

Compaq C Compiler Version 6.2 for Linux Alpha

README

December 1999

This is the README for Version 6.2.9.002-2 of the *Compaq C* compiler for Linux *Alpha*.

**Compaq Computer Corporation
Houston, Texas**

Compaq Computer Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Compaq or an authorized sublicensor.

© 1999 Compaq Computer Corporation. All Rights Reserved.

This document was prepared using DECdocument, Version 3.2-1n.

Contents

1	Introduction	1
2	Supported Linux Alpha Distributions	1
3	Summary of Changes from 10/15 Public Beta	1
4	Contents of the Compaq C CD-ROM	2
5	Installation Requirements and Instructions	3
6	Invoking the Compiler	4
6.1	Simple Optimization	4
6.2	Using ccc with .s or .S Files	4
7	Documentation	5
8	Redistributing Runtime Libraries with Applications	5
9	Known Problems	6
10	gcc Compatibility	6
10.1	Unsupported Extensions	7
10.2	Supported or Partially Supported Extensions	8
10.3	Likely Future Extensions	10
11	Downloading Updates and Learning about Related Products	10
12	Technical Support	10

1 Introduction

CCC is the *Compaq C* compiler for Linux *Alpha*. It is a port of the same compiler that is available on the Compaq *Tru64 UNIX* platform (and also on *OpenVMS Alpha*). The compiler produces excellent optimized code for the *Alpha* architecture, particularly for floating-point intensive applications.

There are some specific limitations and differences relative to the C compiler for Compaq's *Tru64 UNIX*, including:

- No support for feedback-based optimization.
- No support for parallel programming (that is, no OpenMP).
- No support for Tru64-based performance tools like *CORD*, *atom*, or *om*.
- No `-[x]taso` features or `pointer_size` pragmas.
- No structured exception handling.
- The `wchar_t` type on Linux is unsigned int; on *Tru64 UNIX* it is signed.
- The long double data type is the same as double (*Tru64 UNIX* Version 5.0 makes long double a 16-byte IEEE quad-precision type).
- The `-ieee` command-line option on Linux has additional properties that are not present on *Tru64 UNIX* systems. See the CCC manpage for more information.
- Compiler-generated `.S` files cannot be assembled by the assembler.

While the compiler accepts some GCC extensions, and successfully builds about 70% of the SRPMS coded in C for *Alpha* that are in the Red Hat 5.2 distribution, the compiler is primarily a standard C and K&R compiler. To the extent that GCC extensions are supported, they are mostly limited to features that are in common with the forthcoming C99 revision to the C standard, such as variable-length arrays and initializers with designators, and to features commonly encountered in source packages and header files even when configured for using a compiler other than GCC.

2 Supported Linux Alpha Distributions

Compaq C Version 6.2 has been tested primarily under the Red Hat 5.2 Linux distribution for *Alpha*, which comes with `glibc-devel-2.0.7` and `egcs-1.0.3a`. There has also been some limited testing under Red Hat 6.0 and SuSE 6.1, which come with `glibc-devel-2.1.1-7` and `egcs-1.1.2-12`. It may work with other distributions and other versions of `glibc-devel`, but no other configurations have been tested.

The basic support mechanism for *Compaq C* is provided by the UNH web site identified at the end of this document. In addition, Compaq offers per-incident support for specific Linux distributions, which can be extended to include *Compaq C* on those distributions (for an additional charge). More information on this option will be forthcoming.

3 Summary of Changes from 10/15 Public Beta

- New versions of `libots`, `cpml_ev5`, `cpml_ev6`, and `ladebug` are provided. If you have beta versions of any of these installed, you should erase those versions (using `rpm -e`) before installing the new ones. Upgrading from the beta versions to product versions using the `-U` option could result in problems.

- If the package `compat-glibc-5.2-2.0.7.1.alpha.rpm` provided in updates to the Red Hat 6.0 distribution was installed before CCC was installed, the CCC installation would add `-L /usr/alpha-glibc20-linux/lib` to its `comp.config` file. This would cause all linker invocations through CCC to fail looking for `/usr/i386-glibc20-linux/lib/libc.so.6`.

The failure was caused by a bug in the `compat-glibc` package itself. But since the updates did not cause `gcc` to search `/usr/alpha-glibc20-linux/lib` by default, this version of `ccc` will not do that either. Therefore, the problem does not occur in this version. The bug in `compat-glibc` was reported to the maintainer, but we do not know the status of the package.

- The `-threads` command-line option now produces an informational message and is ignored because the *Tru64 UNIX* behavior of this option is not really available on Linux, and the option itself is obsolete.
- A new CCC command-line option, `-version`, allows you to select which version of CCC to use (see `CCC(1)`). This is intended for use only when newer versions of the `ccc` package become available. You cannot use it to select the previous beta versions (which should be erased with `rpm -e`). In order to use the option, newer versions would need to be installed with `rpm -i --replacefiles` instead of `rpm -U`.
- In order to support the use of CCC with `imake`, the compiler recognizes exactly one special case of `gcc`-style inline asm. If the construct `asm(".weak <symbol-name>")` appears as a file scope declaration, it makes the named symbol a weak symbol.
- A new preprocessor directive, `#pragma assert`, has been added. See the *Programmers Guide* for more information. Note that this is not directly related to the `gcc #assert` directive. It provides assertions about code that can be used by the optimizer or to report at compile-time bad values for compile-time constant expressions.
- The `protect_headers_setup(8)` man page, and the script itself, have been modified to work for Linux.

4 Contents of the Compaq C CD-ROM

The *Compaq C* CD-ROM contains in its root directory the software packages for installation, as well as a `README` file that points to this `README` file (the file you are reading) and to other documentation files under a directory named `/docs`. The `/docs` directory contains documentation files for the different software components, each in its own subdirectory named after its package name.

The software components are:

- Common Compiler Support Libraries (package `libots`)
- Compaq Portable Math Library (packages `cpml_ev5` and `cpml_ev6`)
- *Compaq C* compiler (package `CCC`)
- Compaq Ladebug debugger (package `ladebug`)

Use the supplied Red Hat Package Manager (RPM) files to install each of the components listed above.

5 Installation Requirements and Instructions

To use *Compaq C*, the following packages should be installed in the order listed below.

1. `libots-2.2.7-2.alpha.rpm`

Compaq C depends on the `libots` runtime library provided for Compaq's *Alpha* Linux compilers in the `libots` package. You need to install `libots` in order to use the compiler, and `libots` must be installed before `cpml` is installed.

2. `cpml` library

The `CCC` command adds `-lcpml` to the options passed to the linker whenever you specify `-lm` or `-ieee` on the command line, so one of the two following packages also needs to be installed:

- `cpml_ev5-5.0.0-1.alpha.rpm`

This package has code that runs best on *Alpha* chips prior to the EV6 (21264) processor. The code will also run on EV6 machines, but the `ev6` version of the package provides code that runs significantly faster on EV6 processors.

- `cpml_ev6-5.0.0-1.alpha.rpm`

This package provides a library that is highly tuned for the EV6 processor, but it will not run on earlier processors (it might produce "Illegal instruction" traps).

3. `ccc-6.2.9.002-2.alpha.rpm`

The C compiler package.

4. `ladebug-4.0.59-6.alpha.rpm`

This package is optional. The code produced by `CCC` can be debugged by `gdb`. But for those accustomed to the *Tru64 UNIX* `ladebug` debugger, and because of its superior support for Fortran debugging, `ladebug` has been ported to Linux *Alpha*. `Ladebug` might also work better than `gdb` on some `CCC`-compiled code because `gcc` does not yet generate DWARF2 debugging records on Linux *Alpha* by default.

To install each of these packages (if the package is not already installed), use the following command:

```
rpm -i package-file-name
```

To see if any of these are already installed, and what versions they are, use the following command:

```
rpm -q libots cpml_ev5 cpml_ev6 ladebug ccc
```

If you already have a package of sufficient version installed (that is, because you have installed Compaq Fortran), there is no need to reinstall. But if you have older versions from any of the public beta tests, erase them first with `rpm -e`.

6 Invoking the Compiler

The command line for invoking the compiler and linker is mostly compatible with both the gcc compiler and the *Tru64 UNIX* compiler. Where possible, command-line options for gcc are translated to near-equivalents for the *Tru64 UNIX* compiler. Options that do not have *Tru64 UNIX* equivalents are silently ignored by default. But overall, there is a reasonable probability that a makefile that works on Linux *Alpha* using the gcc compiler could be used to build the same application with the CCC compiler by changing only the compiler invocation command from gcc (or CC) to CCC.

There is also a reasonable probability that a *Tru64 UNIX* application will build on Linux *Alpha* from its *Tru64 UNIX* makefile (if it is GNU compatible) by changing only the compiler invocation command from CC to CCC.

A quick way to try building with CCC is to set an environment variable named CC to CCC, and then invoke make with the -e option to override the makefile's definition of the CC macro. In order to find out just what the CCC compiler is doing with your command-line options, you can add -V to the command line to see the programs it is invoking and the options it is passing to them. You can also set an environment variable named DRV_DUMP to the value 1, which causes the compiler to report any gcc options that are simply being parsed and consumed by the compiler driver without having any effect.

6.1 Simple Optimization

If you can successfully build your application with CCC, you might want to try making it run faster. There is a good chance that you will get some speedup over gcc by default. But if you've really tuned your build with gcc-specific optimization controls that ccc will ignore, you might have to do some tuning with CCC to see improvements.

A good first attempt would be just to add -fast to the beginning of your compiler options. Note that the -fast option by default tells the compiler to generate the fastest code for the machine the compiler is running on. So if you compile on an advanced processor like EV6 (e.g. DS10), the code might produce "Illegal instruction" traps if you try to run it on an older machine. If you want to try some quick tuning but want to make sure the result will run on any *Alpha* machine, specify -fast -arch generic. You might also want to try adding -O4.

The CCC(1) man page and Programmer's Guide contain more information about optimization controls and tuning. See the documentation section for the location and status of the documentation in this kit.

6.2 Using ccc with .s or .S Files

The CCC compiler driver has only rudimentary support for handling assembly language source files (.s or .S files). It invokes the native preprocessor (/lib/cpp) and assembler (/usr/bin/as) to handle these types of files, but it does not understand all of the command-line options that may be passed to them. To pass non-CCC switches through the CCC command line to the assembler, prepend -Wa, to the switch, and change any blanks that separate switch letters from their arguments into commas. To pass non-CCC switches through to the preprocessor, prepend -Wp, instead of -Wa,.

For example, to pass -Wall to the preprocessor, enter CCC -Wp, -Wall foo.S. This results in -Wall being passed to the /lib/cpp command line.

Switches that are handled without the need for `-Wp`, or `-Wa`, include:

- `-C`
- `-D`
- `-E`
- `-I`
- `-U`

If you are having problems, you can use the `-V` switch to see the transformed command line. From there you can verify that the switches made it through the transformation. If not, then use the `-Wp`, or `-Wa`, prefixing described above. Alternatively, you might want to change your build to avoid using CCC to process assembly language source files, and invoke gcc directly.

7 Documentation

The man pages provided are `ccc(1)` and `protect_headers_setup(8)`.

A Programmer's Guide and a Language Reference Manual are provided in both HTML and PDF format (and there is also a plain text version of the Language Reference Manual). The CCC package installs these in subdirectories `Programmers_Guide` and `Language_Reference` under `/usr/doc/ccc-6.2.9.002`, with an `index.htm` file in the parent directory. All of the files for each document (`.htm`, `.pdf`, and `.txt`) are in the document's subdirectory. On the CDROM, the subdirectories are under `/docs/ccc`.

Documentation for `cpml` is installed under `/usr/doc/cpml_ev?-5.0.0`. On the CDROM it is in `/docs/cpml`.

Documentation for `ladebug` is a man page, `ladebug(1)`, and what is installed under `/usr/doc/ladebug-4.0.59`. On the CDROM, it is in `/docs/ladebug`.

`libots` is undocumented.

8 Redistributing Runtime Libraries with Applications

Users with a valid license are permitted to redistribute the `libots` and `cpml` runtime library packages included in the product, as follows.

If the application is linked with the `-non_shared` option, or is linked against the archive library (`.a`) form of these libraries, no redistribution is necessary (although it is permitted).

If the application is linked `-call_shared` and depends on `libots.so` and/or `libcpml.so`, then it is necessary for you to provide these libraries along with your application. The libraries should be redistributed in the form of the original RPMs (or updated versions of them) as provided by Compaq, and the users of the application should be advised to install them on the systems that will run the application.

If you redistribute `cpml_ev6`, you need to caution your users that it should only be installed on an EV6 or newer version of the *Alpha* processor. The `cpml_ev5` package can be installed on any version of the *Alpha* processor, but will not provide maximum performance on EV6 or newer processors.

Applications built by Compaq C on Linux *Alpha* systems running the Red Hat 5.2 or 6.0 distributions can be distributed for use on those systems. Distributing those applications on other versions or other distributions of Linux *Alpha* has not been tested (although there has been limited testing with the SuSE 6.1 distribution).

9 Known Problems

The following are known problems that we are working on—no need to report these:

- Problems with gcc command-line options for the assembler or preprocessor, as noted above under "Using ccc with .s or .S files".
- The preprocessor token-pasting operator has problems when one of its arguments is a pp-num like 0ea (that is, it looks like a floating-point constant but does not have a sign following the "e").
- Multibyte characters in source code files are not processed correctly, regardless of the setting of the locale. This is because the compiler is linked `-non_shared` against `libc` on a Red Hat 5.2 system to avoid binary incompatibilities in `glibc` shared objects between `glibc-2.0.7` (on RH5.2), `glibc-2.1.1` (on RH6.0), and `libc-99.5.28` (on SuSE 6.1).

The multibyte locale support in `glibc-2.0.7` does not work. Note that applications built on systems with working multibyte locale support in `libc` are not affected. This is only a problem if those applications depend on locale-specific processing of multibyte characters at compile time.

- Because the compiler and driver executables are linked `-non_shared` against `glibc-2.0.7`, which is licensed under the LGPL, we need to provide the capability to users to whom we distribute the compiler to re-link it against modified versions of `glibc`. This capability will be provided on the same web site that distributes the compiler, accessed through the "software" link from <http://www.compaq.com/linux>.

10 gcc Compatibility

There are a number of extensions to the C language that are documented for gcc in the GCC and CPP manuals at <http://egcs.cygnus.com/onlinedocs/>.

As noted earlier, *Compaq C* implements a number of these extensions but not all of them. The major themes for *Compaq C* are performance, reliability, and standards compliance. Language extensions that are not standardized are usually only implemented to address a specific identified need. So gcc extensions that are the most likely to be implemented are those that match C99 features. Also, features that are very heavily used in applications that "ought" to be compiled by *Compaq C* for performance reasons are good candidates.

While gcc does represent a defacto standard for portability because of its availability on so many platforms, source code that stays within the bounds of the formal ANSI/ISO standard is much more portable. It is not a goal to make the *Compaq C* compiler 100% compatible with gcc, but rather to reduce the difficulty of building "fairly portable" source code that has become somewhat dependent on gcc features.

We would appreciate feedback on specific features we do not implement that you find would make a significant difference to the ease of building your code with this compiler if we did implement them.

10.1 Unsupported Extensions

The following extensions are considered unlikely to be implemented in a future version of Compaq C:

- enum types are compatible with unsigned int.

This is not an extension as much as a different implementation choice. The C standard requires each enum type to be compatible with some integer type. gcc chooses unsigned int (on both *Tru64 UNIX* and Linux), while *Compaq C* chooses signed int (on both *Tru64 UNIX* and Linux). This difference seems unlikely to present a problem large enough to warrant a change.

- Preprocessor Assertions

The #assert/#unassert directives, and the predefined assertions #system, #cpu, and #machine do not seem to be in common use in user-space source code.

- #pragma once

This is already noted in the CPP manual as obsolete, and *Compaq C* has done automatic detection of header files "guarded" by conditionals to include only once for quite some time.

- Statements and Declarations in Expressions

A compound statement enclosed in parentheses may appear as an expression in GNU C.

- Locally Declared Labels

Support for __label__

- Labels as Values

Ability to take the address of a local label using the && operator.

- Nested Functions

Ability to define a function within another function.

- Constructing Function Calls

Using any of the following:

- __builtin_apply_args
- __builtin_apply
- __builtin_return

- Naming an Expression's Type

Using the typedef <name> = <expression> syntax.

- Generalized Lvalues

Allowing the following as lvalues if their operands are lvalues: compound expressions, conditional expressions, and casts.

Note

CCC does accept a very limited subset of this capability if the K&R mode switch (-std0) is specified.

- Conditionals with Omitted Operands

Using the `?:` operator without a second operand.

- **Case Ranges**
Specifying a range of consecutive values in a single case label.
- **Cast to a Union Type**
Casting a scalar type to a union type.
- **Declaration Attributes of Functions**
Using the `__attribute__` qualifier.
- **Prototypes and Old-Style Function Definitions**
Using a function prototype declaration to override a later old-style definition.
- **Specifying Attributes of Types**
Using the `__attribute__` keyword on a type.
- **Controlling Names Used in Assembler code**
Using the `asm` keyword in a declaration.
- **Variables in Specified Registers**
Using the `asm` keyword in a declaration.
- **Incomplete enum Types**
Defining an enum tag without listing its values.
- **Built-in functions**
CCC and gcc provide a different set of built-in functions. There may or may not be a CCC equivalent for a gcc built-in. See the Programmer's Guide and the `/usr/lib/compaq/ccc-<version>/alpha-linux/include/machine/builtins.h` header file.
- **Objective C**
Compaq C tries to remain closer to standard C.
- **Casting an expression to a union**
An expression can be cast to a union if the union contains a member of the expression's type.

10.2 Supported or Partially Supported Extensions

The following extensions are supported or partially supported in the current version of *Compaq C* (There are many other minor gcc features and behaviors in common with *Compaq C* not listed here):

- **va_start**
The second operand to `va_start` can be parenthesized.
- **#include_next**
This is very useful in adapting to various irregularities in system-supplied header files without having to provide or generate entirely new versions to make them compatible with the compiler.
- **Constructor Expressions**
Using the construct `(type-name){initializer-list}`. This is what C99 refers to as a *compound literal*.
- **Referring to a Type with `typeof`**

CCC only supports the `__typeof__` spelling of this operator. It does not support the `typeof` spelling.

- Arrays of Length Zero

When arrays of zero length appear in a `struct` or union or parameter declaration, CCC treats them the same way gcc does.

When arrays of zero length appear elsewhere, CCC issues a diagnostic and treats the array as if it were declared with one element.

- Inquiring on Alignment

gcc supports the `__alignof__` operator. CCC supports a similar `__builtin_alignof` operator.

- Specifying Attributes of Variables

Although CCC does not support the `__attribute__` keyword on declarations, it does provide some similar capabilities.

The gcc aligned attribute corresponds to the `__align` storage-class modifier on CCC.

Some of the gcc `nocommon`, section attributes can be set using `#pragma extern_model` in ccc.

- Inline Functions

Both gcc and C99 define an inline (`__inline`) qualifier for functions, and allow it to be applied to functions with external linkage. Some of the details differ—in particular the conditions under which a function body with an external symbol is generated in the object module.

To get the gcc behavior (the default on Linux), use the command-line option `-accept gccinline`.

To get the C99 behavior (the default on *Tru64 UNIX* systems) use the command-line option `-accept nogccinline`.

- Assembler Instructions with C Expression Operands

The compiler provides a simple but powerful form of inline-assembler capability that resembles the gcc feature because it looks like a function call. But the detailed syntax and semantics differ substantially. The *Compaq C* form is simple and flexible to use and generally works well with the optimizer without the need for semantic annotations that are often required with the machine-description-based notation used by gcc.

See the Programmers Guide and the `/usr/lib/compaq/ccc-<version>/alpha-linux/include/c_asm.h` header file.

- Function Names as Strings

The compiler implements the C99 `__func__` predeclared variable that is equivalent to the gcc variable `__FUNCTION__`.

By default, the CCC driver provides a macro equivalence: `D__FUNCTION__ = __func__`.

- The restrict keyword

In its default mode of compilation, gcc does not recognize the new C99 keyword `restrict` (it only recognizes `__restrict`). While CCC does recognize `restrict` as a keyword in its default mode, the command-line option `-accept norestrict_keyword` will give the gcc behavior.

10.3 Likely Future Extensions

The following extensions are likely to be implemented at least partially or in a slightly different form that is consistent with C99:

- An empty set of braces {} can be used in an initializer to designate all zero values.
- The `#warning` preprocessor directive
This is very similar to a form of `#pragma message` already supported by *Compaq C*.
- Macros with Variable Number of Arguments
Using a macro that accepts a variable number of arguments, as in C99.
- Hex Floats
Using floating constants such as `0x1.fp3`, as in C99.
- Newlines within string literals
This extension seems to invite undetected typos that gobble up source code unintentionally. And the same functionality of spreading a long string literal over multiple source lines is given by two different "safer" mechanisms that are also portable in standard C: adjacent string concatenation and trailing-backslash line splicing. The latter was even portable in K&R C within a string literal. But for some reason, this feature continues to be used in a number of source packages.
- The Character `esc` in constants
Using the `'\e'` escape sequence. This was not incorporated into C99. It might be supported under an option if its absence proves to be a problem.
- Complex Numbers
C99 defines builtin types for complex numbers that are somewhat different from the current gcc implementation. We will be providing the C99 version, but not a gcc-compatible variation.

11 Downloading Updates and Learning about Related Products

For information about downloading updates to *Compaq C*, *Compaq Ladebug*, *CPML*, and related products for Linux *Alpha* systems, please see the following Web page:

<http://www.compaq.com/linux/>

For information about other third-party products (including web browsers) for the Linux *Alpha* platform, please see the following Web page:

<http://www.alphalinux.org/>

12 Technical Support

Technical support for this software is available only through the Peer to Peer Support program hosted at the University of New Hampshire:

<http://www.iol.unh.edu/linux/alpha/development/>