Intel® C++ Composer XE 2013 SP1 for Linux* Installation Guide and Release Notes

Document number: 321412-004US
29 July 2013

Table of Contents

1 Introduction

This document describes how to install the product, provides a summary of new and changed features and includes notes about features and problems not described in the product documentation.

Due to the nature of this comprehensive integrated software development tools solution, different Intel® C++ Composer XE components may be covered by different licenses. Please see the licenses included in the distribution as well as the Disclaimer and Legal Information section of these release notes for details.

1.1 Change History

This section highlights important from the previous product version and changes in product updates. For information on what is new in each component, please read the individual component release notes.

1.1.1 Changes since Intel® Composer XE 2013

- Online installation
- GUI installation
- Intel® C++ Compiler XE 14.0.0
- GNU* Project Debugger (GDB*)
- Fedora* 18 and 19 are now supported
- Ubuntu* 13.04 and Debian* 7.0 are now supported
- Support for the following versions of Linux distributions has been dropped:
    - Fedora* 17
    - Ubuntu* 11.10
    - Pardus* 2011.2
- Features from C++11 (-std=c++11)
- Partial OpenMP* 4.0 RC2 support
- Intel® Cilk™ Plus changes
- DWARF V4 support
- __INTEL_COMPILER_UPDATE predefined macro
- Pointer type alignment qualifiers
- Variable definition attributes to avoid false sharing
- -mtune performance tuning option
- Using offload code in shared libraries requires main program to be linked with –offload=mandatory or –offload=optional option
- -openmp-offload/-openmp-simd options added for controlling the enabling/disabling of specific OpenMP* 4.0 features independently of other OpenMP features
- __GXX_EXPERIMENTAL_CXX0X__ Macro Not Supported
- -xATOM_SSE4.2 option added to support Silvermont microarchitecture
- Intel® Math Kernel Library 11.1
- Intel® Integrated Performance Primitives 8.0 update 1
- Intel® Threading Building Blocks 4.2

1.2 Product Contents

*Intel® C++ Composer XE 2013 SP1 for Linux** includes the following components:

- Intel® C++ Compiler XE 14.0.0 for building applications that run on IA-32, Intel® 64 architecture systems and Intel® Xeon Phi™ coprocessors running the Linux* operating system
- GNU* Project Debugger (GDB*) 7.5
- Intel® Debugger 13.0
- Intel® Integrated Performance Primitives 8.0 update 1
- Intel® Math Kernel Library 11.1
- Intel® Threading Building Blocks 4.2
- Integration into the Eclipse* development environment
- On-disk documentation

1.3 System Requirements

For an explanation of architecture names, see http://intel.ly/q9JVjE

- A PC based on an IA-32 or Intel® 64 architecture processor supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium® 4 processor or later, or compatible non-Intel processor)
    - Development of 64-bit applications or applications targeting Intel® MIC Architecture is supported on a 64-bit version of the OS only. Development of 32-bit applications is supported on either 32-bit or 64-bit versions of the OS Development for a 32-bit on a 64-bit host may require optional library components (ia32-libs, lib32gcc1, lib32stdc++6, libc6-dev-i386, gcc-multilib, g++-multilib) to be installed from your Linux distribution.
- For Intel® MIC Architecture development/testing:
    - Intel® Xeon Phi™ coprocessor
    - Intel® Manycore Platform Software Stack (Intel® MPSS)
- For the best experience, a multi-core or multi-processor system is recommended
- 1GB of RAM (2GB recommended)
- 2.5GB free disk space for all features
- One of the following Linux distributions (this is the list of distributions tested by Intel; other distributions may or may not work and are not recommended - please refer to Technical Support if you have questions):
    - Fedora* 18,19
    - Red Hat Enterprise Linux* 5, 6
    - SUSE LINUX Enterprise Server* 10, 11 SP2
    - Ubuntu* 12.04 LTS, 13.04
    - Debian* 6.0, 7.0
    - Intel® Cluster Ready
- Linux Developer tools component installed, including gcc, g++ and related tools
    - gcc versions 4.1-4.8 supported
    - binutils versions 2.17-2.23 supported

- Library libunwind.so is required in order to use the –traceback option. Some Linux distributions may require that it be obtained and installed separately.

*Additional requirements to use GDB**

- To use the provided GDB*, Python* version 2.6 or 2.7 is required.

Additional requirements to use the Graphical User Interface of the Intel® Debugger

- Java* Runtime Environment (JRE) 6.0 (also called 1.6†) – 5.0 recommended
    - A 32-bit JRE must be used on an IA-32 architecture system and a 64-bit JRE must be used on an Intel® 64 architecture system

Additional requirements to use the integration into the Eclipse development environment*

- Eclipse Platform version 4.2 with:
    - Eclipse C/C++ Development Tools (CDT) 8.1 or later
    - Java* Runtime Environment (JRE) 6.0 (also called 1.6†) or later
- Eclipse Platform version 3.8 with:
    - Eclipse C/C++ Development Tools (CDT) 8.1 or later
    - Java* Runtime Environment (JRE) 6.0 (also called 1.6†) or later
- Eclipse Platform version 3.7 with:
    - Eclipse C/C++ Development Tools (CDT) 8.0 or later
    - Java* Runtime Environment (JRE) 6.0 (also called 1.6†) or later

† There is a known issue with JRE 6.0 through update 10 that causes a crash on Intel® 64 architecture. It is recommended to use the latest update for your JRE. See http://www.eclipse.org/eclipse/development/readme_eclipse_3.7.html section 3.1.3 for details.

Notes

- The Intel compilers are tested with a number of different Linux distributions, with different versions of gcc. Some Linux distributions may contain header files different from those we have tested, which may cause problems. The version of glibc you use must be consistent with the version of gcc in use. For best results, use only the gcc versions as supplied with distributions listed above.
- The default for the Intel® compilers is to build IA-32 architecture applications that require a processor supporting the Intel® SSE2 instructions - for example, the Intel® Pentium® 4 processor. A compiler option is available to generate code that will run on any IA-32 architecture processor. However, if your application uses Intel® Integrated Performance Primitives or Intel® Threading Building Blocks, executing the application will require a processor supporting the Intel® SSE2 instructions.
- Compiling very large source files (several thousands of lines) using advanced optimizations such as -O3, -ipo and -openmp, may require substantially larger amounts of RAM.

- The above lists of processor model names are not exhaustive - other processor models correctly supporting the same instruction set as those listed are expected to work. Please refer to Technical Support if you have questions regarding a specific processor model
- Some optimization options have restrictions regarding the processor type on which the application is run. Please see the documentation of these options for more information.

1.3.1 Red Hat Enterprise Linux 5* and SuSE Enterprise Linux 10* are deprecated

Support for these operating system versions is deprecated, and support may be removed in a future release.

1.3.2 IA-64 Architecture (Intel® Itanium®) Development Not Supported

This product version does not support development on or for IA-64 architecture (Intel® Itanium®) systems. The version 11.1 compiler remains available for development of IA-64 architecture applications.

1.4 Documentation

Product documentation can be found in the `Documentation` folder as shown under Installation Folders.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

1.5 Samples

Samples for each product component can be found in the `Samples` folder as shown under Installation Folders.

1.6 Japanese Language Support

Intel compilers provide support for Japanese language users. Error messages, visual development environment dialogs and some documentation are provided in Japanese in addition to English. By default, the language of error messages and dialogs matches that of your operating system language selection. Japanese-language documentation can be found in the `ja_JP` subdirectory for documentation and samples.

Japanese language support will be available in an update on or after the release of Intel® C++ Composer XE 2013.

If you wish to use Japanese-language support on an English-language operating system, or English-language support on a Japanese-language operating system, you will find instructions at http://intel.ly/qhINDv

1.7 Technical Support

If you did not register your compiler during installation, please do so at the Intel® Software Development Products Registration Center at http://registrationcenter.intel.com. Registration entitles you to free technical support, product updates and upgrades for the duration of the support term.

For information about how to find Technical Support, Product Updates, User Forums, FAQs, tips and tricks, and other support information, please visit http://www.intel.com/software/products/support/

Note: If your distributor provides technical support for this product, please contact them for support rather than Intel.

2 Installation

The installation of the product requires a valid license file or serial number. If you are evaluating the product, you can also choose the “Evaluate this product (no serial number required)” option during installation.

If you received your product on DVD, mount the DVD, change the directory (`cd`) to the top-level directory of the mounted DVD and begin the installation using the command:

```
./install.sh
```

If you received the product as a downloadable file, first unpack it into a writeable directory of your choice using the command:

```
tar -xzvf name-of-downloaded-file
```

Then change the directory (`cd`) to the directory containing the unpacked files and begin the installation using the command:

```
./install.sh
```

Follow the prompts to complete installation.

Note that there are several different downloadable files available, each providing different combinations of components. Please read the download web page carefully to determine which file is appropriate for you.

You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions.

Please do not run the install script as a background process (i.e. running "./install.sh &"). This is not supported.

2.1 GUI installation now available in Intel® Composer XE 2013 SP1

If on a Linux* system with GUI support, the installation will now provide a GUI-based installation. If a GUI is not supported (for example if running from an ssh terminal), a command-line installation will be provided.

2.2 Online Installation now available in Intel® Composer XE 2013 SP1

The default electronic installation package for Intel® Composer XE 2013 SP1 now consists of a smaller installation package that dynamically downloads and then installs packages selected to be installed. This requires a working internet connection and potentially a proxy setting if you are behind an internet proxy. Full packages are provided alongside where you download this online install package if a working internet connection is not available.

2.2.1 `http_proxy` is set, but sudo installation still fails to connect

Most sudo profiles are set to not inherit certain settings like `http_proxy` from the original user. Make sure your /etc/sudoers file contains a line like the following to allow your proxy settings to propagate:

```
Defaults   env_keep += "http_proxy"
```

2.3 Intel® Software Manager

The installation now provides an Intel® Software Manager to provide a simplified delivery mechanism for product updates and provide current license status and news on all installed Intel® software products.

You can also volunteer to provide Intel anonymous usage information about these products to help guide future product design. This option, the Intel® Software Improvement Program, is not enabled by default – you can opt-in during installation or at a later time, and may opt-out at any time. For more information please see http://intel.ly/SoftwareImprovementProgram.

2.4 Installation of Intel® Manycore Platform Software Stack (Intel® MPSS)

The Intel® Manycore Platform Software Stack (Intel® MPSS) may be installed before or after installing the Intel® Composer XE 2013 SP1 for Linux* including Intel® MIC Architecture product.

Refer to the Intel® MPSS documentation for the necessary steps to install the user space and kernel drivers.

2.5 Cluster Installation

To install a product on multiple nodes of a cluster on Linux*, the following steps should be taken:

1) Run the installation on a system where Intel® Cluster Studio is installed. Also, passwordless ssh must be configured between machines in a cluster.
2) On step "`4 Options`" there will be a "`Cluster installation`" option. The default value is "`Current node`".
3) To install on a cluster, the user must select this option and then provide a `machines.LINUX` file with IP-addresses, hostnames, FQDNs, and other information for the cluster nodes (one node per line). The first line is expected to describe the current (master) node.
4) Once the `machines.LINUX` file is provided, additional options will appear: `Number of parallel installations`, `Check for shared installation directory`.
5) When all options are configured and installation has begun, the installation will check connectivity with all nodes (a prerequisite) and only then will it install the product on these nodes.

2.6 Silent Install

For information on automated or “silent” install capability, please see http://intel.ly/ngVHY8.

2.7 Using a License Server

If you have purchased a "floating" license, see http://intel.ly/pjGfwC for information on how to install using a license file or license server. This article also provides a source for the Intel® License Server that can be installed on any of a wide variety of systems.

2.8 Eclipse* Integration Installation

Please refer to the section below on Eclipse Integration

2.9 Security-Enhanced Linux* (SELinux*)

In previous Composer XE versions, installation required setting the `SELINUX` mode to `permissive`. Starting with Composer XE 2013, this is no longer required.

2.10 Known Installation Issues

- On some versions of Linux, auto-mounted devices do not have the "exec" permission and therefore running the installation script directly from the DVD will result in an error such as:

  ```
  bash: ./install.sh: /bin/bash: bad interpreter: Permission denied
  ```

  If you see this error, remount the DVD with exec permission, for example:

  ```
  mount /media/<dvd_label> -o remount,exec
  ```

  and then try the installation again.
- The product is fully supported on Ubuntu* and Debian* Linux distributions for IA-32 and Intel® 64 architecture systems as noted above under System Requirements. Due to a restriction in the licensing software, however, it is not possible to use the Trial License feature when evaluating IA-32 components on an Intel® 64 architecture system under Ubuntu or Debian. This affects using a Trial License only. Use of serial numbers, license files, floating licenses or other license manager operations, and off-line activation (with

serial numbers) is not affected. If you need to evaluate IA-32 components of the product on an Intel® 64 architecture system running Ubuntu or Debian, please visit the Intel® Software Evaluation Center (http://intel.ly/nJS8y8) to obtain an evaluation serial number.

2.11 Installation Folders

The compiler installs, by default, under `/opt/intel` – this is referenced as `<install-dir>` in the remainder of this document. You are able to specify a different location, and can also perform a "non-root" install in the location of your choice.

The directory organization has changed since the Intel® Compilers 11.1 release.

While the top-level installation directory has also changed between the original C++ Composer XE 2011 release and Composer XE 2013, the `composerxe` symbolic link can still be used to reference the latest product installation.

Under `<install-dir>` are the following directories:

- `bin` – contains symbolic links to executables for the latest installed version
- `lib` – symbolic link to the lib directory for the latest installed version
- `include` – symbolic link to the include directory for the latest installed version
- `man` – symbolic link to the directory containing man pages for the latest installed version
- `ipp` – symbolic link to the directory for the latest installed version of Intel® Integrated Performance Primitives
- `mkl` – symbolic link to the directory for the latest installed version of Intel® Math Kernel Library
- `tbb` – symbolic link to the directory for the latest installed version of Intel® Threading Building Blocks
- `ism` – contains files for Intel® Software Manager
- `composerxe` – symbolic link to the `composer_xe_2013` directory
- `composer_xe_2013_sp1` – directory containing symbolic links to subdirectories for the latest installed Intel® Composer XE 2013 SP1 compiler release
- `composer_xe_2013_sp1.<n>.<pkg>` - physical directory containing files for a specific compiler version. `<n>` is the update number, and `<pkg>` is a package build identifier.

Each `composer_xe_2013_sp1` directory contains the following directories that reference the latest installed Intel® Composer XE 2013 SP1 compiler:

- `bin` – directory containing scripts to establish the compiler environment and symbolic links to compiler executables for the host platform
- `pkg_bin` – symbolic link to the compiler `bin` directory
- `include` – symbolic link to the compiler `include` directory
- `lib` – symbolic link to the compiler `lib` directory
- `ipp` – symbolic link to the `ipp` directory

- `mkl` – symbolic link to the `mkl` directory
- `tbb` – symbolic link to the `tbb` directory
- `debugger` – symbolic link to the `debugger` directory
- `eclipse_support` – symbolic link to the `eclipse_support` directory
- `man` – symbolic link to the `man` directory
- `Documentation` – symbolic link to the `Documentation` directory
- `Samples` – symbolic link to the `Samples` directory

Each `composer_xe_2013_sp1.<n>.<pkg>` directory contains the following directories that reference a specific update of the Intel® Composer XE 2013 SP1 compiler:

- `bin` – all executables
- `pkg_bin` – symbolic link to `bin` directory
- `compiler` – shared libraries and header files
- `debugger` – debugger files
- `Documentation` – documentation files
- `man` – `man` pages
- `eclipse_support` – files to support Eclipse integration
- `ipp` – Intel® Integrated Performance Primitives libraries and header files
- `mkl` – Intel® Math Kernel Library libraries and header files
- `tbb` – Intel® Threading Building Blocks libraries and header files
- `Samples` – Product samples and tutorial files
- `Uninstall –` Files for uninstallation
- `.scripts` – scripts for installation

If you have both the Intel C++ and Intel Fortran compilers installed, they will share folders for a given version and update.

This directory layout allows you to choose whether you want the latest compiler, no matter which version, the latest update of the Intel® Composer XE 2013 compiler, or a specific update. Most users will reference `<install-dir>/bin` for the `compilervars.sh [.csh]` script, which will always get the latest compiler installed. This layout should remain stable for future releases.

2.12 Removal/Uninstall

Removing (uninstalling) the product should be done by the same user who installed it (root or a non-root user). If `sudo` was used to install, it must be used to uninstall as well. It is not possible to remove the compiler while leaving any of the performance library or Eclipse* integration components installed.

1. Open a terminal window and set default (`cd`) to any folder outside `<install-dir>`
2. Type the command: `<install-dir>/bin/uninstall.sh`
3. Follow the prompts

4. Repeat steps 2 and 3 to remove additional platforms or versions

If you have the same-numbered version of Intel® Fortran Compiler installed, it may also be removed.

If you have added the Intel C++ Eclipse integration to an instance of Eclipse in your environment, you will need to update your Eclipse configuration by removing the Intel integration extension site from your Eclipse configuration. To do this, Go to Help > About Eclipse and click on "Installation Details". Select "Intel(R) C++ Compiler XE 14.0 for Linux* OS " under "Installed Software" and click on "Uninstall..." Click "Finish". When asked to restart Eclipse, select "Yes".

3 Intel® Many Integrated Core Architecture (Intel® MIC Architecture)

This section summarizes changes, new features and late-breaking news about the Intel Composer XE 2013 for Linux* including Intel® MIC Architecture.

3.1 About Intel® Composer XE 2013 for Linux* including Intel® MIC Architecture

The Intel® Composer XE 2013 for Linux* including Intel® MIC Architecture extends the feature set of the Intel® C++ Composer XE 2013 and the Intel® Fortran Composer XE 2013 products by enabling predefined sections of code to execute on an Intel® Xeon Phi™ coprocessor.

These sections of code run on the coprocessor if it is available. Otherwise, they run on the host CPU.

This document uses the terms *coprocessor* and *target* to refer to the target of an offload operation.

The current components of Intel® Composer XE 2013 that support Intel® MIC Architecture are the:

- Intel® C++ and Fortran Compilers
- Intel® Debugger (Intel® IDB)
- Intel® Math Kernel Library (Intel® MKL)
- Intel® Threading Building Blocks (Intel® TBB)
- Eclipse* IDE Integration

3.2 Compatibility

This release supports the Intel® Xeon Phi™ coprocessor. Refer to the Technical Support section for additional information.

It's recommended to rebuild all code with Composer XE 2013 update 1 or later due to this binary compatibility change in the offload libraries.

3.3 Getting Started

There is only one compiler that generates code both for Intel® 64 architecture and for Intel® MIC Architecture. Refer to the section on Establishing the Compiler Environment to get started, using intel64 as the architecture you setup for. Refer to the Notes section below for further changes.

3.4 Product Documentation

Documentation concerning the Intel® MIC Architecture for Composer XE 2013 is currently undergoing change. For the latest documentation updates, please go to our web site at http://intel.ly/MxPFYx.

3.5 Debuggers

For graphical debugging, use the Eclipse* integration pointing to the `idb_mpm` debugger startup script (under `bin/intel64_mic`). You can attach to code running on Intel® MIC Architecture or you can debug code offloaded from the CPU.

Use of the debuggers on a remote system through SSH requires setting the DISPLAY environment variable to your local X display to minimize lag caused by SSH display forwarding.

The package also contains command line versions of these debuggers. They are called `idbc` (for the Intel® 64 architecture host) and `idbc_mic` (for the Intel® MIC Architecture target).

Please note that the auto-attach feature is not supported in the command line versions of the debuggers.

3.6 Intel® Math Kernel Library (Intel® MKL)

For details on Intel® MIC Architecture support, see the section on Intel MKL.

3.7 Notes

3.7.1 Intel C++ Compiler

3.7.1.1 Using offload code in shared libraries requires main program to be linked with –offload=mandatory or –offload=optional option

There is initialization required for offload that can only be done in the main program. For offload code in shared libraries, this means that the main program must also be linked for offload so that the initialization happens. This will happen automatically if the main code or code statically linked with the main program contains offload constructs. If that is not the case, you will need to link the main program with the `-offload=mandatory` or `-offload=optional` compiler options.

3.7.1.2 New offload clauses in Intel® Composer XE 2013 SP1

The three clauses “mandatory”, “optional”, and “status” have been added to the offload directives in Intel® Composer XE 2013 SP1.

- “mandatory”: Offloaded code aborts if card not available for offload, if “status” clause not added. If “status” clause added, user code directs action.
- “optional”: If card is not available for download, code runs on CPU.

These clauses in code override offload compiler option settings.

3.7.1.3 –offload option changed in Intel® Composer XE 2013 SP1

-offload now takes a keyword in Composer XE 2013 SP1

-offload=none: Any offload directives are ignored and cause warnings to be emitted at compile-time

-offload=mandatory (default): Any offload directives are processed. If card is not available for offload, program aborts.

-offload=optional: Any offload directives are processed. If card is not available, code runs on CPU.

These options are overridden by user-specified offload clauses.

3.7.1.4 New environment variables to control offload behavior in Intel® Composer XE 2013 SP1

Several new environment variables have been added:

- OFFLOAD_DEVICES: Restricts the process to use only the Intel® Xeon™ Phi coprocessor cards specified by the variable.
- OFFLOAD_INIT: Specifies a hint to the offload runtime when it should initialize Intel® Xeon™ Phi coprocessors.
- OFFLOAD_REPORT: Supports several levels of tracing and statistical information from offload.
- OFFLOAD_ACTIVE_WAIT: Controls keeping the CPU busy during DMA transfers.

3.7.1.5 Runtime errors or crashes when running an application built with the initial Intel® Composer XE 2013 product release with the offload libraries from 2013 update 1

There is a breaking binary compatibility change in the offload libraries for Intel® Composer XE 2013 update 1 that will cause runtime errors or crashes if you use the libraries from update 1 or later with a binary built with the initial release of the Intel Composer XE 2013 compiler. Examples of the errors you may observe in this situation are:

Error 1:

```
***Warning: offload to device #0 : failed
```

Error 2:

```
Segmentation fault (core dumped)
```

Error 3:

```
terminate called after throwing an instance of 'COIRESULT'
terminate called recursively
```

Error 4:

```
CARD--ERROR:1 myoiPageFaultHandler: (nil) Out of Range!
CARD--ERROR:1 _myoiPageFaultHandler: (nil) switch to default signal
handle
CARD--ERROR:1 Segment Fault!
HOST--ERROR:myoiScifGetRecvId: Call recv() Header Failed ! errno = 104
^CHOST--ERROR:myoiScifSend: Call send() Failed! errno = 104
HOST--ERROR:myoiSend: Fail to send message!
HOST--ERROR:myoiBcastToOthers: Fail to send message to 1!
HOST--ERROR:myoiBcast: Fail to send message to others!
```

To resolve these issues, you should recompile your application fully with the Intel Composer XE 2013 update 1 or newer compiler in order to use the offload libraries included in the new package.

3.7.1.6 Default code generation no longer supports Intel® Xeon Phi™ coprocessor A0 stepping in Composer XE 2013 Update 1

Composer XE 2013 update 1 now generates new streaming store instructions that were introduced in the Intel® Xeon Phi™ coprocessor B0 stepping. These instructions are not supported on the A0 stepping, and will cause runtime errors. If you require your application to run on A0 steppings, use the option `-opt-streaming-stores never` to avoid generating these instructions. This may decrease performance on B0 or later steppings.

3.7.1.7 Missing symbols not detected at link time

In the offload compilation model, the binaries targeting the Intel® MIC Architecture are generated as dynamic libraries (.so). Dynamic libraries do not need all referenced variables or routines to be resolved during linking as these can be resolved during load time. This behavior could mask some missing variable or routine in the application resulting in a failure during load time. In order to identify and resolve all missing symbols at link time, use the following command line option to list the unresolved variables.

```
-offload-option,mic,compiler,"-z defs"
```

*3.7.1.8 *MIC* tag added to compile-time diagnostics*

The compiler diagnostics infrastructure is modified to add an additional offload *MIC* tag to the output message to allow differentiation from the Target (Intel® MIC Architecture) and the host CPU compilations. The additional tag appears only in the Target compilation diagnostics issued when compiling with offload extensions for Intel® MIC Architecture.

In the examples below the sample source programs trigger identical diagnostics during both the host CPU and Target Intel® MIC Architecture compilations; however, some programs will generate different diagnostics during these two compilations. The new tag permits easier association with either the CPU or Target compilation.

```
$ icc -c sample.c
sample.c(1): warning #1079: *MIC* return type of function "main" must
be "int"
  void main()
       ^

sample.c(5): warning #120: *MIC* return value type does not match the
function type
      return 0;
             ^

sample.c(1): warning #1079: return type of function "main" must be
"int"
  void main()
       ^

sample.c(5): warning #120: return value type does not match the
function type
      return 0;
```

3.7.1.9 Runtime Type Information (RTTI) not supported

Runtime Type Information (RTTI) is not supported under the Virtual-Shared memory programming method; specifically, use of `dynamic_cast<>` and `typeid()` is not supported.

3.7.1.10 Direct (native) mode requires transferring runtime libraries like libiomp5.so to coprocessor

The Intel® Manycore Platform Software Stack (Intel® MPSS) no longer includes Intel compiler libraries under /lib, for example the OpenMP* library, libiomp5.so.

When running OpenMP* applications in direct mode (i.e. on the coprocessor card), users must first upload (via scp) a copy of the Intel® MIC Architecture OpenMP* library (`<install_dir>/compiler/lib/mic/libiomp5.so`) to the card (device names will be of the format `micN`, where the first card will be named `mic0`, the second `mic1`, and so on) before running their application.

Failure to make this library available will result in a run-time failure like:

```
/libexec/ld-elf.so.1: Shared object "libiomp5.so" not found, required
by "sample"
```

This can also apply to other compiler runtimes like libimf.so. The required libraries will depend on the application and how it's built.

3.7.1.11 Calling exit() from an offload region

When calling exit() from within an offload region, the application terminates with an error diagnostic "`offload error: process on the device 0 unexpectedly exited with code 0`"

3.7.2 Debugging, GNU Project Debugger (GDB*), and the Intel® Debugger (IDB)

3.7.2.1 Using the GNU Project Debugger (GDB) in Eclipse**

There is a document eclmigdb_config_guide.pdf in the Documentation/{en_US|ja_JP}/debugger/gdb directory that explains how to use GDB under Eclipse.

*3.7.2.2 The IDB Debugger may fail to setup the command line argument for the debuggee under Eclipse**

The debugger may not set the command line argument for the debuggee correctly under Eclipse when loading an application using the `file' command in GDB mode. The debuggee may abort with the message:

```
*** abort -internal failure : get_command_argumentfailed
```

In this case, add the executable to the command line argument of IDB.

3.7.2.3 Eclipse fails to display local variables*

Local variables cannot be seen under the Eclipse environment while debugging an application.

Workaround: Enter the local variable into the Expression Window of Eclipse to get its value.

3.7.2.4 Safely ending offload debug sessions

To avoid issues like orphan processes or stale debugger windows when ending offload applications, manually end the debugging session before the application is reaching its exit code. The following procedure is recommended for terminating a debug session.

- Manually stop a debug session before the application reaches the exit-code.
- When stopped, press the red stop button in the toolbar in the Intel® MIC Architecture-side debugger first. This will end the offloaded part of the application.
- Next, do the same in the CPU-side debugger.
- The link between the two debuggers will be kept alive. The Intel® MIC Architecture-side debugger will stay connected to the debug agent and the application will remain loaded in the CPU-side debugger, including all breakpoints that have been set.
- At this point, both debugger windows can safely be closed.

3.7.2.5 Intel® MIC Architecture-side debugger asserts on setting source dirs

Setting source directories in the GNU Project Debugger (GDB*) might lead to an assertion.

Resolution:

The assertion should not affect debugger operation. To avoid the assertion anyway, don't use source directory settings. The debugger will prompt you to browse for files it cannot locate automatically.

3.7.2.6 Accessing _Cilk_shared variables in the debugger

Writing to a shared variable in an offloaded section from within the CPU-side debugger **before** the CPU-side debuggee has accessed that variable may result in loss of the written value/might display a wrong value or cause the application to crash.

Consider the following code snippet:

```
_Cilk_shared bool is_active;
_Cilk_shared my_target_func() {
//Accessing "is_active" from the debugger *could* lead to unexpected
//results e.g. a lost write or outdated data is read.
is_active = true;
//Accessing "is_active" (read or write) from the debugger at this
//point is considered safe e.g. correct value is displayed.
}
```

4 Intel® C++ Compiler

This section summarizes changes, new features and late-breaking news about the Intel C++ Compiler.

4.1 Compatibility

In version 11.0, the IA-32 architecture default for code generation changed to assume that Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions are supported by the processor on which the application is run. See below for more information.

4.2 New and Changed Features

C++ Composer XE 2013 SP1 now contains Intel® C++ Compiler XE 14.0. The following features are new or significantly enhanced in this version. For more information on these features, please refer to the documentation.

- Features from C++11 (-std=c++11)
  - Complete (instead of partial) implementation of initializer lists. See N2672 and N3217.
  - Complete implementation of inline namespaces. See N2535.
  - Complete implementation of non-static data member initializers. See N2756.
  - Complete implementation of generalized constant expressions. See N2235.
  - Complete implementation of unrestricted unions. See N2544.
  - Delegating constructors. See N1986.
  - Rvalue references for *this. See N2439.
  - Raw string literals. See N2442.
  - Conversions of lambdas to function pointers.

    - Implicit move constructors and assignment operators. See N3053.
    - __bases and __direct_bases type traits.
    - The context-sensitive keyword "final" can now be used on a class definition, and "final" and "override" can be used on member function declarations. See N2928, N3206, and N3272.
    - Complete implementation of the "noexcept" specifier and operator. See N3050. Includes the late instantiation of noexcept per core issue 1330.
- Partial OpenMP* 4.0 RC1 and TR1 support
- Intel® Cilk™ Plus changes
- DWARF V4 support
- __INTEL_COMPILER_UPDATE predefined macro
- Pointer type alignment qualifiers
- Variable definition attributes to avoid false sharing
- -mtune performance tuning option

4.2.1 Updated Support for Upcoming OpenMP* features added in Composer XE 2013 SP1

Composer XE 2013 SP1 adds support for certain preliminary OpenMP* features. The features added as defined in the OpenMP* 4.0 Public Review Release Candidate 2 specification available from http://openmp.org are:

- SIMD pragmas, directives, and clauses
- TARGET directives for attached accelerators
- `#pragma omp taskgroup` construct
- Atomic clause `seq_cst`
- Six new forms of atomic capture and update:
    - `Atomic swap: {v = x; x = expr;}`
    - `Atomic update: x = expr binop x;`
    - `Atomic capture 1: v = x = x binop expr;`
    - `Atomic capture 2: v =x = expr binop x;`
    - `Atomic capture 3: {x = expr binop x; v = x;}`
    - `Atomic capture 4: {v = x; x = expr binop x;}`
- `proc_bind(<type>)` clause where <type> is “spread”, “close”, or “master”
- OMP_PLACES environment variable
- OMP_PROC_BIND environment variable
- `omp_get_proc_bind()` API

For more information, see http://intel.ly/W7CHjb.

4.2.2 Intel® Cilk™ Plus changes in Intel® C++ Composer XE 2013 SP1

Please note the following new features for Intel® Cilk™ Plus in Intel C++ Composer XE 2013 SP1:

- Elemental function implementation has changed to be more compatible with other vector function implementations in gcc and OpenMP*. This breaks binary compatibility with

previous Intel® C++ Compiler versions (13.1 and earlier). You should either rebuild all codes using elemental functions with the version 14.0 compiler, or use the –vecabi=legacy compiler option to use the previous implementation.

- New multiply reducer defined in `cilk/reducer_opmul.h`
- Three new array notation reduction intrinsics have been added to support bitwise reduction operations:
  - `__sec_reduce_and`
  - `__sec_reduce_or`
  - `__sec_reduce_xor`

4.2.3 New attribute for pointers and pointer types to specify assumed data alignment in Composer XE 2013 SP1

__declspec(align_value(N)) and __attribute__((align_value(N))) have been added to indicate to the compiler it can assume the specified alignment “N” when using the attributed pointer type. For example:

```
typedef float float_a16
__attribute__((align_value (16)));

void foo(float_a16 *restrict dest, float_a16 *restrict src){
```

Let’s the compiler know that the src and dest arguments should be aligned by the user on 16-byte boundaries.

4.2.4 New attribute to variable declarations to avoid false sharing in Composer XE 2013 SP1

__declspec(avoid_false_share)/__attribute__((avoid_false_share)) and __declspec(avoid_false_share(identifier))/__attribute__((avoid_false_share(identifier))) have been added to indicate to the compiler that the variable attributed should be suitably padded or aligned to avoid false sharing with any other variable. If an identifier is specified, then any variables attributed with that identifier will be padded or aligned to avoid false sharing with any other variables except those others with the same identifier. These attributes must be on variable definitions in function, global, or namespace scope. If in function scope, the scope of the identifier is the current function. If the variable definition is in global or namespace scope, the scope of the identifier is in the current compilation unit.

4.2.5 New __INTEL_COMPILER_UPDATE predefined macro in Composer XE 2013 SP1

A new __INTEL_COMPILER_UPDATE predefined macro can now be used to obtain the minor update number for the Intel® Compiler being used. For example, for a compiler version 14.0.2, the macro would preprocess to “2”.

4.2.6 Static Analysis Feature (formerly “Static Security Analysis” or “Source Checker”) Requires Intel® Inspector XE

The “Source Checker” feature, from compiler version 11.1, has been enhanced and renamed “Static Analysis”. The compiler options to enable Static Analysis remain the same as in compiler version 11.1 (for example, `-diag-enable sc`), but the results are now written to a

file that is interpreted by Intel® Inspector XE rather than being included in compiler diagnostics output.

4.3 New and Changed Compiler Options

For details on these and all compiler options, see the Compiler Options section of the on-disk documentation.

4.3.1 New and Changed in Composer XE 2013 SP1

- -[no-]openmp-offload
- -[no-]openmp-simd
- -xATOM_SSE4.2
- -xATOM_SSSE3
- -vecabi=<arg>
- -gdwarf-4
- -standalone
- -offload=<arg>
- -mtune=<arch>
- -mlong-double-64
- -mlong-double-80

For a list of deprecated compiler options, see the Compiler Options section of the documentation.

4.3.2 –[no-]openmp-offload and –[no-]openmp-simd added to Composer XE 2013 SP1

These two options allow you to enable/disable the TARGET and SIMD features of OpenMP* 4.0 independently of support of the rest of OpenMP* (enabled with –openmp). When –openmp is specified, -openmp-offload and –openmp-simd are set as well, allowing the use of these features.

4.3.3 –mtune added to Composer XE 2013 SP1

-mtune=<arch> can now be used to specify the compiler “tuning” for a specific architecture, similar to how the equivalent gcc* option behaves.

4.3.4 –gdwarf-4 added to Composer XE 2013 SP1

Support for generating DWARF V4 debug information is now available via the –gdwarf-4 option.

4.3.5 -vec-report7 added to Composer XE 2013 Update 2

A new vectorizer reporting level has been added to update 2 to provide more detailed and advanced information on loop vectorization. See the article at http://intel.ly/XeSkW6 for more information.

4.3.6 –gcc-version is deprecated in Composer XE 2013 Update 2

-gcc-version functionality has been superseded by –gcc-name. –gcc-version has therefore been deprecated and may be removed from a future release.

4.4 Other Changes

4.4.1 __attribute__((always_inline)) now requires inline keyword to enable inlining with Composer XE 2013 SP1

In previous Intel compiler versions, a routine declared with the "always_inline" attribute would always be inlined. In Composer XE 2013 SP1, the compiler now requires that the routine also be inline (either explicitly declared that way using the "inline" keyword or implicitly inline because it is a member function whose definition appears inside the class) in order for the routine to be inlined. The compiler will now match gcc behavior and also give a warning for this, i.e.:

```
// t.cpp
__attribute__((always_inline)) int foo2(int x)  // need to add
"inline" keyword also
{
  return x;
}

icpc -c t.cpp
t.cpp(2): warning #3414: the "always_inline" attribute is ignored on
non-inline functions
  __attribute__((always_inline)) int foo2(int x)
       ^
```

4.4.2 Establishing the Compiler Environment

The `compilervars.sh` script is used to establish the compiler environment. `compilervars.csh` is also provided.

The command takes the form:

`source <install-dir>/bin/compilervars.sh` *`argument`*

Where *`argument`* is either `ia32` or `intel64` as appropriate for the architecture you are building for. Establishing the compiler environment also establishes the environment for the Intel® Debugger, Intel® Performance Libraries and, if present, Intel® Fortran Compiler.

4.4.3 Instruction Set Default Changed to Require Intel® Streaming SIMD Extensions 2 (Intel® SSE2)

When compiling for the IA-32 architecture, `-msse2` (formerly `-xW`) is the default. Programs built with `-msse2` in effect require that they be run on a processor that supports the Intel® Streaming SIMD Extensions 2 (Intel® SSE2), such as the Intel® Pentium® 4 processor and some non-Intel processors. No run-time check is made to ensure compatibility – if the program is run on an unsupported processor, an invalid instruction fault may occur. Note that this may change floating point results since the Intel® SSE instructions will be used instead of the x87 instructions and therefore computations will be done in the declared precision rather than sometimes a higher precision.

All Intel® 64 architecture processors support Intel® SSE2.

To specify the older default of generic IA-32, specify `-mia32`

4.5 Known Issues

4.5.1 `__GXX_EXPERIMENTAL_CXX0X__` Macro Not Supported

In the Gnu* version 4.8 or later environments, using the `-std=c++11` or `-std=gnu++0x` options may lead to a diagnostic of the form:

```
This file requires compiler and library support for the upcoming ISO
C++ standard, C++0x. This support is currently experimental, and must
be enabled with the -std=c++0x or -std=gnu++0x compiler options.
```

The Intel compiler does not currently define the `__GXX_EXPERIMENTAL_CXX0X__` macro in gcc 4.8 mode, since it does not yet support some C++11 features enabled by the macro in the C++ standard library headers. This may lead to incompatibilities with g++ when using the C++ standard library in the `-std=c++11` or `-std=gnu++0x` modes.

4.5.2 Missing documentation for functions to check decimal floating-point status

To detect exceptions occurring during decimal floating-point arithmetic, use the following floating-point exception functions:

Function	Brief Description
fe_dec_feclearexcept	Clears the supported floating-point exceptions
fe_dec_fegetexceptflag	Stores an implementation-defined representation of the states of the floating-point status flags
fe_dec_feraiseexcept	Raises the supported floating-point exceptions
fe_dec_fesetexceptflag	Sets the floating-point status flags
fe_dec_fetestexcept	Determines which of a specified subset of the floating point exception flags are currently set

The decimal floating-point exception functions are defined in the fenv.h header file.

Similar binary floating-point exception functions are described in ISO C99.

To compile the source using DFP, use the preprocessor macro __STDC_WANT_DEC_FP__.

4.5.3 Intel® Cilk™ Plus Known Issues

- Static linkage of the runtime is not supported

  Static versions of the Intel® Cilk™ Plus library are not provided by design. Using `-static-intel` to link static libraries will generate an expected warning that the dynamic version of the of Intel® Cilk™ Plus library, libcilkrts.so, is linked.

  ```
  $ icc -static-intel sample.c

  icc: warning #10237: -lcilkrts linked in dynamically, static
  library not available
  ```

  Alternatively, you can build the open source version of Intel Cilk Plus with a static runtime. See http://cilk.com for information on this implementation of Intel Cilk Plus.

4.5.4 Guided Auto-Parallel Known Issues

Guided Auto Parallel (GAP) analysis for single file, function name or specific range of source code does not work when Whole Program Interprocedural Optimization (-ipo) is enabled

4.5.5 Static Analysis Known Issues

4.5.5.1 Excessive false messages on C++ classes with virtual functions

Note that use of the Static Analysis feature also requires the use of Intel® Inspector XE.

Static analysis reports a very large number of incorrect diagnostics when processing any program that contains a C++ class with virtual functions. In some cases the number of spurious diagnostics is so large that the result file becomes unusable.

If your application contains this common C++ source construct, add the following command line switch to suppress the undesired messages: `/Qdiag-disable:12020,12040` (Windows) or `-diag-disable 12020,12040` (Linux). **This switch must be added at the link step because that is when static analysis results are created.** Adding the switch at the compile step alone is not sufficient.

If you are using a build specification to perform static analysis, add the `-disable-id 12020,12040` switch to the invocation of the inspxe-runsc, for example,

```
inspxe-runsc -spec-file mybuildspec.spec -disable-id 12020,12040
```

If you have already created a static analysis result that was affected by this issue and you are able to open that result in the Intel® Inspector XE GUI, then you can hide the undesired messages as follows:

- The messages you will want to suppress are `"Arg count mismatch"` and `"Arg type mismatch"`. For each problem type, do the following:
- Click on the undesired problem type in the Problem filter. This hides all other problem types.
- Click on any problem in the table of problem sets
- Type control-A to select all the problems

- Right click and select *Change State -> Not a problem* from the pop-up menu to set the state of all the undesired problems
- Reset the filter on problem type to All
- Repeat for the other unwanted problem type
- Set the Investigated/Not investigated filter to *Not investigated*. You may have to scroll down in the filter pane to see it as it is near the bottom. This hides all the undesired messages because the "Not a problem" state is considered a "not investigated" state.

5 GDB* Debugger

This section summarizes the changes, new features, customizations and known issues related to the GDB* delivered with Intel® Composer XE 2013 SP1.

5.1 Features

The GDB* delivered with Intel® Composer XE 2013 SP1 is based on GDB 7.5 with enhancements provided by Intel. This debugger is planned to replace the Intel® Debugger in a future release. In addition to features found in GDB 7.5, there are several other new features:

- Support for Intel® Many Integrated Core Architecture (Intel® MIC Architecture)
- Intel® Parallel Debug Extension for data race detection
- Trace enhancements based on Branch Trace Store
- Intel® Pointer Checker support
- Support for Intel® Transactional Synchronization Extensions (Intel® TSX)

5.2 Starting the debugger

To start the GDB delivered with Intel® Composer XE, the compiler environment has to be set, after that you should call "gdb-ia" (for CPU debugging) or "gdb-mic" (for debugging on Intel® Many Integrated Core Architecture).

5.3 Documentation

In the documentation folder (<installdir>/Documentation/<locale>) there is full GDB* documentation.

5.4 Compatibility

Python* version 2.6 or 2.7 is required.

6 Intel® Debugger (IDB)

The following notes refer to the Graphical User Interface (GUI) available for the Intel® Debugger (IDB) when running on IA-32 and Intel® 64 architecture systems. In this version, the `idb` command invokes the GUI – to get the command-line interface, use `idbc`.

6.1 Support Deprecated for Intel® Debugger

In a future major release of the product, the Intel® Debugger may be removed. This would remove the ability to use:

- The `idbc` command line debugger
- The `idb` GUI based debugger

6.2 Setting up the Java* Runtime Environment

The Intel® IDB Debugger graphical environment is a Java application and requires a Java Runtime Environment (JRE) to execute. The debugger will run with a 6.0 (also called 1.6) JRE.

Install the JRE according to the JRE provider's instructions.

Finally you need to export the path to the JRE as follows:

```
export PATH=<path_to_JRE_bin_dir>:$PATH
```

6.3 Starting the Debugger

To start the debugger, first make sure that the compiler environment has been established as described at Establishing the Compiler Environment. Then use the command:

```
idb
```

or

```
idbc
```

as desired.

Once the GUI is started and you see the console window, you're ready to start the debugging session.

Note: Make sure that the executable you want to debug is built with debug info and is an executable file. Change permissions if required, e.g. `chmod +x <application_bin_file>`

6.4 Additional Documentation

Online help titled *Intel® Compilers / Intel® Debugger Online Help* is accessible from the debugger graphical user interface as `Help > Help Contents`.

Context-sensitive help is also available in several debugger dialogs where a `Help` button is displayed.

6.5 Debugger Features

6.5.1 Main Features of IDB

The debugger supports all features of the command line version of the Intel® IDB Debugger. Debugger functions can be called from within the debugger GUI or the GUI-command line. Please refer to the Known Limitations when using the graphical environment.

6.5.2 Inspector XE 2011 Update 6 Supports “break into debug” with IDB

Inspector XE 2011 Update 6 now supports “break into debug” mode with the Composer XE 2011 Update 6 and later versions of IDB. Refer to the Inspector XE 2011 Release Notes for more information.

6.6 Known Issues and Changes

6.6.1 Thread Data Sharing Filters may not work correctly

Setting Thread Data Sharing Filters may lead to unexpected behavior of the debugger. It may happen that threads will not continue after a data sharing detection and the debugger may exit with a SIG SEGV.

If you encounter issues related to Data Sharing Detection with filters enabled, disable all filters in the ‘Thread Data Sharing Filters’ window context menu.

6.6.2 Core File Debugging

To be able to debug core files you need to start the debugger (command line debugger `idbc` or GUI debugger `idb`) with command-line options as follows:

```
idb|idbc <executable> <corefile>
```

<or>

```
idb|idbc <executable> –core <corefile>
```

Once started with a core file, the debugger is not able to debug a live process e.g. attaching or creating a new process. Also, when debugging a live process a core file cannot be debugged.

6.6.3 Debugger crash if $HOME not set on calling shell

The debugger will end with a “Segmentation fault” if no $HOME environment variable is set on the shell the debugger is started from.

6.6.4 Command line parameter –idb and -dbx not supported

The debugger command line parameters –idb and -dbx are not supported in conjunction with the debugger GUI.

6.6.5 Watchpoints limitations

For IA-32 and Intel® 64 architecture systems there are the following limitations (if possible IDB will raise appropriate error messages to assist the user):

- Possible sizes of the watched memory areas are only 1, 2, 4 or 8 (Intel® 64 architecture only) bytes.
- The start address of the watched memory area has to be aligned with its size. For example it is not possible to watch 2 bytes starting with an odd address.
- There is only support for a maximum of 4 active/enabled watchpoints. Unused ones can be disabled to free resources and to enable/create other ones.
- Only the following access modes are supported:

    - Write: trigger on write accesses
    - Any: trigger on either write or read accesses
    - Changed: trigger on write accesses that actually changed the value
- Watched memory areas must not overlap each other.
- Watchpoints are not scope related but tied to a process. As long as a process exists the watchpoints are active/enabled. Only if the process is terminated (e.g. rerun) will the watchpoints will be disabled. They can be enabled again if the user wishes to do so.
- Using the debugger to access the watched memory area (e.g. assign a different value to a variable) bypasses the hardware detection. Hence watchpoints only trigger if the debuggee itself accessed the watched memory area.
- If the debuggee is running on a guest OS inside a virtual machine, stepping over an instruction or code line might continue the process without stopping. Watchpoints are only guaranteed to work when the debuggee runs on real hardware.

6.6.6 Position Independent Executable (PIE) Debugging not Supported

On some systems the compiler is tuned to produce Position Independent Executable (PIE) code. In those cases the flag –fno-pie has to be used both for compilation and linking, otherwise the application cannot be debugged.

6.6.7 Command line parameter –parallel not supported

The debugger command line parameter –parallel is not supported on the shell command prompt or on the Console Window of the Debugger GUI.

6.6.8 Signals Dialog Not Working

The Signals dialog accessible via the GUI dialog Debug / Signal Handling or the shortcut Ctrl+S is not working correctly. Please refer to the Intel® Debugger (IDB) Manual for use of the signals command line commands instead.

6.6.9 Resizing GUI

If the debugger GUI window is reduced in size, some windows may fully disappear. Enlarge the window and the hidden windows will appear again.

6.6.10 `$cdir, $cwd` Directories

`$cdir` is the compilation directory (if recorded). This is supported in that the directory is set; but `$cdir` is not itself supported as a symbol.

`$cwd` is the current working directory. Neither the semantics nor the symbol is supported.

The difference between `$cwd` and '.' is that `$cwd` tracks the current working directory as it changes during a debug session. '.' is immediately expanded to the current directory at the time an entry to the source path is added.

6.6.11 `info stack` Usage

The GDB mode debugger command `info stack` does not currently support negative frame counts the way GDB does, for the following command:

```
info stack [num]
```

A positive value of num prints the innermost num frames, a zero value prints all frames and a negative one prints the innermost –num frames in reverse order.

6.6.12 `$stepg0` Default Value Changed

The debugger variable $stepg0 changed default to a value of 0. With the value "0" the debugger will step over code without debug information if you do a "step" command. Set the debugger variable to 1 to be compatible with previous debugger versions as follows:

```
(idb) set $stepg0 = 1
```

6.6.13 SIGTRAP error on some Linux* Systems

On some Linux distributions (e.g. Red Hat Enterprise Linux Server release 5.1 (Tikanga)) a SIGTRAP error may occur when the debugger stops at a breakpoint and you continue debugging. As a workaround you may define the SIGTRAP signal as follows on command line:

```
(idb) handle SIGTRAP nopass noprint nostop
SIGTRAP is used by the debugger.
SIGTRAP       No        No      No              Trace/breakpoint trap
(idb)
```

Caveat: With this workaround all SIGTRAP signals to the debuggee are blocked.

6.6.14 idb GUI cannot be used to debug MPI processes

The idb GUI cannot be used to debug MPI processes. The command line interface (idbc) can be used for this purpose.

6.6.15 Thread Syncpoint Creation in GUI

While for plain code and data breakpoints the field “Location” is mandatory, thread syncpoints require both “Location” and “Thread Filter” to be specified. The latter specifies the threads to synchronize. Please note that for the other breakpoint types this field restricts the breakpoints created to the threads listed.

6.6.16 Data Breakpoint Dialog

The fields “Within Function” and “Length” are not used. The location to watch provides the watched length implicitly (the type of the effective expression is used). Also “Read” access is not working.

6.6.17 Stack Alignment for IA-32 Architecture

Due to changes in the default stack alignment for the IA-32 architecture, the usage of inferior calls (i.e. evaluation of expressions that cause execution of debuggee code) might fail. This can cause as well crashes of the debuggee and therefore a restart of the debug session. If you need to use this feature, make sure to compile your code with 4 byte stack alignment by proper usage of the `-falign-stack=<mode>` option.

6.6.18 GNOME Environment Issues

With GNOME 2.28, debugger menu icons may not being displayed by default. To get the menu icons back, you need to go to the “System->Preferences->Appearance, Interface” tab and enable, "Show icons in menus". If there is not “Interface” tab available, you can change this with the corresponding `GConf` keys in console as follows:

```
    gconftool-2 --type boolean --set
/desktop/gnome/interface/buttons_have_icons true
    gconftool-2 --type boolean --set
/desktop/gnome/interface/menus_have_icons true
```

6.6.19 Accessing Online-Help

On systems where the Online-Help is not accessible from the IDB Debugger GUI Help menu, you can access the web-based debugger documentation from
http://intel.ly/o5DMp9

7 Eclipse Integration

The Intel C++ Compiler installs an Eclipse feature and associated plugins (the Intel C++ Eclipse Product Extension) which provide support for the Intel C++ compiler when added as an Eclipse product extension site to an existing instance of the Eclipse* Integrated Development Environment (IDE). With this feature, you will be able to use the Intel C++ compiler from within the Eclipse integrated development environment to develop your applications.

7.1 Supplied Integrations

The Intel feature provided in the following directory:

```
<install-dir>/eclipse_support/cdt8.0/eclipse
```

supports and requires Eclipse Platform versions 4.2, 3.8, and 3.7; Eclipse C/C++ Development Tools (CDT) version 8.0 or later; and a functional Java Runtime Environment (JRE) version 6.0 (also called 1.6) update 11 or later.

7.1.1 Integration notes

If you already have the proper versions of Eclipse, CDT and a functional JRE installed and configured in your environment, then you can add the Intel C++ Eclipse Product Extension to your Eclipse Platform, as described in the section, below, entitled How to Install the Intel C++ Eclipse Product Extension in Your Eclipse Platform. Otherwise, you will first need to obtain and install Eclipse, CDT and a JRE, as described in the section, below, entitled How to Obtain and Install Eclipse, CDT and a JRE and then install the Intel C++ Eclipse Product Extension.

If your installation of Eclipse already has an earlier Intel® C++ Composer XE integration installed, installing the updated integration will not work. You will need to install a fresh version of Eclipse into which you can install the latest Composer XE integration. For this same reason, using the Eclipse update mechanism to install a newer Composer XE integration will not work.

7.2 How to Install the Intel C++ Eclipse Product Extension in Your Eclipse Platform

To add the Intel C++ product extension to your existing Eclipse configuration, follow these steps, from within Eclipse.

Open the "Available Software" page by selecting: `Help > Install New Software...` Click on the "Add..." button. Select "Local...". A directory browser will open. Browse to select the `eclipse` directory in your Intel C++ compiler installation. For example, if you installed the compiler as root to the default directory, you would browse to `/opt/intel/composer_xe_2013.<n>.<xxx>/eclipse_support/cdt8.0/eclipse`. Select “OK” to close the directory browser. Then select "OK" to close the “Add Site” dialog. Select the two boxes for the Intel C++ integration: there will be one box for “Intel® C++ Compiler Documentation” and a second box for “Intel® C++ Compiler XE 14.0 for Linux* OS”. Note: The Intel features will not be visible if you have Group items by category set – unset this option to view the Intel features. If you also installed the Intel® Debugger (idb) with its Eclipse product extension and would like to use idb from within Eclipse, repeat the above steps for the idb product extension site.

Click the “Next” button. An “Install” dialog will open which gives you a chance to review and confirm you want to install the checked items. Click “Next”. You will now be asked to accept the license agreement. Accept the license agreement and click “Finish”. Select “OK” on the “Security Warning” dialog that says you are installing software that contains unsigned content. The installation of the Intel support will proceed.

When asked to restart Eclipse, select “Yes”. When Eclipse restarts, you will be able to create and work with CDT projects that use the Intel C++ compiler. See the Intel C++ Compiler documentation for more information. You can find the Intel C++ documentation under `Help > Help Contents > Intel(R) C++ Compiler XE 14.0 User and Reference Guides`.

7.2.1 Integrating the GNU* Project Debugger into Eclipse

See the section Adding GDB* (provided by Intel® Composer XE) to the Eclipse* CDT.

7.2.2 Integrating the Intel® Debugger into Eclipse

After completing the above steps, including restarting Eclipse, follow these steps to integrate the Intel® Debugger into Eclipse:

- Create a Debug launch configuration by selecting Run > Debug Configurations…
- In the dialog box that pops up, right click on `C/C++ Application` and select New.
- You will now see some tabs on the right. At the bottom-right you should see a label `Using GDB (DSF) Create Process Launcher - Select other…` Click this label – a new dialog will appear. Select `Standard Create Process Launcher` and click OK.
- Go to the Debugger tab and select the Intel® Debugger (idbc) from the combo box. Replace `idbc` with the full path to idbc.

7.3 How to Obtain and Install Eclipse, CDT and a JRE

Eclipse is a Java application and therefore requires a Java Runtime Environment (JRE) to execute. The choice of a JRE is dependent on your operating environment (machine architecture, operating system, etc.) and there are many JRE's available to choose from.

A package containing both Eclipse 4.2 and CDT 8.1 is available from:

http://www.eclipse.org/downloads/

Scroll down to find “Eclipse IDE for C/C++ Developers”. Choose either the Linux 32-bit or Linux 64-bit download as desired.

7.3.1 Installing JRE, Eclipse and CDT

Once you have downloaded the appropriate files for Eclipse, CDT, and a JRE, you can install them as follows:

1. Install your chosen JRE according to the JRE provider's instructions.
2. Create a directory where you would like to install Eclipse and `cd` to this directory. This directory will be referred to as `<eclipse-install-dir>`
3. Copy the Eclipse package binary `.tgz` file to the `<eclipse-install-dir>` directory.
4. Expand the `.tgz` file.
5. Start `eclipse`

You are now ready to add the Intel C++ product extension to your Eclipse configuration as described in the section, *How to Install the Intel C++ Eclipse Product Extension in Your Eclipse Platform*. If you need help with launching Eclipse for the first time, please read the next section.

7.4 Launching Eclipse for Development with the Intel C++ Compiler

If you have not already set your `LANG` environment variable, you will need to do so. For example,

```
setenv LANG en_US
```

Setup Intel C++ compiler related environment variables by executing the `compilervars.csh` (or `.sh`) script prior to starting Eclipse:

`source <install-dir>/bin/compilervars.csh arch_arg` (where "arch_arg" is one of "ia32" or "intel64").

Since Eclipse requires a JRE to execute, you must ensure that an appropriate JRE is available to Eclipse prior to its invocation. You can set the `PATH` environment variable to the full path of the folder of the `java` file from the JRE installed on your system or reference the full path of the java executable from the JRE installed on your system in the `-vm` parameter of the Eclipse command, e.g.:

```
eclipse -vm /JRE folder/bin/java
```

Invoke the Eclipse executable directly from the directory where it has been installed. For example:

```
<eclipse-install-dir>/eclipse/eclipse
```

7.5 Installing on Fedora* Systems

If the Intel C++ Compiler for Linux is installed on an IA-32 or Intel® 64 architecture Fedora* system as a "local" installation, i.e. not installed as root, the installation may fail to properly execute the Eclipse graphical user interfaces to the compiler or debugger. The failure mechanism will typically be displayed as a `JVM Terminated` error. The error condition can also occur if the software is installed from the root account at the system level, but executed by less privileged user accounts.

The cause for this failure is that a more granular level of security has been implemented on Fedora, but this new security capability can adversely affect access to system resources, such as dynamic libraries. This new *SELinux* security capability may require adjustment by your system administrator in order for the compiler installation to work for regular users.

7.6 Selecting Compiler Versions

For Eclipse projects you can select among the installed versions of the Intel C++ Compiler. On IA-32 architecture systems, the supported Intel compiler versions are 9.1, 10.0, 10.1, 11.0, 11.1, 12.0, 12.1, 13.0, and 14.0. On Intel® 64 architecture systems, only compiler versions 11.0, 11.1, 12.0, 12.1, 13.0, and 14.0 are supported.

8 Intel® Integrated Performance Primitives

This section summarizes changes, new features and late-breaking news about this version of Intel® Integrated Performance Primitives (Intel® IPP).

The latest information on Intel® IPP 8.0 can be found in the product release notes under `<install dir>/composer_xe_2013_sp1.x.xxx/Documentation/<locale>/ipp/ReleaseNotes.htm`.

For detailed information about IPP see the following links:

- **New features**: see the information below and visit the main Intel IPP product page on the Intel web site at: http://intel.ly/OG5IF7; and the Intel IPP Release Notes at http://intel.ly/OmWI4d.
- **Documentation, help, and samples**: see the documentation links on the IPP product page at: http://intel.ly/OG5IF7.

8.1 Intel® IPP Cryptography Libraries are Available as a Separate Download

The Intel® IPP cryptography libraries are available as a separate download. For download and installation instructions, please read http://intel.ly/ndrGnR

8.2 Intel® IPP Code Samples

The Intel® IPP code samples are organized into downloadable packages at http://intel.ly/pnsHxc

The samples include source code for audio/video codecs, image processing and media player applications, and for calling functions from C++, C# and Java*. Instructions on how to build the sample are described in a readme file that comes with the installation package for each sample.

9 Intel® Math Kernel Library

This section summarizes changes, new features and late-breaking news about this version of the Intel® Math Kernel Library (Intel MKL). All the bug fixes can be found here: http://intel.ly/OeHQqf

9.1 Notices

Please refer to the Knowledge Base article on Deprecations for more information on the following notices

- Intel® MKL now provides a choice of components to install. Components necessary for PGI* compiler, Compaq Visual Fortran* Compiler, SP2DP interface, BLAS95 and LAPACK95 interfaces, Cluster support (ScaLAPACK and Cluster DFT) and Intel® Many Integrated Core Architecture (Intel® MIC Architecture) support are not installed unless explicitly selected during installation
- Unaligned Conditional Numerical Reproducibility (CNR) is not available for Intel MKL Cluster components (ScaLAPACK and Cluster DFT)
- Examples for using Intel MKL with Boost* uBLAS and Java* have been removed from product distribution and placed in the following articles:
    - How to use Intel MKL with Java*
    - How to use Boost* uBLAS with Intel MKL

9.2 Changes in This Version

9.2.1 What's New in Intel MKL 11.1

- Conditional Numerical Reproducibility : Introduced support for Conditional Numerical Reproducibility (CNR) mode on unaligned data
- Introduced MP LINPACK support for heterogeneous clusters - clusters whose nodes differ from each other, either by processor type or by having varying number of attached Intel® Xeon Phi™ coprocessors
- Improved performance of CNR=AUTO mode on recent AMD* systems
- BLAS:

    - Improved performance of [S/D]GEMV on all Intel processors supporting Intel® SSE4.2 and later
    - Optimized [D/Z]GEMM and double-precision Level 3 BLAS functions on Intel® Advanced Vector Extensions 2 (Intel® AVX2)
    - Optimized [Z/C]AXPY and [Z/C]DOT[U/C] on Intel® Advanced Vector Extensions (Intel® AVX) and Intel AVX2
    - Optimized sequential version of DTRMM on Intel MIC Architecture
    - Tuned DAXPY on Intel AVX2
- LAPACK:
    - Improved performance of (S/D)SYRDB and (S/D)SYEV for large dimensions when only eigenvalues are needed
    - Improved performance of xGESVD for small sizes like M,N<10
- VSL:
    - Added support and examples for mean absolute deviation
    - Improved performance of Weibull Random Number Generator (RNG) for alpha=1
    - Added support of raw and central statistical sums up to the 4th order, matrix of cross-products and median absolute deviation
    - Added a VSL example designed by S. Joe and F. Y. Kuo illustrating usage of Sobol QRNG with direction numbers which supports dimensions up to 21,201
    - Improved performance of SFMT19937 Basic Random Number Generator (BRNG) on Intel MIC Architecture
- DFT:
    - Improved performance of double precision complex-to-complex transforms on Intel MIC Architecture
    - Optimized complex-to-complex DFT on Intel AVX2
    - Optimized complex-to-complex 2D DFT on Intel® Xeon processor E5 v2 series
    - Improved performance for workloads specific to GENE application on Intel Xeon E5-series (Intel AVX) and on Intel AVX2
    - Improved documentation data layout for DFTI compute functions
    - Introduced scaling in large real-to-complex FFTs
- Data Fitting:
    - Improved performance of df?Interpolate1D and df?SearchCells1D functions on Intel Xeon processors and Intel MIC Architecture
    - Improved performance of df?construct1d function for linear and Hermite/Bessel/Akima cubic types of splines on Intel MIC Architecture, Intel® Xeon® processor X5570 and Intel® Xeon® processor E5-2690
- Transposition
    - Improved performance of in-place transposition for square matrices
- Examples and tests for using Intel MKL are now packaged as an archive to shorten the installation time

9.3 Attributions

As referenced in the End User License Agreement, attribution requires, at a minimum, prominently displaying the full Intel product name (e.g. "Intel® Math Kernel Library") and

providing a link/URL to the Intel® MKL homepage (http://www.intel.com/software/products/mkl) in both the product documentation and website.

The original versions of the BLAS from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/blas/index.html.

The original versions of LAPACK from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/lapack/index.html. The authors of LAPACK are E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. Our FORTRAN 90/95 interfaces to LAPACK are similar to those in the LAPACK95 package at http://www.netlib.org/lapack95/index.html. All interfaces are provided for pure procedures.

The original versions of ScaLAPACK from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/scalapack/index.html. The authors of ScaLAPACK are L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley.

The Intel® MKL Extended Eigensolver functionality is based on the Feast Eigenvalue Solver 2.0 http://www.ecs.umass.edu/~polizzi/feast/

PARDISO in Intel® MKL is compliant with the 3.2 release of PARDISO that is freely distributed by the University of Basel. It can be obtained at http://www.pardiso-project.org.

Some FFT functions in this release of Intel® MKL have been generated by the SPIRAL software generation system (http://www.spiral.net/) under license from Carnegie Mellon University. The Authors of SPIRAL are Markus Puschel, Jose Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo.

10 Intel® Threading Building Blocks

For information on changes to Intel® Threading Building Blocks, please read the file `CHANGES` in the TBB documentation directory.

11 Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING

BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:

http://www.intel.com/products/processor%5Fnumber/

The Intel® C++ Compiler, Intel® Debugger, Intel® Integrated Performance Primitives, Intel® Math Kernel Library, and Intel® Threading Building Blocks are provided under Intel's End User License Agreement (EULA).

The GNU* Project Debugger, GDB is provided under the General GNU Public License GPL V3.

Please consult the licenses included in the distribution for details.

Celeron, Centrino, Intel, Intel logo, Intel386, Intel486, Atom, Core, Itanium, MMX, Pentium, VTune, Cilk, Xeon Phi, and Xeon are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.