Concurrent C/C++

Version 5.3 Release Notes (Linux)

July 2001

0898497-5.3

READ ME BEFORE
INSTALLING
THIS PRODUCT

CONCURRENT
COMPUTER
CORPORATION

Contents

1.0. Introduction

Concurrent C/C++ is part of the PowerWorks™ Linux Development Environment (PLDE) and utilizes Edison Design Group's C++ front end and Concurrent's Common Code Generator (CCG) technology to produce highly optimized object code tailored to Concurrent systems running PowerMAX OS™.

There are several compiler switches that provide a degree of compatibility with previous drafts of the ANSI C++ Standard, USL versions 2.1 and 3.0 C++ compilers (also known as **`cfront`**), Kernighan and Ritchie C and SVR4 C, in addition to compatibility with the ANSI C++ and C languages.

This release combines into a single compiler what were, previous to release 5.1, separate C and C++ compilers. Release 5.1 was the "next" release after Concurrent C Compiler 4.3 and Concurrent C++ Compiler 3.1. This release does not require un-installing previous versions of the C and C++ compilers. It installs in its own unique location and provides a mechanism for supporting the use of multiple releases.

This release provides a command-line-based program development environment for building complex projects. Alternatively, the user may use NightBench™ to interface with this program development environment.

As of release 5.1, C/C++ no longer ships with USL iostream and complex libraries. They are available separately.

2.0. Documentation

Table 2-1 lists the Concurrent C/C++ 5.3 documentation available from Concurrent.

Table 2-1. Concurrent C/C++ Version 5.3 Documentation

Manual Name	Pub. Number
Concurrent C/C++ Reference Manual	0890497-030
Concurrent C/C++ Version 5.3 Release Notes (Linux)	0898497-5.3

Copies of the Concurrent documentation can be ordered by contacting the Concurrent Software Support Center. The toll-free number for calls within the continental United States is 1-800-245-6453. For calls outside the continental United States, the number is 1-954-283-1822 or 1-305-931-2408.

Additionally, the manuals listed above are available:

- online using the PowerWorks Linux Development Environment utility, **`nhelp`**
- in PDF format in the **`documentation`** directory of the PLDE Installation CD
- on the Concurrent Computer Corporation web site at www.ccur.com

3.0. Prerequisites

Prerequisites for Concurrent C/C++ Version 5.3 for both the host system and target system are as follows:

3.1. Host System

3.1.1. Software

- Red Hat® Linux*
- Required capabilities

NOTE

The following capabilities are normally installed as part of the standard installation of Red Hat Linux and the PowerWorks Linux Development Environment. During installation of the PLDE, the user will be notified if required capabilities do not exist on the Linux system.

- PowerWorks Linux Development Environment

Capabilities	RPMs providing these capabilities

`plde-HyperHelp`
`plde-HyperHelp-scripts`
`plde-nhelp`
`plde-pmax-crossdev`
	`plde-x11progs-6.4.2-003`
	`plde-HyperHelp-scripts-6.4.2-002`
	
	any or all of the following:
	`plde-pmax-crossdev-4.3-P7-1`
	`plde-pmax-crossdev-5.0-SR1-4`
	`plde-pmax-crossdev-5.1-SR0-1`

User applications built with Concurrent C/C++ may require other capabilities which are provided by additional RPMs included on the PLDE Installation CD. Refer to the section titled “Cross-Development Libraries and Headers” in the *PowerWorks Linux Development Environment Release Notes* (0898000) for more information.

* This product has been extensively tested on Red Hat Linux 6.1 and 6.2. However, this product has not been tested with versions of Linux supplied by other vendors.

- Red Hat Linux

Capabilities	**RPMs providing these capabilities**

`/bin/sh`
`ld-linux.so.2`
`libc.so.6`
`libc.so.6(GLIBC_2.0)`
`libc.so.6(GLIBC_2.1)`
`libnsl.so.1`
`rpm >= 3.0.3`
	Red Hat 6.1:
	`bash-1.14.7-16`
	`glibc-2.1.2-11`
	`rpm-3.0.3-2`
	
	Red Hat 6.2:
	`bash-1.14.7-22`
	`glibc-2.1.3-15`
	`rpm-3.0.4-0.48`

3.1.2. Hardware

- an Intel®-based PC - 300Mhz or higher (recommended minimum configuration)
- 64MB physical memory (recommended minimum configuration)

3.2. Target System

3.2.1. Software

- PowerMAX OS 4.3 or later

3.2.2. Hardware

- Computer Systems:

  Power Hawk™ 620 and 640

  Power Hawk 710, 720 and 740

  PowerStack™ II and III

  Night Hawk® Series 6000

  TurboHawk™

  PowerMAXION™

- Board-Level Products:

  Motorola® MVME2604

  Motorola MVME4604

4.0. System Installation

Installation of the host portion of Concurrent C/C++ is normally done as part of the general installation of the PowerWorks Linux Development Environment software suite. A single command installs (or uninstalls) all software components of the PLDE, as described in the *PowerWorks Linux Development Environment Release Notes* (0898000).

The following section describes how to install (or uninstall) Concurrent C/C++ separately from the PLDE suite for those rare cases when this is necessary.

In addition, it is necessary to install the PowerMAX OS cross-development libraries on your Linux system in order to cross-compile and cross-link on that system.

4.1. Separate Host Installation

In rare cases, it may be necessary to install (or uninstall) Concurrent C/C++ independent of the installation of the PowerWorks Linux Development Environment software suite. This may be done using the standard Linux product installation mechanism, **`rpm`** (see **`rpm(8)`**).

The names of the RPMs associated with Concurrent C/C++ 5.3 are:

```
plde-c++-5.3
plde-c++invoker
plde-c++help-5.3
```

and the files associated with these RPMs, respectively, are:

```
plde-c++-5.3-001-1.i386.rpm
plde-c++invoker-5.3-001.i386.rpm
plde-c++help-5.3-001-1.i386.rpm
```

which can be found in the **`linux-i386`** directory on the PowerWorks Linux Development Environment Installation CD.

NOTE

The package **`plde-c++help-5.3`** contains the online manuals for the C/C++ compiler. The installation of **`plde-c++help-5.3`** is not necessary for the proper operation of the compiler.

NOTE

The user must be root in order to use the **`rpm`** product installation mechanism on the Linux system.

To install the Concurrent C/C++ RPMs, issue the following commands on your Linux system:

1. Insert the PowerWorks Linux Development Environment Installation CD in the CD-ROM drive

2. Mount the CD-ROM drive (assuming the standard mount entry for the CD-ROM device exists in **/etc/fstab**)

   ```
   mount /mnt/cdrom
   ```

3. Change the current working directory to the directory containing the Concurrent C/C++ RPMs

   ```
   cd /mnt/cdrom/linux-i386
   ```

4. Install the RPMs

   ```
   rpm -i plde-c++invoker-5.3-001.i386.rpm \
          plde-c++-5.3-001-1.i386.rpm \
          plde-c++help-5.3-001-1.i386.rpm
   ```

   By default, the product is installed in **/usr/opt**. To install in a different directory, issue the following commands instead:

   ```
   rpm -i plde-c++invoker-5.3-001.i386.rpm
   rpm -i --relocate /usr/opt=directory \
          plde-c++-5.3-001-1.i386.rpm \
          plde-c++help-5.3-001-1.i386.rpm
   ```

   where *directory* is the desired directory.

5. Change the current working directory outside the **/mnt/cdrom** hierarchy

   ```
   cd /
   ```

6. Unmount the CD-ROM drive (otherwise, you will be unable to remove the PowerWorks Linux Development Environment Installation CD from the CD-ROM drive)

   ```
   umount /mnt/cdrom
   ```

NOTE

The optional **plde-c++cfront** package containing USL iostream and complex libraries should be installed after the **plde-c++** package.

To uninstall the Concurrent C/C++ RPMs, use the following command:

```
rpm -e plde-c++help-5.3 \
       plde-c++invoker \
       plde-c++-5.3
```

4.2. Cross-Development Libraries

In order to cross-compile and cross-link on the Linux system, it is necessary to have certain PowerMAX OS libraries installed on that system.

The names of the RPMs containing the PowerMAX OS libraries minimally required for cross-linking are:

```
plde-pmax-crossdev-4.3
plde-pmax-crossdev-5.0
plde-pmax-crossdev-5.1
```

and are used when linking for a PowerMAX OS 4.3, PowerMAX OS 5.0, and PowerMAX OS 5.1 target system, respectively. The files associated with these RPMs are:

```
plde-pmax-crossdev-4.3-P7-1.i386.rpm
plde-pmax-crossdev-5.0-SR1-4.i386.rpm
plde-pmax-crossdev-5.1-SR0-1.i386.rpm
```

NOTE

The version number is part of the name of the RPM. Because of that, it is possible to install both RPMs on the Linux system at the same time. This allows the user to generate executables for multiple PowerMAX OS versions from the same Linux system.

User applications built with Concurrent C/C++ may require other capabilities which are provided by additional RPMs included on the PLDE Installation CD. Refer to the section titled “Cross-Development Libraries and Headers” in the *PowerWorks Linux Development Environment Release Notes* (0898000) for more information.

5.0. Using Concurrent C/C++ with the PLDE

The following should be taken into consideration in order to use Concurrent C/C++ with the PowerWorks Linux Development Environment.

5.1. PATH Considerations

On Linux systems, the **cc**, **c++**, **gcc**, and **g++** commands invoke the native Linux compilers, which are completely unrelated (and incompatible at the object level) with PowerMAX OS and the Concurrent C/C++ cross-compiler.

To utilize the Concurrent C/C++ compiler, specify the following in your `PATH` environment variable:

```
PATH=$PATH:/usr/ccs/bin
```

The compiler should then be invoked with either **ec** or **ec++**.

However, if you wish to be able to invoke the Concurrent C/C++ compiler as **cc** or **c++**, insert the following at the head of your `PATH` environment variable:

```
PATH=/usr/ccs/crossbin:$PATH
```

The **/usr/ccs/crossbin** directory contains commands named **cc** and **c++** which invoke the Concurrent C/C++ compiler as opposed to the Linux compilers.

See “Makefile Considerations” on page 10 below for more information.

5.2. Include Files and Libraries

By default, the Concurrent C/C++ compiler automatically looks for PowerMAX OS include files and libraries in the tree rooted as:

/pmax/os/*version***/***arch*

where *version* and *arch* indicate the PowerMAX OS version and target architecture of your choice (see “OS Versions and Target Architectures” on page 9 for more details).

Files located under **/usr/include** and **/usr/lib** are native Linux files and are unrelated and incompatible with the corresponding files for PowerMAX OS. Do not attempt to utilize files from those directories when building PowerMAX OS programs.

Remove any explicit references to these directories in:

- source files (e.g. `#include "/usr/include/unistd.h"`)
- Makefiles (e.g. **cc -I/usr/include**)
- build scripts

Include file references of the form:

```
#include <unistd.h>
```

or

```
#include "unistd.h"
```

need not be changed. These forms are supported, as the appropriate **`/pmax/os/`***version***`/`***arch* trees are searched.

5.3. OS Versions and Target Architectures

The PowerWorks Linux Development Environment supports building PowerMAX OS programs for various versions of PowerMAX OS and various systems.

The current versions of PowerMAX OS (**`osversion`**) that are supported are:

- **`4.3`**
- **`5.0`**
- **`5.1`**

The current architectures (**`arch`**) that are supported are:

- **`nh`**
- **`moto`**
- **`synergy`**

which correspond to the following systems:

System type	*architecture*
PowerMAXION-4	**`nh`**
PowerMAXION	**`nh`**
Night Hawk 6800	**`nh`**
Night Hawk 6800 Plus	**`nh`**
TurboHawk	**`nh`**
Power Hawk 610	**`moto`**
Power Hawk 620	**`moto`**
Power Hawk 640	**`moto`**
PowerStack	**`moto`**
PowerStack II	**`moto`**
Power Hawk 710	**`synergy`**
Power Hawk 720	**`synergy`**
Power Hawk 740	**`synergy`**

NOTE

The default OS version is currently **`4.3`** and the default target architecture is **`nh`**.

You can change the **`osversion`** and **`arch`** settings in several ways:

- Specify the options on the **`ec`** or **`ec++`** command line:

  `ec -o main main.c --arch=`*arch* **`--osversion=`***os*

- Change the default for your user on a specific Linux system using the Concurrent C/C++ command line utility **`c.release`**:

  `c.release -arch` *arch* **`-osversion`** *os*

- When using the Concurrent C/C++ PDE utilities (**`c.build`**, etc.), you can:
  - Set the **`arch`** and **`osversion`** for an environment using **`c.mkenv`**:

    `c.mkenv -arch` *arch* **`-osversion`** *os*

  - Set the **`arch`** and **`osversion`** for a specific partition using **`c.partition`**:

    `c.partition -oset "--arch=`*arch* **`--osversion=`***os***`" main`**

5.4. Shared vs. Static Linking

By default, the Concurrent C/C++ compiler links with shared libraries.

Thus, if you attempt to execute your C++ program on a PowerMAX OS system it will require, at a minimum, the shared library **`libCruntime.so`**.

If your PowerMAX OS system doesn't have either the Concurrent C/C++ product or the **`c++runtime`** package installed, your program will fail to execute.

You can install the full PowerMAX OS version of the Concurrent C/C++ compiler, install just the **`c++runtime`** package, or relink your program using static libraries.

The PowerMAX OS **`c++runtime`** package is included on the PowerWorks Linux Development Environment Installation CD. See the section titled “Target Installation” in the *PowerWorks Linux Development Environment Release Notes* (0898000) for installation instructions.

To link your program using static libraries, append the **`-Zlink=static`** option to your command line:

```
ec++ -o main main.c -Zlink=static
```

5.5. Makefile Considerations

Makefiles may already contain references to **`cc`** or **`c++`** commands explicitly within them. Additionally, if default rules for compilation, such as

```
.c.o:
```

or

```
.cc.o:
```

are not explicitly mentioned, the **`make`** processor will also attempt to invoke **`cc`**, **`c++`**, or even **`g++`**.

By default, unless you have **/usr/ccs/crossbin** early in your `PATH` variable, these situations will result in the Linux native compilers being invoked instead of the Concurrent C/C++ compiler.

To resolve these problems you can take any of the following approaches.

5.5.1. Explicit Modification Using ec/ec++

Ensure that **/usr/ccs/bin** is in your `PATH` environment variable.

Modify all occurrences of **cc** and **c++** to utilize **ec** and **ec++**, respectively.

Supply default `.c.o` rules (and the like) to explicitly utilize the **ec** and **ec++** commands.

5.5.2. Use of /usr/ccs/crossbin in PATH Environment Variable

Put **/usr/ccs/crossbin** at the head of your `PATH` environment variable.

This will cause references to **cc** and **c++** to invoke the Concurrent C/C++ compiler as opposed to the Linux compilers.

5.5.3. Use of CC Environment or Make Variables

If you don't want **/usr/ccs/crossbin** early on your `PATH` (perhaps because you plan to build for Linux and/or PowerMAX OS at various times), then you'll want to just use the **ec** and **ec++** when you want to compile for PowerMAX OS (it is still necessary to add **/usr/ccs/bin** to your `PATH`).

One approach to using **ec** and **ec++** that requires minimal changes to Makefiles, etc., is to use environment variables or **make** variables to control which C/C++ compiler you're using. The following commands will all build using the PLDE cross-compilers:

Short-lived environment variables:

```
# CC=ec CXX=ec++ make arguments
```

make variables:

```
# make arguments CC=ec CXX=ec++
```

Long-lived environment variables:

```
# export CC=ec
# export CXX=ec++
# make arguments
```

You can also use the long-lived environment variable approach if you intend to always build for PowerMAX OS, by adding the following to your login script (e.g. **.profile** or **.login** depending on your shell):

```
export CC=ec
export CXX=ec++
```

Or, if you prefer finer-grained control, you can add lines like the following to the top of any Makefiles that should use the Concurrent C/C++ cross-compiler:

```
CC=ec
CXX=ec++
```

The changes will then only affect the modified Makefiles. Note that this solution only works for Makefiles that use the default `.c.o` and `.cpp.o`, etc. rules. If they contain hard-coded references to **cc** or **cc++**, then either **/usr/ccs/crossbin** must be used, or the Makefiles must be changed to use `$(CC)` and `$(CXX)` instead. If the Makefile references anything like **g++** (Linux's GNU C++ compiler), then it will need to be changed, regardless.

Here are two more complete and robust sets of variables which will work equally well with well-written Makefiles.

```
CC=/usr/ccs/crossbin/cc
CXX=/usr/ccs/crossbin/c++
AS=/usr/ccs/crossbin/as
AR=/usr/ccs/crossbin/ar
LD=/usr/ccs/crossbin/ld
```

Or, alternatively:

```
CC=/usr/ccs/bin/ec
CXX=/usr/ccs/bin/ec++
AS=/usr/ccs/bin/as.pmax
AR=/usr/ccs/bin/ar.pmax
LD=/usr/ccs/bin/ld.pmax
```

These two sets are mentioned in order to provide very easy support for those users that want to compile only for PowerMAX OS (**`/usr/ccs/crossbin`**) and for those users that may want to compile for either Linux or PowerMAX OS, depending on the application (**`/usr/ccs/bin`**).

6.0. Changes in This Release

6.1. Documentation Updates

The *Concurrent C/C++ Reference Manual* has been greatly expanded and is now available online through the **`nhelp`** and **`c.man`** tools. The traditional man pages have been abbreviated to provide only brief descriptions of the options, and the user is directed to the Reference Manual for complete documentation.

6.2. Invoking The Compiler

To use the C/C++ compiler in releases 5.1 and later, it is necessary to add the path **`/usr/ccs/bin`** to your `PATH` environment variable. The C compiler is invoked by the **`ec`** command, and the C++ compiler is invoked by the **`ec++`** command.

If an earlier C or C++ compiler is still installed on the system, the system administrator may choose to configure the previous compiler commands (**`cc`**, **`hc`**, **`cc++`**, **`c++`**, **`analyze`**, and **`report`**) to invoke the pre-5.1 release. However, by default, they will invoke the 5.3 release. The user may override this default behavior by setting the `PDE_RELEASE` environment variable to either **`pre5.1`**, **`5.2`**, or another release name, or by using the **`c.release`** command to select the user-specific default release.

6.3. The `--static_Cruntime` Link Option

The **`--static_Cruntime`** option directs the **`ec++`** driver program to use the archive rather than the shared object form of the C++ runtime library when creating a dynamically-linked program. Because the C++ runtime library is implicitly included in links, there had been no supported way to achieve this, although there were some kludge techniques that are not guaranteed to work in the future.

6.4. Function Try Blocks

Function Try Blocks are now implemented. Note, there is a known problem when inlining is disabled (such as at optimization level `NONE`). This will be fixed in a patch.

6.5. Compound Literals

Compound literals are now implemented.

6.6. The --[no]_vector_safe_prologs Compile Option

The **`--vector_safe_prologs`** compile option places a runtime test in function prologs to avoid executing AltiVec instructions in the prolog. This makes it possible for the user to write a single function that uses AltiVec instructions if available, or non-vector operations otherwise.

6.7. Various Bug Fixes

Several bugs were fixed. This list includes, but is not limited to, the following:

- Certain runtime calls failed to generate PIC code;
- Typechecking was wrong on some AltiVec intrinsic functions;
- Strict standard conforming mode permitted AltiVec enhancements;
- Debug information for PIC was incorrect;
- The **`min_align`** program did not work for predeclaring struct/union/class;
- Some intrinsics were not properly enabled for ppc750/ppc7400 targets;
- The **`-Qquick_alias`** option sometimes did not work;
- Analyze with the **`-Zlive_out`** option did not deal with exception handlers correctly;
- AltiVec intrinsics generated inefficient code for indirecting through a register.

7.0. Cautions

7.1. va_list Structure Format

When using PowerMAX OS 5.1 headers, **`varargs`** and **`stdarg`** use a different format for the `va_list` structure. Because of this, the user should avoid passing `va_list` structures to routines that are built with previous versions of the PowerMAX OS headers.

Code build with PowerMAX OS 5.1 headers can handle both the old and the new format of `va_list`. There should therefore be no problem with old code, compiled with PowerMAX OS 5.0 and earlier headers, passing `va_list` to code compiled with PowerMAX OS 5.1 headers.

For example, a dynamically-linked program that calls `vfprintf` and that is compiled under PowerMAX OS 4.3 will have no problem running on a PowerMAX OS 5.1 system, calling the `vfprintf` in the 5.1 version of **`libc.so`**. That same program compiled under PowerMAX OS 5.1 will not work if run on a PowerMAX OS 4.3 system. The `vfprintf` in the 4.3 version of **`libc.so`** will not know how to handle the new `va_list` format.

7.2. Retain Source

Users are encouraged to retain the source for their applications. Major releases may have changes in the object-file format which will require the recompilation of their programs. This release is one such release. In the process of implementing additional ANSI/ISO C++ features, some changes to “name mangling” and interfaces to runtime routines were necessary.

7.3. curses.h and bool

The **`<curses.h>`** header file contains a **`bool`** type definition. In compilation modes where **`bool`** is a keyword, this results in a compilation error. There are two workarounds. The user may turn off the **`bool`** keyword with the **`--no_bool`** option, or he may use the following sequence when using the header file:

```
#define bool _curses_bool
#include <curses.h>
#undef bool
```

When using this second workaround, if the user makes reference to curses’ `bool` type definition, the user must use the name **`_curses_bool`** instead.

7.4. Structure Compares

The C 4.3 and earlier releases supported an enhancement that allowed the user to implicitly compare a structure to zero (meaning all bytes of the structure are zero) in an if or while statement:

```
struct S {int a,b,c,d,e,f;} s;
...
if (s) {
...
```

As this enhancement is not present in any other major C compiler and is undocumented in the Concurrent C compiler, it is no longer present in C/C++ 5.1. The user can easily write a small function to test a structure for being zero should any code actually use this feature.

7.5. Known Problem with C/C++ Standard Library

The Concurrent software workaround provided below addresses a problem detected by the C++ standards committee Library Working Group. Concurrent will continue to monitor the situation and evaluate the group's final recommendation for any further action that might be necessary.

If a function is explicitly instantiated and its type is supplied, the compiler looks up all functions of that name and tries to instantiate in an attempt to determine what the closest match is. If one of these cannot be instantiated for the given type, the compiler will then issues an error.

The workaround is to omit the explicit type, but let the compiler determine it from context. An example is provided below:

```
namespace std {
 template <class Iterator> struct iterator_traits {
   typedef typename Iterator::difference_type difference_type;
 };

 template <class T> class reverse_iterator;

 template <class T>
   void operator+(typename iterator_traits<T>::difference_type,
            const reverse_iterator<T>&);

 template <class T> struct complex;

 template <class T> void operator+(const T& lhs, const complex<T>& rhs){;}

}

// Explicit instantiation which gives the error:
template void std::operator+<float>(const float&, const std::complex<float>&);

// Workaround which does not:
template void std::operator+<>(const float&, const std::complex<float>&);
```

7.6. Pragma min_align

Using a pragma `min_align` of the form:

```
#pragma min_align [struct | union | class] name align
[struct | union | class] name {
  declaration
};
```

will result in a front end assertion violation.

The workaround is to declare the `[struct | union | class]` *name* previous to the `#pragma` statement.

Specifically:

```
struct foo_tag;
#pragma min_align foo_tag 16
struct foo_tag {
   double b1;
   double b2;
   double b3;
} foo;
```

will work while:

```
#pragma min_align foo_tag 16
struct foo_tag {
   double b1;
   double b2;
   double b3;
} foo;
```

will give an assertion violation.

8.0. Direct Software Support

Software support is available from a central source. If you need assistance or information about your system, please contact the Concurrent Software Support Center at 1-800-245-6453. Our customers outside the continental United States can contact us directly at 1-954-283-1822 or 1-305-931-2408. The Software Support Center operates Monday through Friday from 8 a.m. to 7 p.m., Eastern Standard time.

Calling the Software Support Center gives you immediate access to a broad range of skilled personnel and guarantees you a prompt response from the person most qualified to assist you. If you have a question requiring on-site assistance or consultation, the Software Support Center staff will arrange for a field analyst to return your call and schedule a visit.