procedures

  An elemental user-defined procedure is a restricted form of pure procedure. An elemental procedure can be passed an array, which is acted upon one element at a time. To specify an elemental procedure, use the ELEMENTAL prefix in the function or subroutine statement.

- Pointer initialization

In Fortran 95/90, there was no way to define the initial value of a pointer or to assign a null value to the pointer by using a pointer assignment operation. A Fortran 95/90 pointer had to be explicitly allocated, nullified, or associated with a target during execution before association status could be determined.

Fortran 95 provides the NULL intrinsic function that can be used to nullify a pointer.

- Derived-type structure default initialization

  Fortran 95 lets you specify, in derived-type definitions, default initial values for derived-type components.

- Automatic deallocation of allocatable arrays

  Arrays declared using the ALLOCATABLE attribute can now be automatically deallocated in cases where Fortran 95/90 would have assigned them undefined allocation status.

  Compaq Fortran previously provided this feature as a language extension.

- CPU_TIME intrinsic subroutine

  This new intrinsic subroutine returns a processor-dependent approximation of processor time.

- Enhanced CEILING and FLOOR intrinsic functions

  KIND can now be specified for these intrinsic functions.

- Enhanced MAXLOC and MINLOC intrinsic functions

  DIM can now be specified for the MAXLOC and MINLOC intrinsic functions. DIM was allowed previously as a Compaq Fortran language extension.

- Enhanced SIGN intrinsic function

  The SIGN function can now distinguish between positive and negative zero (if the processor is capable of doing so).

- Enhanced WHERE construct

  The WHERE construct has been improved to allow nested WHERE constructs and a masked ELSEWHERE statement. WHERE constructs can now be named.

- Comments allowed in namelist input

  Fortran 95 allows comments (beginning with !) in namelist input data. Compaq Fortran previously allowed this as a language extension.

- Generic identifier to END INTERFACE statement

  The END INTERFACE statement of an interface block defining a generic routine now allows a generic identifier.

- Zero-length formats

  On output, when using I, B, O, Z and F edit descriptors, the specified value of the field width can be zero (0). In such cases, the compiler selects the smallest possible positive actual field width that does not result in the field being filled with asterisks.

- New obsolescent features

  Fortran 95 deletes several language features that were obsolescent in Fortran 90, and identifies new obsolescent features:

  – REAL and DOUBLE PRECISION DO variables

- Branching to an ENDIF from outside its IF
- PAUSE statement
- ASSIGN statement, assigned GOTO, and assigned FORMATs
- H edit descriptor

Compaq Fortran flags these deleted and obsolescent features, but *fully* supports them.

3.9.3 Preliminary Information on Support for Big Objects

Big objects are data items whose size cannot be represented by a signed 32 bit integer. Compaq Fortran supports larger objects than Compaq Fortran 77.

Big objects are good for massive machines and clusters used for numerical analysis, such as weather forecasting and high energy physics problems. Both special knowledge and very large hardware configurations are needed to use this feature.

Your system and its operating system must be configured to:

- Allow a very large stack space.
- Allow a very large data space.
- Allow large values for parameters, such as vm-maxvas.
- Unless huge amounts of physical memory are present, enable lazy swapping.
- Check the size of page/swap files and create larger files if needed.

For more information, see the Compaq Tru64 UNIX system management documentation. For Compaq Tru64 UNIX Version 4.0, you can use the following check list:

1. Either have a large swap space or use deferred swap allocation. This involves either:
   - Have more swap space than the address space used by the largest program you want to run. The following command shows swap allocation:

     ```
     $ /usr/sbin/swapon -s
     ```

   - Use the deferred mode of swap allocation. The following command displays the reference (man) page for swapon, which describes how to change the swap allocation

     ```
     $ man swapon
     ```

2. Reconfigure the UNIX kernel (for Version 4.0 or later) to change the following parameters as desired. For example, on one system, all values were set to 16 GB:

Parameter	Explanation
max-per-proc-address-space	Largest address space
max-per-proc-data-size	Largest data size
max-per-proc-stack-size	Largest stack size
vm-maxvas	Largest virtual-memory

Also set the following per-process values:

Parameter	Explanation
per-proc-address-space	Default address space
per-proc-data-size	Default data size
per-proc-stack-size	Default stack size

The per-process limits can be checked and increased with the `limit` or `ulimit` commands.

You can create big objects as static data, automatic data (stack), or dynamically allocated data (ALLOCATE statement or other means).

The address space limitations depends on the Alpha processor generation in use:

- Address space for ev4 Alpha generation processors is 2**42
- Address space for ev5 Alpha generation processors is 2**46

Although the compiler produces code that computes 63-bit signed addresses, objects and addresses larger than the hardware limitations will not work.

Limitations of using big objects include:

- Initializing big objects by using a DATA statement or a TYPE declaration is not supported.
- Big objects cannot be passed by value as program arguments.
- Debug support for big objects is limited.
- I/O of entire big objects is not supported, but I/O of parts of an array should work.

The following small example program allocates a big character object:

```
character xx(2_8**31+100_8)
integer*8 i
i = 10
xx(i) = 'A'
i = 2_8**31 + 100_8
xx(i) = 'B'
print *,xx(10_8)
print *,xx(i)
end
```

3.9.4 New Random Number Algorithm

A new random_number intrinsic (Version 4.0 or later) uses a different algorithm than the one previously used.

The test program below shows the use of the random_seed and random_number intrinsics.

```
program testrand
   intrinsic random_seed, random_number
   integer size, seed(2), gseed(2), hiseed(2), zseed(2)
   real harvest(10)
   data seed /123456789, 987654321/
   data hiseed /-1, -1/
   data zseed /0, 0/
   call random_seed(SIZE=size)
   print *,"size ",size
   call random_seed(PUT=hiseed(1:size))
   call random_seed(GET=gseed(1:size))
   print *,"hiseed gseed", hiseed, gseed
   call random_seed(PUT=zseed(1:size))
   call random_seed(GET=gseed(1:size))
   print *,"zseed gseed ", zseed, gseed
   call random_seed(PUT=seed(1:size))
   call random_seed(GET=gseed(1:size))
   call random_number(HARVEST=harvest)
   print *, "seed gseed ", seed, gseed
   print *, "harvest"
   print *, harvest
   call random_seed(GET=gseed(1:size))
   print *,"gseed after harvest ", gseed
end program testrand
```

When executed, the program produces the following output:

```
%  testrand
 size           2
 hiseed gseed          -1          -1         171         499
 zseed gseed            0           0  2147483562  2147483398
 seed gseed    123456789   987654321   123456789   987654321
 harvest
  0.6099895      0.9807594      0.2936640      0.9100146      0.8464803
  0.4358687      2.5444610E-02  0.5457680      0.6483381      0.3045360
 gseed after harvest    375533067  1869030476
```

3.9.5 Compaq Fortran 77 Pointers

Compaq Fortran 77 pointers are CRAY® style pointers, an extension to the Fortran 90 standard. The POINTER statement establishes pairs of variables and pointers, as described in the *Compaq Fortran Language Reference Manual*.

3.9.6 Extended Precision REAL (KIND=16) Floating-Point Data

The X_float data type is a little endian IEEE-based format that provides extended precision. It supports the REAL*16 Compaq Fortran Q intrinsic procedures. For example, the QCOS intrinsic procedure for the generic COS intrinsic procedure.

The value of REAL (KIND=16) data is in the approximate range: 6.475175119438025110924438958227647Q-4966 to 1.189731495357231765085759326628007Q4932.

Unlike other floating-point formats, there is little if any performance penalty from using denormalized extended-precision numbers, since accessing denormalized numbers do not result in an arithmetic trap (extended-precision is emulated in software). (The smallest normalized number is 3.362103143112093506262677817321753Q-4932.)

The precision is approximately one part in 2**112 or typically 33 decimal digits.

The X_float format is emulated in software. Although there is no standard IEEE little endian 16-byte REAL data type, the X_float format supports IEEE exceptional values.

For more information, see the revised *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems* and the *Compaq Fortran Language Reference Manual*.

3.9.7 Variable Format Expressions (VFEs)

By enclosing an arithmetic expression in angle brackets, you can use it in a FORMAT statement wherever you can use an integer (except as the specification of the number of characters in the H field). For example:

```
J = 5
FORMAT (I<J+1>)
```

For more information, see the *Compaq Fortran Language Reference Manual*.

3.9.8 Notes on Debugger Support

Compaq Tru64 UNIX provides both the `dbx` and the Compaq Ladebug (formerly DECladebug) debuggers in the programming environment subsets.

These debuggers are very similar and use almost identical set of commands and command syntax. Both have a command-line interface as well as a Motif® windowing interface.

A character-cell Ladebug (ladebug) interface is provided with Ladebug in the Compaq Tru64 UNIX operating system Programmer's Development Toolkit. To use the character-cell interface, use the `ladebug` command.

When using Ladebug with certain versions of the UNIX operating system, be aware that a trailing underscore may be needed to display module variables. For example, to display variable X in module MOD, type:

```
print $MOD$X$_
```

The Parallel Software Environment supports debugging parallel HPF programs (see the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*). This section addresses scalar (nonparallel) debugging.

When using the `f90` command to create a program to be debugged using `dbx` or `ladebug`, consider using the following options:

- Specify `-g` or `-g2` to request symbol table and traceback information needed for debugging.
- Avoid requesting optimization. When you specify `-g` or `-g2`, the optimization level is set to `-O0` by default. Debugging optimized code is neither easy nor recommended.
- When using the Ladebug debugger, you should specify the `-ladebug` option. The `-ladebug` option allows you to print and assign to dynamic arrays using standard Fortran syntax.

For example, the following command creates the executable program `proj_dbg.out` for debugging with Ladebug:

```
% f90 -g -ladebug -o proj_dbg.out file.f90
```

You invoke the character-cell Ladebug debugger by using the `ladebug` command.

For more information, see the debugger chapter in the revised *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems* (Chapter 4).

3.9.8.1 Ladebug Debugger Support Notes

The following improvements in Ladebug support for the Compaq Fortran language were added for DIGITAL UNIX Version 4.0:

- Ladebug now includes a graphical window interface.
- Ladebug now supports the display of array sections.
- Ladebug now displays Fortran data types using Fortran 90 name syntax rather than C names (such as integer rather than int).
- Ladebug provides improved support for debugging mixed-language C and Fortran applications.
- These and other improvements are described in the debugger chapter (Chapter 4) of the *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*.

The following improvements in Ladebug support for the Fortran 90 language were added for DEC OSF/1 Version 3.2 (DECladebug V3.0-16):

- Fortran and Fortran 90 language expression evaluation is built into the Ladebug command language, including:
  - Case-insensitive identifiers, variables, program names, and so on
  - Logical expressions, including:

    Relational operators (.LT., .LE., .EQ., .NE., .GT., .GE.)
    Logical operators (.XOR., .AND., .OR., .EQV., .NEQV., .NOT.)
- Fortran 90 pointers
- Fortran 90 array support, including:
  - Explicit-shape arrays
  - Assumed-shape arrays
  - Automatic arrays
  - Assumed-size arrays
  - Deferred-shape arrays
- COMMON block support, including:
  - Display of whole common block
  - Display of (optionally-qualified) common block members
- COMPLEX variable support, including the display, assignment, and use of arithmetic expressions involving COMPLEX variables
- Alternate entry points, including breakpoints, tracepoints, and stack tracing (`where` command)
- Mixed-language debugging

For more information on using Ladebug, see the debugger chapter in the revised *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems* (Chapter 4).

3.9.8.2 dbx Debugger Support Notes

When using dbx with HP Fortran programs, certain differences exist. For example, in dbx, assumed-shape arguments, allocatable arrays, and pointers to arrays are printed as a derived type. Consider the following program:

```
module foo
      real x
contains
      subroutine bar(a)
      integer a(:)
      a(1) = 1
      end subroutine bar
end module foo

use foo
integer b(100)
call bar(b)
end
```

If the above program were stopped inside BAR, the following would occur:

```
(dbx) print a
   common /
     dim = 1
     element_length = 4
     ptr = 0x140000244
     ies1 = 4
     ub1 = 10
     lb1 = 1
   /
```

The meaning of the fields are:

dim - dimension of the object
element_length - the length of each element in bytes
ptr - the address of the object
ies*n* - distance (in bytes) between elements in the *n*th dimension
ub*n* - upper bound in the *n*th dimension
lb*n* - lower bound in the *n*th dimension

3.9.9 Notes on Fast Math Library Routines

The f90 option -math_library fast provides alternate math routine entry points to the following:

- SQRT, EXP, LOG, LOG10, SIN, and COS intrinsic procedures
- Power (**) in arithmetic expressions

3.9.10 The HP Fortran Array Descriptor Format

In the *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*, Chapter 10, Section 10.1.7 describes the Compaq Fortran array descriptor format.

These notes are an initial attempt to provide a template for those C programmers creating an a .h file that lays out the Fortran array descriptor format.

There are two varying parameters for this descriptor format:

- The element type (shown in this section as <ELEMENT_TYPE>)
- The rank (shown in this section as <RANK>)

Common information for all descriptors is the general layout of the header and the information for each dimension.

One possible C @codefont(struct) definition for the per-dimension information is:

```
struct _f90_array_dim_info {
  int inter_element_spacing;
  int pad1;
  int upper_bound;
  int pad2;
  int lower_bound;
  int pad3;
};
```

The inter-element spacing is measured in 8-bit bytes, not in array elements. This presents a challenge in designing array descriptor definitions in C, since there is no completely clean way to interact with C's pointer arithmetic.

One way to design the struct definition for an array descriptor is to use the template:

```
struct _f90_array_desc_rank<RANK>_<NAME_TOKEN> {
  unsigned   char   dim;
  unsigned   char   flags;
  unsigned   char   dtype;
  unsigned   char   class;
  int               pad;
  long              length;
  <ELEMENT_TYPE> *  pointer;
  long              arrsize;
  void *            addr_a0;
  struct _f90_array_dim_info dim_info[<RANK>];
};
```

Where <RANK>, <NAME_TOKEN> and <ELEMENT_TYPE> are the template parameters. Often <NAME_TOKEN> and <ELEMENT_TYPE> can be the same, but in cases where <ELEMENT_TYPE> has non-identifier characters in it (for example, space or star) then a suitable <NAME_TOKEN> should be devised.

The problem with this approach is that the element addressing, which uses the inter-element spacing, generates an offset in bytes. In order to use C's native pointer arithmetic, either casts need to be done or a division. For example:

- Casting:

  ```
  *((<ELEMENT_TYPE> *) (((char *) desc->pointer) + byte_offset))
  ```

- Division:

  ```
  (desc->pointer)[byte_offset/sizeof(<ELEMENT_TYPE>)]
  ```

Another way to design the struct definition for an array descriptor is to use the template:

```
struct _f90_array_desc_rank<RANK>_general {
  unsigned char  dim;
  unsigned char  flags;
  unsigned char  dtype;
  unsigned char  class;
  int            pad;
  long           length;
  char *         pointer;
  long           arrsize;
  void *         addr_a0;
  struct _f90_array_dim_info dim_info[<RANK>];
};
```

An advantage to this approach is that the same definition can be used for all arrays of the same rank. The problem with this approach is that it forces the programmer to cast:

```
*((<ELEMENT_TYPE> *) (desc->pointer + byte_offset))
```

Another approach is to remove <RANK> from the template as well, yielding:

```
struct _f90_array_desc_general {
   unsigned char  dim;
   unsigned char  flags;
   unsigned char  dtype;
   unsigned char  class;
   int            pad;
   long           length;
   char *         pointer;
   long           arrsize;
   void *         addr_a0;
   struct _f90_array_dim_info dim_info[7];
};
```

On the last line, 7 is used since that is the maximum rank allowed by Fortran. Since the dim field should be checked, this definition can be used in many (perhaps most) of the places a rank-specific definition would be used, provided the programmer is aware that the dim_info fields beyond the actual rank are undefined.

One place such a definition should NOT be used is when an object of this definition is used as part of an assignment. This usage is considered rare. For example:

```
  void
  ptr_assign_buggy(struct _f90_array_desc_general * ptr,
  struct _f90_array_desc_general * tgt)
  {
  *ptr = *tgt;
  }
```

Example of Array Descriptor Format Use

In this example, we have a 'struct tree' and a procedure prune_some_trees_( ) that takes a descriptor of a rank=3 array of such structs and calls prune_one_tree_( ) on each individual tree (by reference):

```
void
prune_some_trees(struct _f90_array_desc_general * trees)
{
  if (trees->dim != 3) {
      raise_an_error();
      return;
    } else {
      int x,y,z;
      int xmin = trees->dim_info[0].lower_bound;
      int xmax = trees->dim_info[0].upper_bound;
      int xstp = trees->dim_info[0].inter_element_spacing;
      int ymin = trees->dim_info[1].lower_bound;
      int ymax = trees->dim_info[1].upper_bound;
      int ystp = trees->dim_info[1].inter_element_spacing;
      int zmin = trees->dim_info[2].lower_bound;
      int zmax = trees->dim_info[2].upper_bound;
      int zstp = trees->dim_info[2].inter_element_spacing;
      int xoffset,yoffset,zoffset;
    for (z = zmin, zoffset = 0; z <= zmax; z+= 1, zoffset += zstp) {
      for (y = ymin, yoffset = 0; y <= ymax; y+= 1, yoffset += ystp) {
       for (x = xmin, xoffset = 0; x <= xmax; x+= 1, xoffset += xstp) {
           struct tree * this_tree =
            (struct tree *) (trees->pointer + xoffset+yoffset+zoffset);
         prune_one_tree_(this_tree);
        }
      }
    }
  }
}
```

Compaq would appreciate feedback on which definitions of array descriptors users have found most useful.

Note that the format for array descriptors used by HPF is more complicated and is not described at this time.

4

New Features for Versions Prior to Version 5

This chapter summarizes the new features for versions prior to Version 5.0:

- New Features and Corrections in Version 4.1 (Section 4.1)
- New Features in Version 4.0 (Section 4.2)
- New Features in Version 2.0 (Section 4.3)
- New Features in Version 1.3 (Section 4.4)
- New Features in Version 1.2 (Section 4.5)
- New Features in Version 1.1 (Section 4.6)

4.1 New Features and Corrections in Version 4.1

Version 4.1 is a maintenance release that contains a limited number of new features and corrections to problems discovered since Version 4.0 was released.

For additional information added to these release notes for Version 4.1, see Section 3.9.3.

The following new features have been added for DIGITAL Fortran 90 Version 4.1:

- This release includes a partial implementation of the proposed Fortran 95 standard.

  The following features of the proposed Fortran 95 standard have been implemented by this version of DIGITAL Fortran 90 and are supported when using the `f90` or `f95` commands:

  - FORALL statement and construct (implemented prior to Version 4.1)
  - Automatic deallocation of ALLOCATABLE arrays (implemented prior to Version 4.1)
  - Dim argument to MAXLOC and MINLOC (implemented prior to Version 4.1)
  - PURE user-defined subprograms (implemented prior to Version 4.1)
  - ELEMENTAL user-defined subprograms (a restricted form of a pure procedure)
  - Pointer initialization (initial value)
  - The NULL intrinsic to nullify a pointer
  - Derived-type structure initialization
  - CPU_TIME intrinsic subroutine
  - Kind argument to CEILING and FLOOR intriniscs
  - Enhanced SIGN intrinsic function

- Nested WHERE constructs, masked ELSEWHERE statement, and named WHERE constructs
- Comments allowed in namelist input
- Generic identifier in END INTERFACE statements
- Detection of Obsolescent and/or Deleted features listed in the proposed Fortran 95 standard

For more information on Fortran 95 features, see the Section 3.9.2.

- The `f95` command is now available for use with the `-std` option:
  - To perform standards conformance checking against the Fortran 90 standard, use the `f90` command with `-std`. Using `f90` with `-std` will issue messages for features (such as FORALL) that have recently been added to the proposed Fortran 95 standard (as well as other extensions to the Fortran 90 standard).
  - To perform standards conformance checking against the Fortran 95 standard, use the `f95` command with `-std`. Using `f95` with `-std` will not issue messages for features (such as FORALL) that have been added to the proposed Fortran 95 standard, but will issue messages for extensions to the Fortran 95 standard.

  For more information on Fortran 95 features, see the Section 3.9.2.
- The `-intconstant` option has been added.

  Specify the `-intconstant` option to use DIGITAL Fortran 77 rather than Fortran 90 semantics to determine kind of integer constants. If you do not specify `-intconstant`, Fortran 90 semantics are used.

  Fortran 77 semantics require that all constants are kept internally by the compiler in the highest precision possible. For example, if you specify `-intconstant`, an integer constant of 14 will be stored internally as INTEGER(KIND=8) and converted by the compiler upon reference to the corresponding proper size. Fortran 90 specifies that integer constants with not explicit KIND are kept internally in the default INTEGER kind (KIND=4 by default).

  Similarly, the internal precision for floating-point constants is controlled by the `-fpconstant` option.
- The `-pad_source` option has been added.

  Specify the `-pad_source` option to request that source records shorter than the statement field width are to be padded with spaces on the right out to the end of the statement field. This affects the interpretation of character and Hollerith literals that are continued across source records.

  The default is `-nopad_source`, which causes a warning message to be displayed if a character or Hollerith literal that ends before the statement field ends is continued onto the next source record. To suppress this warning message, specify the `-warn nousage` option.

  Specifying `-pad_source` can prevent warning messages associated with `-warn usage`.
- The `-warn usage` option has been added.

Specify the -warn nousage option to suppress warning messages about questionable programming practices which, although allowed, often are the result of programming errors. For example, a continued character or Hollerith literal whose first part ends before the statement field ends and appears to end with trailing spaces is detected and reported by -warn usage.

The default is -warn usage.

- The -arch *keyword* option has been added.

  This option determines the type of Alpha chip code that will be generated for this program. The -arch *keyword* option uses the same keywords as the -tune *keyword* option.

  Whereas the -tune *keyword* option primarily applies to certain higher-level optimizations for instruction scheduling purposes, the -arch *keyword* option determines the type of code instructions generated for the program unit being compiled.

  DIGITAL UNIX Version 4.0 and subsequent releases provide an operating system kernel that includes an instruction emulator. This emulator allows new instructions, not implemented on the host processor chip, to execute and produce correct results. Applications using emulated instructions will run correctly, but may incur significant software emulation overhead at runtime.

  All Alpha processors implement a core set of instructions. Certain Alpha processor versions include additional instruction extensions.

  Supported -arch keywords are as follows:

  - -arch generic generates code that is appropriate for all Alpha processor generations. This is the default.

    Running programs compiled with the generic keyword will run all implementations of the Alpha architecture without any instruction emulation overhead.

  - -arch host generates code for the processor generation in use on the system being used for compilation.

    Running programs compiled with this keyword on other implementations of the Alpha architecture might encounter instruction emulation overhead.

  - -arch ev4 generates code for the 21064, 21064A, 21066, and 21068 implementations of the Alpha architecture.

    Running programs compiled with the ev4 keyword will run without instruction emulation overhead on all Alpha processors.

  - -arch ev5 generates code for some 21164 chip implementations of the Alpha architecture that use only the base set of Alpha instructions (no extensions).

    Running programs compiled with the ev5 keyword will run without instruction emulation overhead on all Alpha processors.

  - -arch ev56 generates code for some 21164 chip implementations that use the byte and word manipulation instruction extensions of the Alpha architecture.

    Running programs compiled with the ev56 keyword might incur emulation overhead on ev4 and ev5 processors, but will still run correctly on DIGITAL UNIX Version 4.0 (or later) systems.

  - `-arch pca56` generates code for the 21164PC chip implementation that uses the byte and word manipulation instruction extensions and multimedia instruction extensions of the Alpha architecture.

    Running programs compiled with the pca56 keyword might incur emulation overhead on ev4 and ev5 and ev56 processors, but will still run correctly on DIGITAL UNIX Version 4.0 (or later) systems.

- In addition to ev4, ev5, generic, and host, The ev56 and pca56 keywords are now supported for the `-tune` option.

The following new High Performance Fortran features have been added for DIGITAL Fortran 90 Version 4.1:

- Transcriptive data distributions are now supported.
- The INHERIT directive can now be used to inherit distributions, as well as alignments.
- Distributed components of user-defined types are now handled in parallel. This is not part of standard High Performance Fortran (HPF), but is an approved extension.
- The GLOBAL_SHAPE and GLOBAL_SIZE HPF Local Library routines are now supported.
- There is a new compile-time option named `-show hpf`, which replaces the `-show wsfinfo` option. The `-show hpf` option provides performance information at compile time. Information is given about inter-processor communication, temporaries created at procedure boundaries, optimized nearest-neighbor computations, and code that is not handled in parallel. You can choose the level of detail you wish to see.
- New example programs are available in the following directory:

  ```
  /usr/examples/HPF
  ```

These new features are described in the *DIGITAL High Performance Fortran 90 HPF and PSE Manual*.

The corrections made for DIGITAL Fortran 90 Version 4.1 include the following:

- Fix compiler abort with certain types of pointer assignment.
- Fix incorrect error message for nested STRUCTUREs.
- Fix inconsistent severity for undeclared variable message with IMPLICIT NONE or command line switch.
- Fix incorrect error about passing LOGICAL*4 to a LOGICAL*1 argument.
- Add standards warning for non-integer expressions in computed GOTO.
- Do not flag NAME= as nonstandard in INQUIRE.
- Add standards warning for AND, OR, XOR intrinsics.
- VOLATILE attribute now honored for COMMON variables.
- Allow COMPLEX expression in variable format expression.
- Allow adjustable array to be declared AUTOMATIC (AUTOMATIC declaration is ignored.)
- Honor `-automatic` (/RECURSIVE) in main program.
- Fix incorrect parsing error when DO-loop has bounds of -32768,32767.

- Fix compiler abort when extending generic intrinsic.
- Fix SAVEd variable in inlined routine that didn't always get SAVEd.
- Fix compiler abort with initialization of CHARACTER(LEN=0) variable
- Correct values of PRECISION, DIGITS, etc. for floating types.
- Fix incorrect value of INDEX with zero-length strings.
- Correct value for SELECTED_REAL_KIND in PARAMETER statement.
- Correct compile-time result of VERIFY.
- For OpenVMS only, routine using IARGPTR or IARGCOUNT corrupts address of passed CHARACTER argument.
- Standards warning for CMPLX( ) in initialization expression.
- Fix compiler abort when %LOC(charvar) used in statement function.
- Fix incorrect initialization of STRUCTURE array.
- Fix compiler abort with large statement function.
- RESHAPE of array with a zero bound aborts at runtime.
- For OpenVMS only, /INTEGER_SIZE now correctly processed.
- SIZEOF(SIZEOF( )) is now 8.
- Fix error parsing a derived type definition with a field name starting with "FILL_".
- With OPTIONS /NOI4, compiler complained of IAND with arguments of an INTEGER*4 variable and a typeless PARAMETER constant.
- Fix incorrect standards warning for DABS.
- Add error message for ambiguous generic.
- Corrected error parsing field array reference in IF.
- Bit constants in argument lists are typed based on value, not "default integer".
- Allow module to use itself.
- Fix standards warning for Hollerith constant.
- For OpenVMS only, FOR$RAB is always INTEGER*4.
- For OpenVMS only, wrong values for TINY, HUGE for VAX floating.
- For OpenVMS only, EXPONENT( ) with /FLOAT=D_FLOAT references non-existent math routine.
- The Compaq Fortran run-time library incorrectly failed to release previously allocated memory when padding Fortran 90 input.
- The Compaq Fortran run-time library would incorrectly go into an infinite loop when an embedded NULL character value was found while performing a list-directed read operation.
- The Compaq Fortran run-time library would incorrectly treat an end-of-record marker as a value separator while performing a list-directed read operation.

- The Compaq Fortran run-time library incorrectly produced a "recursive I/O operation" error after a user has made a call to flush( ) to flush a unit which was not previously opened, then attempted to perform any I/O operation on the unit
- The Compaq Fortran run-time library would incorrectly fail to return an end-of-record error for certain non-advancing I/O operations. This occurred when attempting to read into successive array elements while running out of data.
- The Compaq Fortran run-time library, when it encountered the ":" edit descriptor at the end of the input record did not stop reading the next record, causing errors like "input conversion".
- The Compaq Fortran run-time library did not handle implied DO loops on I/O lists when non-native file support (`-convert` or equivalent conversion method) was in use.

The following are corrections for HPF users in this version:

- Expanded I/O Support, including support for all features, including: complicated I/O statements containing function calls, assumed size arrays, or variables of derived types with pointer components, and array inquiry intrinsics using the implied DO loop index.

  In addition, non-advancing I/O (except on stdin and stdout) now works correctly if every PSE peer in the cluster has a recent version of the Fortran run-time library (fortrtl_371 or higher).
- NUMBER_OF_PROCESSORS and PROCESSORS_SHAPE in EXTRINSIC(HPF_SERIAL) routines
- Restriction lifted on user-defined types in some FORALLs
- Declarations in the specification part of a module
- EXTRINSIC(SCALAR) changed to EXTRINSIC(HPF_SERIAL)

These new corrections are described in more detail in the Parallel Software Environment (PSE) release notes.

4.2 New Features in Version 4.0

The following `f90` command options were added for DIGITAL Fortran 90 Version 4.0:

- Specify the `-assume byterecl` option to::
  - Indicate that the OPEN statement RECL unit for unformatted files is in byte units. If you omit `-assume byterecl`, HP Fortran expects the OPEN statement RECL value for unformatted files to be in longword (four-byte) units.
  - Return the record length value for an INQUIRE by output list (unformatted files) in byte units. If you omit `-assume byterecl`, HP Fortran returns the RECL value for an INQUIRE by output list in longword (four-byte) units.
- The `-check nopower` option allows arithmetic calculations that result in 0**0 or a negative number raised to an integer power of type real (such as –3**3.0) to be calculated, rather than stop the program. If you omit `-check nopower`

for such calculations, an exception occurs and the program stops (default is `-check:power`).

For example, if you specified `-check:nopower`, the calculation of the expression 0**0 results in 1 and the expression –3**3.0 results in –9.

- Specify `-hpf_matmul` to use matrix multiplication from the HPF library. Omitting the `-hpf_matmul` option uses inlined intrinsic code that is faster for small matrices. For nonparallel compilations, specifying `-hpf_matmul` to use the HPF library routine is faster for large matrices.
- The `-names` *keyword* option controls how DIGITAL Fortran 90 handles the case-sensitivity of letters in source code identifiers and external names:
  - Using `-names as_is` requests that HP Fortran distinguish between uppercase and lowercase letters in source code identifiers (treats uppercase and lowercase letters as different) and distinguish between uppercase and lowercase letters in external names.
  - Using `-names lowercase` (default) requests that HP Fortran *not* distinguish between uppercase and lowercase letters in source code identifiers (treats lowercase and uppercase letters as equivalent) and force all letters to be *lowercase* in external names.
  - Using `-names uppercase` requests that HP Fortran *not* distinguish between uppercase and lowercase letters in source code identifiers (treats lowercase and uppercase letters as equivalent) and force all letters to be *uppercase* in external names.
- The `-noinclude` option prevents searching for include files in the `/usr/include` directory. This option does *not* apply to the directories searched for module files or `cpp` files.
- The `-O5` option activates both the software pipelining optimization (`-pipeline`) and the loop transform optimizations (`-transform_loops`). You can also specify `-notransform_loops` or `-nopipeline` with `-O5`.

  If you also specify the `-wsf` option to request parallel processing, you cannot use the `-O5` option.
- The `-pipeline` option activates the only software pipelining optimization (previously done only by `-O5`). The software pipelining optimization applies instruction scheduling to certain innermost loops, allowing instructions within a loop to "wrap around" and execute in a different iteration of the loop. This can reduce the impact of long-latency operations, resulting in faster loop execution.

  Software pipelining also enables the prefetching of data to reduce the impact of cache misses. In certain cases, software pipelining improves run-time performance (separate timings are suggested).
- The following `-reentrancy` *keyword* options specify the level of thread-safe reentrant run-time library support needed:

Option Name	Description
`-reentrancy none`	Informs the Compaq Fortran RTL that the program will not be relying on threaded or asynchronous reentrancy. Therefore the RTL need not guard against such interrupts inside the RTL. This is the default.

Option Name	Description
-reentrancy asynch	Informs the Compaq Fortran RTL that the program may contain asynchronous handlers that could call the RTL. Therefore the RTL will guard against asynchronous interrupts inside its own critical regions.
-reentrancy threaded	Informs the Compaq Fortran RTL that the program is multithreaded, such as those using the DECthreads library. Therefore the RTL will use thread locking to guard its own critical regions. To use the threaded libraries, also specify -threads.
-noreentrancy	The same as -reentrancy none.

- The -S option generates a .s file, which can be assembled. This option is intended for systems running Compaq Tru64 UNIX (DIGITAL UNIX) Version 4.0 or later, which has certain new Assembler features.

  Certain complex programs that use modules or common blocks compiled with -S may not generate code completely acceptable to the Assembler.

- The -speculate *keyword* option supports the speculative execution optimization:

  - Use -speculate all to perform the speculative execution optimization on all routines in the program. All exceptions within the entire program will be quietly dismissed without calling any user-mode signal handler.
  - Use -speculate by_routine to perform the speculative execution optimization on all routines in the current compilation unit (set of routines being compiled), but speculative execution will not be performed for routines in other compilation units in the program.
  - Use -speculate none or -nospeculate to suppress the speculative execution optimization. This is the default.

  The speculative execution optimization reduces instruction latency stalls to improve run-time performance for certain programs or routines. This optimization evaluates conditional code (including exceptions) and moves instructions that would otherwise be executed conditionally to a position before the test, so they are executed unconditionally.

  Speculative execution does not support some run-time error checking, since exception and signal processing (including SIGSEGV, SIGBUS, and SIGFPE) is conditional. When the program needs debugging or while testing for errors, use -speculate none (default).

- Specifying -threads requests that the linker use threaded libraries. This is usually used with -reentrancy threaded.

- The -transform_loops option supports a group of optimizations that improve the performance of the memory system and can apply to multiple nested loops. The loops chosen for loop transformation optimizations are always **counted loops** (counted loops include DO or IF loops, but not uncounted DO WHILE loops). In certain cases, loop transformation improves run-time performance (separate timings are suggested).

- Specify -nowsf_main to indicate that the HPF global routine being compiled will be linked with a main program that was not compiled with -wsf.

For more information on f90 command options, see the *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*, Chapter 3, or f90(1).

In addition to the `f90` command-line options, the following new or changed features were added for Version 4.0:

- The random_number intrinsic (as of Version 4.0) uses two separate congruential generators together to produce a period of approximately 10**18, and produces real pseudorandom results with a uniform distribution in (0,1). It accepts two integer seeds, the first of which is reduced to the range [1, 2147483562]. The second seed is reduced to the range [1, 2147483398]. This means that the generator effectively uses two 31-bit seeds.

  The new algorithm behaves differently from one provided prior to Version 4.0 in the following ways:

  - Both seeds are active and contribute to the random number being produced.
  - If the given seeds are not in the ranges given above, they will be reduced to be in those ranges.
  - The sequences of random numbers produced by the new generator will be different from the sequences produced by the old generator.

  For more information on the algorithm, see:

  - *Communications of the ACM* vol 31 num 6 June 1988, entitled *Efficient and Portable Combined Random Number Generators* by Pierre L'ecuyer
  - *Springer-Verlag* New York, N. Y. 2nd ed. 1987, entitled *A Guide to Simulation* by Bratley, P., Fox, B. L., and Schrage, L. E.

  For an example program, see Section 3.9.4.

- The implementation of the MATMUL intrinsic procedure was changed for this release. Previously the compiler called a routine in the scalar HPF library to perform the operation. As of this release, the compiler generates optimized inline code for the MATMUL intrinsic with a significant increase in the performance when the size of the array arguments are small.

  To use previous implementation of the MATMUL intrinsic (routine in the scalar HPF library), specify `-hpf_matmul`.

- The cDEC$ ALIAS directive

  The cDEC$ ALIAS directive is now supported in the same manner as in Compaq Fortran 77. This directive provides the ability to specify that the external name of an external subprogram is different than the name by which it is referenced in the calling subprogram.

  This feature can be useful when porting code between OpenVMS and UNIX systems where different routine naming conventions are in use.

  For more information on the cDEC$ ALIAS directive, see the *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*.

- The cDEC$ ATTRIBUTES directive

  The cDEC$ ATTRIBUTES directive lets you specify properties for data objects and procedures. These properties let you specify how data is passed and the rules for invoking procedures. This directive is intended to simplify mixed language calls with HP Fortran routines written in C or Assembler.

  For more information on the cDEC$ ATTRIBUTES directive, see *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*.

- An additional math library allows use of optimizations for a series of square root calculations.

  The library file `libm_4sqrt` ships on the DIGITAL Fortran 90 Version 4.0 kit (and DIGITAL Fortran 77 Version 4.0). These optimizations improve run-time performance when a series of square root calculations occur within a counted loop.

- Enhanced support for the FORALL statement and construct

  The FORALL construct now allows the following statements in the forall body:

  - Pointer assignment statement
  - FORALL statement or construct (nested FORALL)
  - WHERE statement or construct

  Please note that each statement in the FORALL body is executed completely before execution begins on the next FORALL body statement.

  The compiler now correctly defines the scope of a FORALL subscript name to be the scope of the FORALL construct. That is, the subscript name is valid only within the scope of the FORALL. Its value is undefined on completion of the FORALL construct.

- OPTIONS statement options can now be abbreviated (for compatibility with DIGITAL Fortran 77).
- The `-vms` option now supports use of /LIST or /NOLIST in an INCLUDE statement (for compatibility with DIGITAL Fortran 77).
- To improve run-time performance, new optimizations are now available and certain improvements have been made, including:
  - Certain intrinsic procedures specific to Fortran 90 (not available in FORTRAN-77)
  - Subprogram calls with array arguments
  - New command-line options that activate new optimizations, including the loop transformation optimizations (`-transform_loops` or `-O5`) and the speculative execution optimization (`-speculate` *keyword*). The software pipelining optimization is now activated by using `-pipeline` or `-O5`.
- Variable formats expressions (VFEs) are now allowed in quoted strings.
- Invalid formats in quoted strings are now detected at compile-time rather than run-time.

For more information on compatibility with DIGITAL Fortran 77, see the revised *Compaq Fortran User Manual for Tru64 UNIX and Linux Alpha Systems*, Appendix A.

4.3 New Features in Version 2.0

New features for Version 2.0 include the LOC intrinsic function. LOC returns the internal address of its argument (same as the built-in function %LOC).

In addition, the Compaq Ladebug debugger has added support for Compaq Fortran language features (see Section 3.9.8.1).

4.4 New Features in Version 1.3

New features for Version 1.3 include the *f90* command options that support the Compaq Parallel Software Environment.

To request parallel execution, specify the `-wsf` or `-wsf` *nn* option . This compiles the program to run in parallel using the Compaq Parallel Software Environment product. The optional *nn* parameter specifies the number of processors on which the program is intended to execute. If not specified, the program will be compiled to execute on any number of processors. More efficient code is generated when *nn* is specified.

If you specify the `-wsf` or `-wsf` *nn* option to request parallel execution, you can also use the following related options:

- The `-assume nozsize` option assumes there are no zero-sized array sections.
- The `-nearest_neighbor` or `-nearest_neighbor` *nn* option enables or disables the nearest neighbor parallel optimization. The optional *nn* parameter specifies the width of the shadow edge to use. If you omit *nn*, it is set to 1.
- The `-pprof` *string* option allows parallel profiling of an application. Valid characters for *string* are s for sampling or i for interval. This option must be used with the `-non_shared` option (as well as `-wsf` or `-wsf` *nn*). This option must not be used with the `-p1` option.
- The `-show wsfinfo` option includes information about statements which cause interprocessor communication to be generated or are serialized in the listing file.

Other Version 1.3 new features include the following:

- Support for the DIGITAL Fortran 77 pointers (CRAY® style). This is an extension to the Fortran 95/90 and FORTRAN-77 standards. For more information, see the *DEC Fortran Language Reference Manual* and Section 3.9.5.
- Bit constants with a trailing B or Z or leading X (a Compaq Fortran extension) are now supported for compatibility with Compaq Fortran 77:

```
i = '001'B
k = '0ff'Z
j = X'00f'
```

- The SYSTEM_CLOCK intrinsic procedure has been extended to allow integer arguments of any KIND rather than the default integer KIND . This allows the use of INTEGER*8 arguments to obtain a higher degree of magnitude and accuracy in timings (1,000,000 counts per second). For example:

```
integer*8 count,count_max,count_rate
call system_clock(count,count_rate,count_max)
```

- When it is passed an INTEGER (KIND=4) value, the SYSTEM_CLOCK intrinsic procedure now returns a value in terms of 10,000 instead of 1,000,000 counts per second.
- Debugging support has been enhanced to allow breakpoints on CONTINUE, GOTO, and RETURN statements. Before Version 1.3, breakpoints could not be set on a CONTINUE statement and only on certain GOTO and RETURN statements.

- The following DIGITAL Fortran 90 cDEC$ directives are now supported:
  - cDEC$ IDENT specifies a string that identifies the object file.
  - cDEC$ OPTIONS and cDEC$ END_OPTIONS controls alignment of fields in common blocks, record structures, and most derived-type structures.
  - cDEC$ PSECT modifies certain attributes of a common block, including the [NO]MULTILANGUAGE attribute for compatibility with DIGITAL Fortran 77.
  - cDEC$ TITLE and cDEC$ SUBTITLE specifies strings for the title and subtitle of a listing file header.
- Any number raised to a floating point 2.0 (x ** 2.0) is now transformed to (x ** 2) for compatibility with DIGITAL Fortran 77.
- The Bessel function 3f library (jacket) routines are now supported (see *bessel(3f)*)
- The following `f90` command options were added for Version 1.3:
  - The `-fuse_xref` option requests that DIGITAL Fortran 90 generate a data file that the Compaq FUSE Database Manager uses to create a cross-reference database file. This improves the performance of the Compaq FUSE Call Graph Browser and Cross-Referencer that use the database file for their operations.
  - The `-inline speed` and `-inline size` options have been added in place of `-inline automatic` to provide more control over procedure inlining:

    Use `-inline size` (same as `-inline space`) to inline procedures that will likely improve run-time performance where inlining will not significantly increase program size. This option is meaningful only at optimization levels `-O1` and higher.

    Use `-inline speed` to inline procedures that will likely improve run-time performance where inlining may significantly increase program size. Using `-inline speed` often results in larger executable program sizes (than `-inline size`). This type of inlining occurs automatically with the `-O4` or `-O5` optimization levels. This option is meaningful only at optimization levels `-O1` and higher.

    Other `-inline` *xxxx* options include `-inline none`, `-inline manual`, and `-inline all` (see Section 4.5).
  - The `-ladebug` option includes additional symbolic information in the object file for the DIGITAL Ladebug debugger (see `ladebug`(1). This option enables Ladebug to print and assign to dynamic arrays using standard Fortran syntax, including array sections.
  - The `-show map` option includes a symbol map in the listing file (also specify `-V`).
  - The `-version` option displays DIGITAL Fortran 90 version number information.

For more complete product information, see the Compaq Fortran documentation and the `f90`(1) reference (man) page.

4.5 New Features in Version 1.2

DIGITAL Fortran 90 Version 1.2 contains the following changes since Version 1.1:

- Support for REAL (KIND=16) (or REAL*16) X_float (extended precision) data type and its associated intrinsics (a DIGITAL Fortran extension). For more information see Section 3.9.
- Support for variable format expressions (VFEs), a DIGITAL Fortran extension (see Section 3.9).
- Support for OPTIONS statements, which allow you to specify command-line options in your source files. The OPTIONS statement is a DIGITAL Fortran extension.
- Intrinsic procedures FP_CLASS and IMAG (a DIGITAL Fortran extension).
- STATIC and AUTOMATIC declaration attributes and statements (a DIGITAL Fortran extension).
- The following `f90` command options were added for Version 1.2:
  - The `-convert fgx` and `-convert fdx` options allow conversion of unformatted OpenVMS Alpha DIGITAL Fortran 77 data files. Similarly, the FDX and FGX keywords are recognized for the OPEN statement CONVERT keyword and the FORT_CONVERT*n* environment variable names.

    Specifying `-convert fdx` indicates the data contains::
    - Little endian integer format (INTEGER declarations of the appropriate size)
    - REAL*4 and COMPLEX*8 data in VAX F_float format
    - REAL*8 and COMPLEX*16 data in VAX D_float format
    - REAL*16 data in native X_float format

    Specifying `-convert fgx` indicates the data contains:
    - Little endian integer format (INTEGER declarations of the appropriate size)
    - REAL*4 and COMPLEX*8 data in VAX F_float format
    - REAL*8 and COMPLEX*16 data in VAX G_float format
    - REAL*16 data in native X_float format
  - The `-double_size 128` option specifies that DOUBLE PRECISION declarations are implemented as extended precision REAL (KIND=16) data rather than double precision REAL (KIND=8) data.
  - The `-real_size 128` and `-r16` options allow a REAL declaration to be interpreted using the REAL (KIND=16) data type.
  - The `-inline` *xxxxx* options can be used to specify the type of inlining done independent of the `-O`*n* option (optimization level) specified:
    - To prevent inlining of procedures (except statement functions), use `-inline none` or `-inline manual`.

      This is the type of inlining done with `-O0`, `-O1`, `-O2`, or `-O3`.

- The -inline automatic option was replaced at Version 1.3 with -inline size and -inline speed (see Section 4.4), allowing more control over inlining.
- To inline every call that can possibly be inlined while generating correct code, including: statement functions, procedures that HP Fortran thinks will improve run-time performance, and any other procedures that can possibly be inlined while generating correct code (certain recursive routines cannot be inlined), use -inline all. This option is meaningful only at optimization levels -O1 and higher.

– The -gen_feedback option requests additional profiling information needed for feedback file use. You can use -gen_feedback with any optimization level up to -O3 (to avoid inlining procedures). If you omit a -O*n* option, the -gen_feedback option changes the default optimization level to -O0.

  A typical command-line sequence to create a feedback file (profsample.feedback) follows:

```
% f90 -gen_feedback -o profsample -O3 profsample.f90
% pixie profsample
% profsample.pixie
% prof -pixie -feedback profsample.feedback profsample
```

– The -feedback option now works with -cord or separately without -cord to specify a previously-created feedback file. For example:

```
% f90 -feedback profsample.feedback -o profsample -O3 profsample.f90
```

  The feedback file provides the compiler with actual execution information, which the compiler can use to perform such optimizations as inlining function calls.

  The same optimization level (-O*n* option) must be specified for the f90 command with the -gen_feedback option and the f90 command with the -feedback *name* option.

  You can use the feedback file as input to the f90 compiler and cord, as follows:

```
% f90 -cord -feedback profsample.feedback -o profsample -O3 profsample.f90
```

– The -tune *keyword* option selects processor-specific instruction tuning for implementations of the Alpha architecture. Regardless of the setting of -tune *keyword*, the generated code will run correctly on all implementations of the Alpha architecture. Tuning for a specific implementation can improve run-time performance; it is also possible that code tuned for a specific target may run slower on another target.

  Choose one of the following:

  - To generate and schedule code that will execute well for both types of chips, use -tune generic. This provides generally efficient code for those cases where both types of chips are likely to be used. If you do not specify any -tune *keyword* option, -tune generic is used (default).
  - To generate and schedule code optimized for the type of chip in use on the system being used for compilation, use -tune host.
  - To generate and schedule code optimized for the 21064, 20164A, 21066, and 21068 implementations of the Alpha chip, use -tune ev4.

- To generate and schedule code optimized for the 21164 implementation of the Alpha chip, use `-tune ev5`.

– The `-check noformat` option disables the run-time message (number 61) associated with format mismatches. It also requests that the data item be formatted using the specified descriptor, unless the length of the item cannot accommodate the descriptor (for example, it is still an error to pass an INTEGER (KIND=2) item to an E edit descriptor). Using `-check noformat` allows such format mismatches as a REAL (KIND=4) item formatted with an I edit descriptor.

  If you omit the `-vms` option, the default is `-check noformat`.

  If you specify `-vms` and omit `-check noformat`, `-check format` is used.

– The `-check output_conversion` option disables the run-time message (number 63) associated with format truncation. The data item is printed with asterisks. Error number 63 occurs when a number could not be output in the specified format field length without loss of significant digits (format truncation).

  If you omit the `-vms` option, the default is `-check nooutput_conversion`.

  If you specify `-vms` and omit `-check nooutput_conversion`, `-check output_conversion` is used.

– The `-vms` option now sets defaults for `-check output_conversion` and `-check format`.

For more complete product information, see the Compaq Fortran documentation and the `f90`(1) reference (man) page.

4.6 New Features in Version 1.1

DIGITAL Fortran 90 Version 1.1 contains the following changes since Version 1.0:

- The following `f90` command options were added for Version 1.1:
  – The `-check bounds` option generates additional code to detect out-of-bounds subscripts for array operations and character substring expressions at run-time. Use this option for debugging purposes.
  – The `-I`*dir* option specifies an additional directory to be searched for files specified with an INCLUDE statement or module files. For Version 1.0, this option specified an additional directory searched for module files only.
  – The `-warn argument_checking` option issues a warning message about argument mismatches between the calling and the called procedure when both program units are compiled together.
- The `fsplit` command now accepts DIGITAL Fortran 90 free-form source files (see `fsplit`(1)). For example:

```
% fsplit -f90 -free bigfile.f90
```

For more complete product information, see the Compaq Fortran documentation and the `f90`(1) reference (man) page.