

Tru64 UNIX

Using the Compaq C Compiler

October 2000

Contents

Using the Compaq C Compiler

Is This Best Practice Right for You?	1
Before You Begin	1
Applying the Best Practice	2
Improving Run-Time Performance	3
Tuning Applications for Specific Alpha Hardware Platforms	4
Compiling for a Range of Tru64 UNIX Releases	5
Porting to the 64-bit Alpha Platform	6
Controlling Compiler Diagnostic Information	6
Resolving "Unaligned Access" Errors	8
Fixing Errors Detected by New Versions of Software	9
Creating a Test Case to Demonstrate a Potential Compiler Problem	9
Comments and Questions	11
Legal Notice	11

Using the Compaq C Compiler

This Best Practice describes how to use the Compaq C compiler and other Tru64 UNIX programming tools to solve some common compilation and run-time problems. It addresses a wide range of issues — from how to tune an application for faster performance to how to create test cases for use in analyzing compiler problems.

See the Tru64 UNIX Best Practices Web page for more information about Best Practices documentation.

Is This Best Practice Right for You?

Not all Best Practices apply to all configurations, so you must be sure that it is appropriate for your system and circumstances. To use this Best Practice, you must meet the requirements described in the following table:

System Component	Requirement
Operating System	Tru64 UNIX Version 4.0D or higher. To determine the version number of the operating system, use either of the following commands: <code>more /etc/motd</code> or <code>sizer -v</code>
Hardware Configuration	An Alpha™ system. For tuning purposes, use the <code>/usr/sbin/psrinfo -v</code> command to determine the chip number of the Alpha processor in your system.

Before You Begin

Before you apply the Best Practice for using the Compaq C compiler, you must understand some background information.

- You should understand the compilation process and be familiar with the `cc(1)` reference page — in particular with its descriptions of the options that are recommended in this Best Practice.
- You should also be familiar with the default options that are supported by the `cc` command. Two of the most important default options were chosen to compile the widest possible range of programs, while maximizing performance:

The default language mode for `cc` is `-std0` (K&R C) for Tru64 UNIX Version 4. The mode is `-std` (relaxed ANSI C) for both Tru64 UNIX Version 5 and the DTK Supplement compiler. This user-requested change to `-std` improves Compaq C's default language compatibility with other popular compilers and makes it easier to port applications to Tru64 UNIX systems.

- The default optimization level for `cc` is `-O2`. It was chosen to deliver good performance (measuring both run time and code size), while continuing to make illegal C programs behave as users expect them to behave.

You can override all `cc` option defaults by adding command-line options directly to makefiles and compilation scripts. You can override the defaults indirectly by using either of the following mechanisms:

- A per-user `DEC_CC` environment variable
- A system-wide `comp.config` file

These techniques are described in detail in `cc(1)`.

Applying the Best Practice

Before you use any of the compiler options that are described in this Best Practice, be sure to follow the recommendations in Before You Begin.

The following sections describe the compilation objectives of this Best Practice and the compilation option settings that relate to them:

- Tuning applications for faster performance
- Tuning applications for specific versions of the Alpha chip set (for example, chip number 21064, 21164, 21264, and so forth)
- Building applications to be run on multiple Tru64 UNIX releases (for example, Version 4.0A-G, Version 5.0, and so forth)
- Looking for coding problems in applications that are being ported to the Alpha 64-bit programming environment
- Controlling the level of diagnostic information that is provided by the compiler
- Resolving “unaligned access” run-time errors
- Determining why a working program does not compile or run properly when compiled with a newer release of the compiler
- Creating self-contained test cases for use in analyzing possible compiler problems

Improving Run-Time Performance

The optimization techniques that are described in this section can dramatically improve application performance, often with little effort. We recommend that you explore these techniques for any application for which performance is important. As you begin, consider the following:

- Can you measure the performance of your application?
- Can you verify the correctness of your application?

Performance measurement can assure you that your optimization efforts are worthwhile. Correctness verification (for example, regression testing) can assure you that the optimizers have not exposed latent bugs in your application code.

We recommend the following optimization techniques (and suggest that you experiment with them in the order given):

- Specify a higher optimization level (`-On`) on the `cc` command line. `-O1` is the default. `-O2` (same as `-O`) is almost always beneficial and rarely exposes latent source code bugs. `-O3` and `-O4` are more aggressive (and beneficial), but these are inappropriate for code that is being built into a shared library. See `cc(1)` for details about the specific transformations that are enabled at each optimization level.
Increasing the compiler optimization level is usually the easiest, safest, and most beneficial way to improve run-time performance.
- Compile multiple files together on the same `cc` command line and specify the `-ifo` (interfile optimization) option. Doing so allows the compiler to optimize across file boundaries.
- Specify the `-arch` or `-tune` options to tune the application for a specific Alpha chip or chip set. (The `-arch` and `-tune` options are described in the next section.)
- Specify the `-spike` or `-om` options to enable the spike or om post-compilation optimizers.

The `-spike` option is generally preferred for Tru64 UNIX Version 4.0F and higher. It is shipped with the Tru64 UNIX Version 5.1 operating system and, if necessary, can be downloaded from the Developers' Toolkit Supplement Web site.

The spike post-link optimizer performs code optimizations that are associated with code layout, unreachable code, and address computations. You can access spike by using either the `-spike` option

or the `spike` command. See the *Programmer's Guide*, `cc(1)`, and `spike(1)` for details.

- Specify the `-fast` option.

The `-fast` option enables a set of optimization options that provide maximum run-time performance benefits with minimal risk to most programs that do not require extreme floating-point precision. Many users only need to specify the `-fast` option to achieve their performance objectives. See `cc(1)` for a complete list of the options that `-fast` enables.

If necessary, you can selectively override any of the options that are associated with `-fast`. For example, it is not uncommon for users to disable the ANSI pointer aliasing rules using `-fast -noansi_alias`.

- Explore the use of various feedback methods to improve performance. You can collect feedback data by using any one of the available profiling methods. Then, you can feed the collected data into the compiler, `om`, or `spike`. See the *Programmer's Guide*, `cc(1)`, and `spike(1)` for details.

Tuning Applications for Specific Alpha Hardware Platforms

All versions of the Alpha chip support the same core instruction set. Newer versions of the Alpha chip implement extensions to the core instruction set that can dramatically improve performance. These instruction extensions will execute on all Alpha platforms:

- Instruction extensions that are added to one version of the Alpha chip are always retained in later versions.
- On older platforms, the instruction extensions are emulated by the operating system instead of being directly executed in hardware.

By using the `-arch` and `-tune` options on the `cc` command, you can produce object code that includes the instruction extensions for a particular chip or chip set:

- If the program is to be run on a specific version of the Alpha processor and no earlier versions (for example, `ev5` and later systems), specifying `-arch` (with the specific processor type) can produce dramatic performance improvements on that hardware.

Note

Running a program on an earlier Alpha processor might incur a severe performance penalty because any instruction

extensions are emulated by an instruction emulator in the operating system kernel.

The default setting for `-arch` is `-arch generic`, which generates code that can run on all versions of the Alpha processor.

- If you are planning to run the program on all versions of the Alpha chip (including those with no instruction set extensions), use `-tune` with the type of the processor on which it is most essential to achieve optimum performance.

Unlike `-arch`, the `-tune` option does not rely on operating system emulation to implement instruction set extensions on early Alpha hardware. Instead, the compiler duplicates certain segments of code. One duplicated segment will contain instructions that are supported only by the type of processor that you specify, and the other duplicated segment will contain instructions that any Alpha processor can execute. The duplicated segments are surrounded by a hardware test that selects the appropriate segment at execution time.

The `-tune` option may slightly increase object size because of the duplicated code segments. It may also cause a slight performance degradation on older hardware (ev4 and earlier). However, it can often produce significant performance gains on newer hardware.

To specify either of these options, you need to know which type of chip is in your Alpha system. You can use the following command to determine the chip number: `/usr/sbin/psrinfo -v`

See `cc(1)` for details about the `-arch` and `-tune` options.

Compiling for a Range of Tru64 UNIX Releases

Tru64 UNIX operating systems support forward compatibility across releases but not backward compatibility. Programs may not run correctly on systems that are running earlier versions of Tru64 UNIX than the version on the system on which the programs were built.

For portability across OS releases, compile applications on systems that are running the earliest version of Tru64 UNIX on which you expect the applications to be run.

To determine which version of Tru64 UNIX is installed on a system, you can use either of the following commands: `more /etc/motd` or `sizer -v`

Porting to the 64-bit Alpha Platform

The most common challenge in porting programs to Tru64 UNIX is application code that assumes that `int` and pointer variables are the same size. On Tru64 UNIX systems, an `int` is 32 bits and a pointer is 64 bits. The following option selections may help you detect and resolve this potential problem:

- Specify the `-msg_enable 64bit,portable` option to enable compiler diagnostic messages about coding practices that may have unintended effects on systems with 64-bit addressing. This option also enables diagnostics about coding constructs that are not generally portable. This type of coding problem can often be resolved with only minor coding changes. (You can also use the `#pragma message` directive to enable these diagnostics. See the *Programmer's Guide* for details.)
- Specify the `-taso`, `-xtaso`, or `-xtaso_short` options if you want to retain 32-bit addressing in a 64-bit environment. These options are useful under the following circumstances:
 - Your application assumes assignment compatibility between pointer and `int` variables.
 - Your application does not need the larger 64-bit address space.
 - You need to reduce the size of your program's object file. (32-bit pointers use less memory.)
 - You are developing a program whose in-memory structures are accessed by both 32-bit and 64-bit systems.

If you establish 32-bit pointers in your compilation, you should run the `protect-headers_setup` script to prevent interface problems with the Tru64 UNIX system libraries. See the *Programmer's Guide* and `protect_headers_setup(8)` for more information.

You can also use the Compaq Porting Assistant to detect Tru64 UNIX porting problems. See the Compaq Porting Assistant Web site for details and a free download.

Controlling Compiler Diagnostic Information

The compiler diagnostic facility is highly customizable. Users can control which specific diagnostic messages they want the compiler to issue, the severity-level limit of the messages they want to see, whether they want to see a specific message once per compilation or once per offense, how many total messages they want to see, and even how the compiler should respond to certain offenses.

The compiler diagnostic facility is controlled by both command-line options and `#pragma` directives. The command-line controls are:

```
-msg_enable  
-msg_disable  
-msg_inform  
-msg_warn  
-msg_error  
-msg_fatal  
-msg_once  
-msg_always
```

These same controls are also available in the `#pragma` message directive.

These controls are typically used as follows:

- To increase or decrease the number of diagnostic messages that the compiler issues, use `-msg_enable leveln`, where $n = 1-6$. (Note that `-check` is equivalent to `-msg_enable level5`.)
- To increase the amount of information that the compiler generates for each diagnostic, add `-verbose` to the command line. The verbose display includes a recommended user action for each diagnostic.
- To request diagnostic information for specific interest areas, use `-msg_enable msg-group`. The message groups are listed in `cc(1)` and include: `64bit`, `alignment`, `c_to_cxx`, `noansi`, `obsolescent`, `overflow`, `performance`, `preprocessor`, and more. For example, `-msg_enable defunct` enables messages that report the use of obsolete features. (Note that `-portable` is equivalent to `-msg_enable portable`.)
- To request a specific message in the output, use `-msg_enable msg-id`. Message IDs are given in each diagnostic message. You can use the `-msg_dump` option to display the full set of message IDs that can be issued for a given compilation.
- To disable a specific message, use `-msg_disable msg-id`. Informational and warning messages can also be disabled by group or level. Error-level and fatal-level diagnostics cannot be disabled.

Everything that you can do to control message output from the command line can also be done directly in your source code with the `#pragma` message directive. See the *Programmer's Guide* for details about the `#pragma` message directive, and see `cc(1)` for details about the command-line options.

Resolving “Unaligned Access” Errors

"Unaligned access" error messages appear in your output at run time. These messages do not necessarily indicate incorrect results, but unaligned access does cause additional processing and slows down program execution. We recommend that you fix unaligned accesses if you can. Fixing this type of error typically involves the following steps:

- Run the debugger, specifying the executable file with the unaligned access.
- In the debugger, set a break point (`stopi`) at the pc address that is specified in the “unaligned access” error message and then issue the `run` command.
- When the debugger stops at the break point, use the `where` command to find the location of the alignment error. Then, modify the source code, if possible, to eliminate the error.

By default, the compiler aligns each data item so that its starting address is an even multiple of the size of the data type that was used to declare it, that is, on its natural boundary. Misalignment can occur if a pointer variable is type cast from one data type to a larger data type or if packed structures are created by means of the `#pragma pack` directive.

If you cannot change the source code without creating other problems, specifying additional compiler options may resolve the alignment problem. Compiler options that relate to resolving this type of problem include `-xtaso`, `-nomember_alignment`, and `-assume noaligned_objects` (or `-misalign`).

The `-assume noaligned_objects` option (or `-misalign` option) causes the compiler to fix the problem by adding instructions to the object file. As a result, unaligned access messages do not appear at run time. If these options are not specified, the operating system adds instructions at run time to fix the problem and then issues the error message. Both of these cases (that is, compiler fix and operating system fix) result in slower run-time performance, particularly in the case of the operating system fix, so we recommend that you eliminate unaligned accesses whenever possible.

Use the `uac` command if you are not concerned with unaligned access messages and do not want them to appear in the output. See `uac(1)` for details about the `uac` command.

Fixing Errors Detected by New Versions of Software

When changing to a new version of the compiler, you may encounter compile-time or run-time problems that were not detected by older versions of the compiler. Newer compilers tend to have stricter diagnostics and more aggressive optimizations. More aggressive optimizations sometimes expose coding bugs that were not evident under less aggressive optimizations.

Use one or more of the following techniques to fix a problem with a working program that does not compile or run properly when compiled with a newer version of the compiler:

- Increase the amount of diagnostic information that the compiler produces by increasing the value of the “level” parameter for the `-msg_enable` option, for example, `-msg_enable level15` (or `-check`).
- Make sure that you have investigated all “uninitialized variable” messages (and any other messages) before attempting any other analysis.
- Compile your program with the `-g` option (which produces symbol table information for debugging and suppresses optimizations) and debug the program. If `-g` causes the problem to disappear, compile with the `-g3` option (which produces symbol table information for fully optimized code) and then debug the program.
- Specify the `-check_bounds` option to check for addressing problems in arrays at run time.
- Use the Third Degree utility to analyze the program for heap memory leaks, invalid address references, and uninitialized memory access.
- Incrementally decrease the `-O` level and compare the results. Instead of just repeating this process until the error goes away, we recommend that you attempt to debug the program at each optimization level that produces the error.

Creating a Test Case to Demonstrate a Potential Compiler Problem

When the results of a compilation indicate a potential compiler problem, you need to create a file that reproduces the problem and then send the file to a Compaq support person for analysis.

In some cases, you will not be able to create a simple test case that reproduces the problem. In such instances, you may need to create a preprocessed version of the file.

Creating a preprocessed version of the file is typically most useful when the source code contains “private” header files or complex compile commands. The objective is to capture the test case in a single file and remove the dependencies on private header files or complex compile commands. You can do this by using the `-P` option, which creates a `filename.i` file. The `.i` file provides a self-contained and environment-independent test case that reproduces the problem.

You can specify the `-P` option in the following ways:

- If the application is built by a `cc` command line, add the `-P` option to the command line.
- If the application is built by a complicated makefile or script, establish the `-P` by assigning it to the `DEC_CC` environment variable (`setenv DEC_CC "-P"`). Then, the compiler will automatically produce a `.i` file and you do not need to change the makefile or script.

The text that follows contains an example of using the `-P` option to create a standalone reproducer for a compiler problem. In this example, the problem is a compiler crash resulting from a memory access violation:

```
% cc -std1 -O2 ... -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF ...
-I../include/alpha -I../include/misc ... mysrc.c
cc: Fatal: A memory access violation (bus error or segmentation fault)
has occurred. Please submit a problem report.
```

The `mysrc.c` file in this example includes private header files that would need to be supplied and then built using the exact build commands that caused the problem. Sometimes header files have many dependencies or the build environment may be difficult to recreate, so this can be a difficult task.

Using the `-P` option simplifies the process:

```
% setenv DEC_CC "-P -v"
% cc -std -O2 ... -c -DBSD44 -DMSG -DNLS -DMACH -DCMU -DOSF ...
-I../include/alpha -I../include/misc ... mysrc.c
```

In this case, the fatal error does not occur because the compiler preprocesses the file instead of compiling it. This operation also produces a corresponding `mysrc.i` file:

```
% ls mysrc.*
mysrc.c mysrc.i
```

The preprocessed `mysrc.i` file contains code that is included from the header files in the build tree, so it can now be compiled outside of your build environment. To verify this fact and the fact that the reproducer still demonstrates the problem, issue the following `unsetenv` command (if necessary), strip the `-D`, `-U`, and `-I` options from the original command

line — they are no longer necessary because the file has already been preprocessed — and then try to compile the `mysrc.i` file:

```
% unsetenv DEC_CC
% cc -std -O2 ... -c ... mysrc.i
cc: Fatal: A memory access violation (bus error or segmentation fault)
has occurred. Please submit a problem report.
```

These results indicate that the `mysrc.i` file is a valid standalone reproducer. With this file and the necessary compile options, a Compaq support person should be able to reproduce the problem outside of your environment.

The preprocessed files may contain large amounts of whitespace and extraneous source code lines. You can remove some of the whitespace by using the following commands:

```
% cat -r mysrc.i | cb > mysrc_i.c
```

See `cat(1)` and `cb(1)` for details on these commands.

Comments and Questions

We value your comments and questions on the information in this document. Please mail your comments to us at this address:

`best_practices@zk3.dec.com`

Legal Notice

COMPAQ and the Compaq logo are registered in the U.S. Patent and Trademark Office.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendors standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this publication is subject to change without notice and is provided "as is" without warranty of any kind. The entire risk arising out of the use of this information remains with recipient. In no event shall Compaq be liable for any direct, consequential, incidental, special, punitive, or other damages whatsoever (including without limitation, damages for loss of business profits, business interruption or loss of business information), even if Compaq has been advised of the possibility

of such damages. The foregoing shall apply regardless of the negligence or other fault of either party and regardless of whether such liability sounds in contract, negligence, tort, or any other theory of legal liability, and notwithstanding any failure of essential purpose of any limited remedy.

The limited warranties for Compaq products are exclusively set forth in the documentation accompanying such products. Nothing herein should be construed as constituting a further or additional warranty.