Cray Compiling Environment Release Overview and Installation Guide

(8.7)

S-5212

Contents

1 Cray Compiling Environment 8.7 Release Overview Introduction

This publication is an overview of the Cray Compiling Environment 8.7 release. CCE 8.7 provides Fortran, C, and C++ compilers for Cray XC systems and Cray CS systems.

This publication does **not** describe hardware, software, installation of related products, or products that Cray does not provide.

Cray Compiling Environment 8.7 Release Package Description

The Cray Compiling Environment 8.7 consists of:

- Cray Fortran Compiler, version 8.7
- Cray C and C++ Compiler, version 8.7
- CrayLibs (libraries and utilities), version 8.7
- *Cray Compiling Environment 8.7 Documentation Overview*

All software is installed by means of install tools and RPM Package Manager (RPM) files. For more detail about the Cray Compiling Environment 8.7 release package, refer to the *Cray Compiling Environment 8.7 Release Package Overview* on page 10.

2 Cray Compiling Environment 8.7 Software Enhancements

This topic describes software enhancements provided with the Cray Compiling Environment 8.7 release.

- More aggressive C++ inlining at default
  - The default inlining level for C++ is now `-hipa4`. The heuristics at `-hipa4` allow for much more inlining and faster generated code for C++ applications.
  - The Fortran and C default inlining level remains `-hipa3` (unchanged from previous releases).
  - See the `crayCC(1)` man page for more information.
- New `-hnofma` option to disable the use of fused-multiply-add instructions

  Applications sensitive to rounding differences may benefit from disabling fused multiply add (FMA) instructions. This switch is intended for application debugging purposes.

  Things to note:
  - FMAs may actually have greater accuracy (less rounding error), but may not agree with previous results.
  - FMAs generally perform better than the equivalent multiply/add instruction sequences.
  - See the `crayftn(1)`, `craycc(1)`, and `crayCC(1)` man pages for more information.
- The `-hfp3` option is now enabled when `-O3` is specified. Previously, `-O3` did not modify the `-hfp` level
  - Enabling `-hfp3` by default when `-O3` is specified pulls in a number of optimizations, including but not limited to:
    - Rewrite divides and sqrt into reciprocal sequences at all data widths
    - Aggressive floating point reassociation during optimizations
    - Optimization of EXP/LOG
- Support for C11 atomics

  Atomic operations are an optional part of the C11 standard that are now supported in CCE 8.7. They are available with the `<stdatomic.h>` header.
  - One example is that users can now do something like the following and have it reliably print 1000, because the autoincrement on x is atomic.

```
#include <stdatomic.h>
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

_Atomic (int) x;

void* foo(void *arg) {
```

```
  sleep(1);
  ++x;
}

int main() {
  pthread_t threads[1000];
  for (int i = 0; i < 1000; ++i) pthread_create(&threads[i], 0, foo, 0);
  for (int i = 0; i < 1000; ++i) pthread_join(threads[i], 0);

  printf("%d\n", x);
}
```

- Enhancements for better automatic OpenMP affinity and wait policy settings (`OMP_PROC_BIND`, `OMP_PLACES`, and `OMP_WAIT_POLICY` environment variables)
  - `OMP_PROC_BIND=AUTO` -- This Cray-specific extension, `AUTO`, is the new default value for `OMP_PROC_BIND` (the default in previous releases was `FALSE`). This causes CCE to bind threads to CPUs automatically, by partitioning the initial affinity mask of the process across threads. Binding is not done if there is only one CPU in the initial affinity mask. This policy only binds when doing so is likely to help performance. This behavior can be overridden by explicitly setting `OMP_PROC_BIND`.

    `OMP_WAIT_POLICY=AUTO` -- This Cray-specific extension, `AUTO`, is the new default value for `OMP_WAIT_POLICY` (the default in previous releases was `ACTIVE`). `AUTO` detects oversubscription and chooses the most appropriate wait policy. It behaves like `ACTIVE` if at most one thread per core is detected, and it behaves like PASSIVE if more than one thread per core is detected. `ACTIVE` is optimized for thread response time (i.e., low-overhead fork/join performance). `PASSIVE` is optimized for minimizing idle-thread resource consumption (i.e., idle threads waiting on the same core as the master thread should not degrade performance of the master thread). This behavior can be overridden by explicitly setting `OMP_WAIT_POLICY`.
  - See the `intro_openmp(7)` man page for more information.
- Initial support for ARM NEON intrinsics
  - Many commonly used ARM NEON instruction intrinsics are now supported.
- PGAS support for large memory nodes
  - PGAS optimizes on-node SMP network operations by cross mapping large regions of process virtual address space (VMA) to other on-node PEs. When the amount of available DRAM on a compute node grows beyond 1 TB along with the number of PEs growing beyond 48, PGAS runs out of VMA space for these mappings. To solve this problem, PGAS sub-divides a large memory node into two or more virtual nodes when it detects an out-of-space condition. This virtual split is transparent to the user. The only consideration a user needs to make is that `upc_cast()` will only succeed between PEs on the same SMP virtual node.
- Fortran 2018 features: `SELECT RANK`, `COSHAPE`, `GENERIC`
  - `SELECT RANK` is a language construct that allows users to reference `ASSUMED-RANK` dummy arguments. `ASSUMED-RANK` means that the called routine does not specify the array rank of the dummy argument. The rank is contained within a passed dopevector. This is like a normal select where a runtime test determines the actual rank of the argument and only executes the section of the construct that matches that type. For example:

```
Subroutine sub(array)
Integer :: array(..)     ! assumed rank syntax
Select rank (selector=>array)
RANK(0)
```

```
!  …. Code for a scalar
Rank(1)
! …. Code for a rank 1 array
! etc.
End select
```

`COSHAPE()` is a new intrinsic that returns the shape of the coarray bounds of a coarray object. For example:

```
…
Real :: coarray(10)[3,4,*]
Print *, coshape(coarray)   ! prints 3,4, and the final value determined by
the actual runtime number of images.
```

`GENERIC` is a new statement that is used to specify a generic interface without having to create an interface block. It is useful when procedure interfaces are already declared in other ways and you need to create a generic interface without duplicating those other “specific” interfaces. A call to the name of a generic interface is then resolved to the proper specific interface based on the number and attributes of the actual arguments provided.

- Performance improvements for all supported systems

3 Cray Compiling Environment 8.7 Compatibilities and Differences

There are no changes in CCE 8.7 that impact compatibility with CCE 8.6.

- OpenMP behavior change:
  - The implicit behavior for scalar variables appearing in an `omp target` construct changed in CCE 8.7, from the OpenMP 4.0 behavior to the OpenMP 4.5 behavior. In OpenMP 4.0, scalars referenced in an `omp target` construct, but not appearing in a scoping or mapping clause, are treated as though they appeared in a `map` clause. In OpenMP 4.5, such scalars are treated as though they appeared in a `firstprivate` clause. This distinction is particularly important for scalar variables that appear in a `reduction` or `lastprivate` clause on a construct closely nested in an `omp target` construct, since the implicit `firstprivate` behavior will discard the final reduction or `lastprivate` value at the end of the `target` construct. For these cases, an explicit `map` clause must be added for scalar variables appearing in a `reduction` or `lastprivate` clause. The original OpenMP 4.0 behavior can be restored by adding the `defaultmap(tofrom:scalar)` clause to the `omp target` construct.

4 Cray Compiling Environment 8.7 Documentation Overview

This topic describes the documentation that supports the Cray Compiling Environment 8.7 release.

Access Product Documentation

With each software release, Cray provides publications and man pages, and in some cases, third-party documentation. These documents are provided in the following ways:

Cray Documentation Portal	The Cray Documentation Portal is the Cray documentation delivery system. Access the HTML and PDF documentation via the Cray Documentation Portal at *https://pubs.cray.com/*. Legacy documentation (as well as man pages, and in some cases, third-party documentation) can be found on the CrayDoc website: *http://docs.cray.com* or through the local network location defined by the system administrator.
CrayPort	CrayPort is the external Cray website for registered users that offers documentation for each product. CrayPort has portal pages for each product that contains links to all of the documents that are associated to that product. CrayPort enables the quick access and search of Cray books, man pages, and in some cases, third-party documentation. Access CrayPort by using the following URL: *http://crayport.cray.com*.
Man pages	Man pages are textual help files available from the command line on Cray machines. To access man pages, enter the `man` command followed by the name of the man page. For more information about man pages, see the `man(1)` man page by entering: `% man man`
Third-party documentation	Third-party documentation that is not provided through CrayPort or CrayDoc is included with the third-party product.

Cray Developed Publications Provided with this Release

The publications provided with this release are listed in the table below, which also indicates whether each publication was updated. Publications are provided in HTML and PDF formats.

Table 1. Publications Provided with this Release

Book Title	Number	Updated
Cray Compiling Environment Release Overview and Installation Guide (this document)	8.7	Yes

Book Title	Number	Updated
Cray C and C++ Reference Manual	8.7	Yes
Cray Fortran Reference Manual	8.7	Yes

Additional Documentation Resources

The below table lists additional resources for obtaining documentation not included with this release package.

Table 2. Additional Documentation Resources

Product	Documentation Source
GNU compilers	Documentation for the GNU C and Fortran compilers is available at *http://gcc.gnu.org/onlinedocs/*
`glibc`	`glibc` documentation is available at *http://gcc.gnu.org/onlinedocs*
GLIB	GLIB documentation is available at *http://developer.gnome.org/glib/stable*
RPM	RPM documentation is available at *http://www.rpm.org*

Other Related Documents Available

The following publications contain additional information to help set up the Cray Compiling Environment 8.7; they are not provided with this release but are supplied with other products purchased from Cray:

- *Cray Programming Environments Installation Guide (16.06)*
- *Cray Programming Environment User's Guide (S-2529)*
- *Managing System Software for the Cray Linux Environment (S-2393)*
- *Installing the Cray Programming Environment for Cray CS Systems (S-2800)*

5 Cray Compiling Environment 8.7 Release Package Overview

Hardware and Software Requirements

This CCE release is supported on:

- Cray XC systems with CLE 6.0 or CLE 5.2, and Cray Developer Toolkit (CDT) 17.12 (or later) release.
- Cray CS systems with RedHat 7.2 or RedHat 7.3, and Cray CS Programming Environment (CPE-CCS) 17.11 (or later) release.

Contents of the Release Package

The release package includes:

- Cray Fortran Compiler, version 8.7
- Cray C and C++ Compiler, version 8.7
- CrayLibs (libraries and utilities), version 8.7
- Documentation, described in the *Cray Compiling Environment 8.7 Documentation Overview* topic.

6 Install Cray Compiling Environment 8.7

Cray Compiling Environment 8.7 is installed on the shared root. `root` permissions are necessary in order to install this software. Cray Compiling Environment 8.7 requires that asynchronous products are installed on the system.

For Cray XC systems, please refer to the most recent version of the *Cray Programming Environments Installation Guide (S-2372)* available at *http://pubs.cray.com*. For Cray CS systems, please refer to the most recent version of *Installing the Cray Programming Environment for Cray CS Systems (S-2800).*

6.1 Use of the Cray Compiling Environment 8.7

Load the `PrgEnv-cray` module to use CCE.

Load the appropriate targeting modules to set the necessary compile and link options for the system. There are separate modules to target the desired processor, system network, and optionally an accelerator. The appropriate modules for the system may be loaded by default when logging in, if configured to do so by the system administrator.

Examples of processor targeting modules are `craype-x86-skylake` and `craype-mic-knl`. Examples of system network modules are `craype-network-aries` and `craype-network-infiniband`. Examples of accelerator modules are `craype-accel-nvidia35` (for Kepler) and `craype-accel-nvidia60` (for Pascal), where 35 and 60 refer to the NVIDIA compute capability level. When using an accelerator, the module environment forces dynamic linking.

Use either `ftn`, `cc`, or `CC` command to compile.

Because of the multiple compiling environments potentially available on Cray systems, the `ftn(1)`, `cc(1)`, and `CC(1)` man pages provide basic introductions to the compiler environment. For more information about the Cray compiler command-line options, see the `crayftn(1)`, `craycc(1)`, and `crayCC(1)` man pages.

For more detailed information about the Cray compiler options, directives, pragmas, and optimizations, see *Cray Fortran Reference Manual* and *Cray C and C++ Reference Manual*.