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

Document number: 321412-004US
24 January 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.

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 Update 2 (2013.2)

- Intel® C++ Compiler XE 13.1.0
- Intel® Math Kernel Library 11.0 Update 2
- Intel® Threading Building Blocks 4.1 Update 2
- Support for OpenMP* SIMD and accelerator features
- New KMP_PLACE_THREADS environment variable
- New __INTEL_PRE_CFLAGS and __INTEL_POST_CFLAGS environment variables
- New -vec-report7 vectorization report level
- -W[no-]pch-messages to enable/disable precompiled header diagnostics
- Corrections to reported problems

1.1.2 Update 1 (2013.1)

- Intel® C++ Compiler XE 13.0.1
- Intel® Debugger 13.0.1
- Intel® Math Kernel Library 11.0 Update 1
- Intel® Integrated Performance Primitives 7.1 Update 1
- Intel® Threading Building Blocks 4.1 Update 1
- Breaking binary compatibility change in Update 1 offload libraries for Intel® Many Integrated Core Architecture
- Default code generation for Intel® Xeon Phi™ processors no longer supported on A0 processor steppings
- Support for Eclipse* platform versions 4.2 and 3.8 and Eclipse CDT 8.1
- Inline assembly and intrinsic support for Intel architecture code named Broadwell
- -opt-assume-safe-padding support for Intel® Many Integrated Core architecture
- -opt-streaming-cache-evict and -opt-threads-per-core support for Intel® Many Integrated Core architecture
- -check-pointers=w

- core_4th_gen_avx added for manual cpu dispatch in Composer XE 2013 Update 1
- Microsoft* loop pragma support
- Corrections to reported problems

1.1.3 Changes since Intel® C++ Composer XE 2011

- Development of applications that offload work to or natively run on an Intel® Xeon Phi™ coprocessor is now supported. For details, see the section on Intel® Many Integrated Core Architecture.
- Intel® C++ Compiler updated to version 13.0.
- Intel® Debugger updated to version 13.0
  - Intel® Debugger support deprecated
- Intel® Math Kernel Library updated to version 11.0
  - Removed support for Intel® Pentium® III processor. See the Knowledge Base article on Deprecations for further information.
- Intel® Integrated Performance Primitives updated to version 7.1
  - Intel® IPP static threaded libraries now available in separate package
- Intel® Threading Building Blocks updated to version 4.1
- Fedora 17*, SUSE* LINUX Enterprise Server 11 SP2, Ubuntu 11.10* and Ubuntu 12.04* now supported
- Support for the following versions of Linux distributions has been dropped:
  - Red Hat Enterprise Linux 4*
  - SUSE* LINUX Enterprise Server 11 SP1
  - Fedora 15*
  - Ubuntu 11.04*
  - Ubuntu 10.04*
  - Asianux*
- The Intel® Software Manager has been added to help you manage product updates and license activation
- New C++11 features
- Improved support for future Intel processors
- Out-of-bounds memory checking
- New Warning Level –w3 and Changes to Warning Levels in Composer XE 2013
- Static Analysis Improvements

1.2 Product Contents

*Intel® C++ Composer XE 2013 Update 2 for Linux** includes the following components:

- Intel® C++ Compiler XE 13.1.0 for building applications that run on IA-32, Intel® 64, and Intel® Many Integrated Core Architecture (Intel® MIC Architecture) systems running the Linux* operating system
- Intel® Debugger 13.0
- Intel® Integrated Performance Primitives 7.1 Update 1
- Intel® Math Kernel Library 11.0 Update 2
- Intel® Threading Building Blocks 4.1 Update 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™ processor
  - 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* 17
  - Red Hat Enterprise Linux* 5, 6
  - SUSE LINUX Enterprise Server* 10, 11 SP2
  - Ubuntu* 11.10, 12.04
  - Debian* 6.0
  - Intel® Cluster Ready
  - Pardus* 2011.2 (x64 only)
- Linux Developer tools component installed, including gcc, g++ and related tools
- 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 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

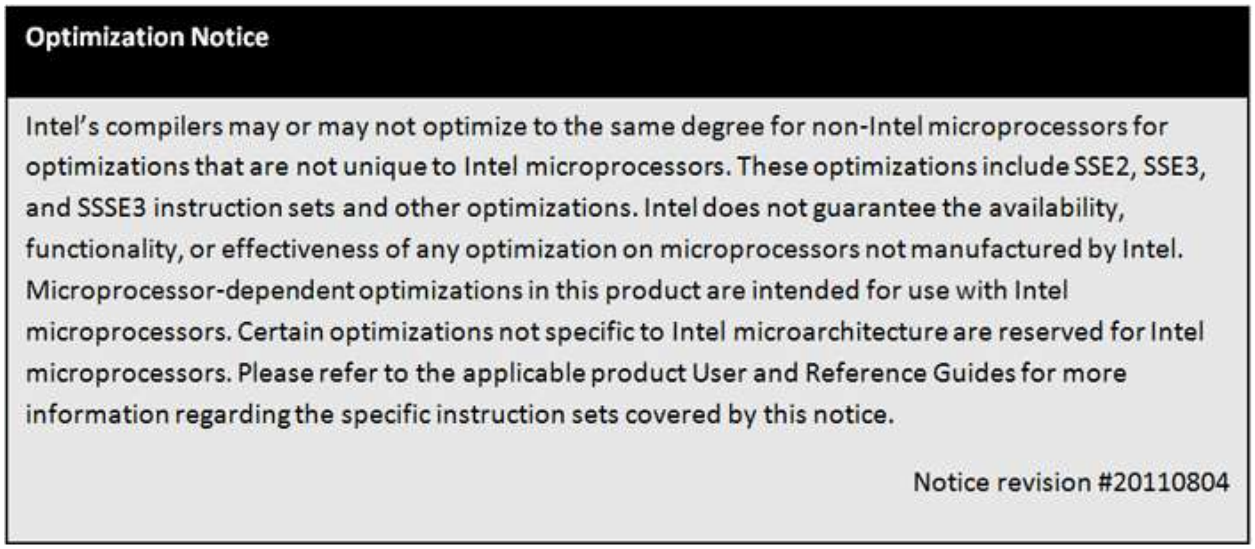
- The Intel compilers are tested with a number of different Linux distributions, with different versions of gcc. Some Linux distributions may contain header files different from those we have tested, which may cause problems. The version of glibc you use must be consistent with the version of gcc in 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 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.

2.1 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.

2.2 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 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.

2.3 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.

2.4 Silent Install

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

2.5 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.

2.6 Eclipse* Integration Installation

Please refer to the section below on Eclipse Integration

2.7 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.

2.8 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*, Debian*, and Pardus* 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, Debian or Pardus. 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 Ubuntu, Debian, or Pardus system, please visit the Intel® Software Evaluation Center (http://intel.ly/nJS8y8) to obtain an evaluation serial number.

2.9 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
- `composerxe` – symbolic link to the `composer_xe_2013` directory
- `composer_xe_2013` – directory containing symbolic links to subdirectories for the latest installed Intel® Composer XE 2013 compiler release
- `composer_xe_2013.<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` directory contains the following directories that reference the latest installed Intel® Composer XE 2013 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.<n>.<pkg>` directory contains the following directories that reference a specific update of the Intel® Composer XE 2013 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
- `.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.

2.10 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 13.0 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 due to this binary compatibility change in the offload libraries.

If you need generated code to support A0 steppings of this coprocessor, please see the note recommending use of `-opt-streaming-stores never`.

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 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 update 1 libraries with a binary built with the previous 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 compiler in order to use the offload libraries included in the new package.

3.7.1.2 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.

3.7.1.3 –opt-assume-safe-padding new in Composer XE 2013 Update 1

By default, the compiler should not assume that sizes of objects in memory are greater than written in program even though it is allowed to insert padding if needed. When `–opt-assume-safe-padding` is set, the compiler may assume that variables and dynamically allocated memory is padded, so that accessing up to 64 bytes beyond what is specified in the program source will not cause any faults. This new option provides a way for the programmer to declare safety of access slightly beyond object boundary which may be used by optimizations to loosen safety considerations required by some Intel® Many Integrated Core instructions (Intel® MIC Instructions) and thus emit faster code. The option doesn't tell or require the compiler to provide such padding for static and automatic objects: providing padding is the programmer's responsibility. The option simply changes the assumptions the compiler can safely make. The default is disabled (`-no-opt-assume-safe-padding`).

3.7.1.4

3.7.1.5 –opt-streaming-cache-evict and –opt-threads-per-core new in Composer XE 2013 Update 1

The option `–opt-streaming-cache-evict` specifies cache eviction level when streaming loads/stores are used on Intel® MIC Architecture:

`–opt-streaming-cache-evict =0`

    no cache evict instructions generated

`–opt-streaming-cache-evict =1`

    L1 cache eviction instruction generated after streaming load/store

`–opt-streaming-cache-evict =2`

    L2 cache eviction instruction generated after streaming load/store

`–opt-streaming-cache-evict =3`

L1 and L2 cache eviction instruction generated after streaming load/store

The option `–opt-threads-per-core` specifies the number of hardware threads per core on the Intel® MIC Architecture to be used for an application. The number of threads specified can be a number between 1 and 4.

3.7.1.6 New -fimf-domain-exclusion Compiler Option

The following compiler option information was omitted from the on-disk documentation.

`fimf-domain-exclusion`

Indicates the domain on which a function is evaluated.

IDE Equivalent

None

Architectures

Intel® 64 architecture, targeting Intel® MIC Architecture

Syntax

Linux*: `-fimf-domain-exclusion=`*`classList`*`[:`*`funcList`*`]`

OS X*: None

Windows*: None

Arguments

classList A comma separated list of:

- One or more of the five floating point value class names you can exclude from the function domain without affecting the correctness of your program
- One of the short-hand tokens.

The class names are:

- extremes
- nans
- infinities
- denormals
- zeros

The short-hand tokens are:

- none: None of the above are excluded from the domain.
- all: All of the above are excluded from the domain.
- common: Same as extremes,nans,infinities,denormals.

The order of the class tokens is unimportant, and you can specify each token more than once.

funcList A comma separated list of function names.

If you don't specify this argument, the domain restrictions apply to all math library functions.

Default

None

Description

This indicates the domain on which a function is evaluated, and specifies that your program will function correctly if the functions specified in *funcList* do not produce standard conforming results on the number classes.

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 Tuning Memory Allocation Performance

This content updates/overrides similar content in the C++ Composer XE 2013 User's Guide.

For user-allocated data on the coprocessor, using large (2 MB) page allocations via `mmap()`, instead of `malloc` or `_mm_malloc`, may improve application performance.

Not all applications benefit from using a larger page size. In general, the performance impact from a larger page-size depends greatly on the data access pattern. If the application accesses multiple data-structures that are allocated in different pages, having only limit TLB entries for 2 MB pages on the coprocessor can cause performance degradation.

The default page size is 4KB for `malloc` and `_mm_malloc`. To enable your application to use 2MB pages, use the sample method provided below:

```
#include <string.h>
#include <sys/mman.h>
#include <stdio.h>
#include <errno.h>

#define TWO_MB (2*1024*1024ULL)
#define MAP_HUGETLB 0x40000        /* create a huge page mapping */

/* allocate memory using huge page support */
#define MALLOC_2M(size)  \
   mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_ANONYMOUS
|        MAP_SHARED | \
    MAP_HUGETLB| MAP_POPULATE, -1, 0)

/* free allocated memory */
#define FREE_2M(addr, size) munmap(addr, (size + TWO_MB & ~(TWO_MB-1)))

/**
* allocate_huge_pages - Allocate memory using 2MB page support
* @size -  Size of the memory to allocate
*/

static inline void* allocate_huge_pages(size_t size)
{
   size_t sz = size + TWO_MB & ~(TWO_MB-1);
   void* mem = MALLOC_2M(sz);
   if (mem == MAP_FAILED)
       printf("mmap allocation failed\n");
   return mem;
}
```

3.7.1.9 –opt-prefetch-distance compiler option

Syntax:

`-opt-prefetch-distance=n1[,n2]`

Allowed values for n1,n2 are >=0 (non-negative integer values). n2 is optional. If n2 is specified, n1 should be >= n2.

Sample usages:
```
-opt-prefetch-distance=64,32
-opt-prefetch-distance=16,2
-opt-prefetch-distance=24
```

Description:
This option is used for performance tuning. This option allows the user to specify the prefetch-distance. The unit is the number of (possibly-vectorized) iterations that should be used for compiler-generated prefetches inside loops. The value of n1 will be used as the distance for the vprefetch1 instruction (prefetch from memory to L2 cache) and value of n2 (if specified) will be used as the distance for the vprefetch0 instruction (prefetch from L2 cache to L1 cache). If the option is not used, the prefetch distances will be determined based on compiler heuristics. Note that the -opt-prefetch option is turned ON by default for Intel® MIC Architecture.

This option is ignored if –opt-prefetch=0 is specified.

3.7.1.10 *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;
```

3.7.1.11 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.

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

The Intel® Manycore Platform Software Stack (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.

3.7.1.13 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`”

3.7.1.14 Environment Variable for Controlling Offload Behavior

Several additional environment variables are available for controlling offload behavior.

3.7.1.14.1 MIC_ENV_PREFIX

This is the general mechanism to pass environment variable values to each Intel® Xeon Phi™ coprocessor.

The value of `MIC_ENV_PREFIX` sets the value of the prefix which is used to recognize environment variable values intended for coprocessors. For example,

```
setenv MIC_ENV_PREFIX MYCARDS
```

will use “MYCARDS” as the string that indicates that an environment variable is intended for a specific coprocessor.

Environment variable values of the form

`<mic-prefix>_<var>=<value>`

will send `<var>=<value>` to each coprocessor.

Environment variable values of the form `<mic-prefix>_<card-number>_<var>=<value>` will send `<var>=<value>` to the coprocessor numbered `<card-number>`.

Environment variable values of the form

`<mic-prefix>_ENV=<variable1=value1|variable2=value2>`

will send `<variable1>=<value1>` and `<variable2>=<value2>` to each coprocessor.

Environment variable values of the form

`<mic-prefix>_<card-number>_ENV=<variable1=value1|variable2=value2>`

will send `<variable1>=<value1>` and `<variable2>=<value2>` to the coprocessor numbered `<card-number>`.

Examples:

```
setenv MIC_ENV_PREFIX PHI // Defines the prefix to be used
setenv PHI_ABCD abcd // Sets ABCD=abcd on all coprocessors
setenv PHI_2_EFGH efgh // Sets EFGH=efgh on coprocessor 2
setenv PHI_VAR X=x|Y=y // Sets X=x and Y=y on all coprocessors
setenv PHI_4_VAR P=p|Q=q // Sets P=p and Q=q on coprocessor 4
```

3.7.1.14.2 MIC_USE_2MB_BUFFERS
Sets the threshold for creating buffers with large pages. A buffer is created with the large pages hint if its size exceeds the threshold value.

Example:

```
// any variable allocated on a coprocessor that is equal to
// or greater than 100KB in size will be allocated in large pages.
setenv MIC_USE_2MB_BUFFERS 100k
```

3.7.1.14.3 MIC_STACKSIZE
Sets the size of the offload process stack for all Intel® Xeon Phi™ coprocessors used in the application. This is the overall stack size. Use MIC_OMP_STACKSIZE to modify the size of each OpenMP* thread.

Example:

```
setenv MIC_STACKSIZE 100M // Sets MIC stack to 100 MB
```

3.7.1.14.4 OFFLOAD_DEVICES

The environment variable `OFFLOAD_DEVICES` restricts the process to use only the coprocessors specified as the value of the variable. `<value>` is a comma separated list of physical device numbers in the range 0 to (number_of_devices_in_the_system-1).
Devices available for offloading are numbered logically. That is _Offload_number_of_devices() returns the number of allowed devices and device indexes specified in the target specifier of an offload directive are in the range 0 to (number_of_allowed_devices-1).

Example

```
setenv OFFLOAD_DEVICES "1,2"
```

3.7.2 Debugging and Intel® Debugger

3.7.2.1 Using IDB with Intel® Manycore Platform Software Stack (Intel® MPSS Alpha 2 (2.1.3126-x)

When using the Intel® Debugger for Intel® Many Integrated Core architecture with firmware version Intel® MPSS Alpha 2 (2.1.3126-x) the following limitations apply:

- The Eclipse* CDT integration of the Intel® Debugger does not support "Attach to process" for native applications running on the coprocessor
- When debugging native coprocessor applications on the command line, the remote debug agent `idbserver_mic` is uploaded and started using scp/ssh. This implies that the user id used to start `idbc_mic` must also exist on the coprocessor card. Unless passwordless authentication has been configured for this user id, scp and ssh will require a password being typed.
- When debugging native coprocessor applications on the command line, the shared library `libmyodbl-service.so`, that used to be uploaded to the coprocessor automatically needs to be uploaded manually now.

  The way to achieve this is to create an overlay, so the file is uploaded at boot time. Follow the instructions on how to use overlays in the Intel® MPSS readme.txt (currently available from the "`Intel® SDP MAKC1 Family`" product in Intel® Premier Support). The steps needed to implement this specific overlay look like this:

  a. Create `/etc/sysconfig/mic/conf.d/myo.conf` containing the following:
  ```
  # MYO download files
  Overlay /
  /opt/intel/mic/myo/config/myo.filelist
  ```
  b. Create `/opt/intel/mic/myo/config/myo.filelist` containing
  ```
  dir /lib64 755 0 0
  ```

```
file /lib64/libmyodbl-service.so
opt/intel/mic/myo/lib/libmyodbl-service.so 755 0 0
```

- When debugging heterogeneous applications on the command line, the offload process is started as root. Using `idbc_mic` with a different user id than root will cause the offload process to not be visible by the remote debug server `idbserver_mic`. The workaround is to launch the command line debugger `idbc_mic` as root. Alternatively the options `-mpm-launch=1 -mpm-cardid=<card-id>` can be added to the default launch options: `idbc_mic -mpm-launch=1 -mpm-cardid=<card-id>  -tco -rconnect=tcpip:<cardip>:<port>`
- When debugging heterogeneous applications from Eclipse*, you may get the error "offload error: cannot start process on the device 0 (error code 1)" when creating the offload process. To work around this, restart the debugging session until you get a session that creates the offload process successfully.

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 debugee 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 ABR debugger 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 debugee 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 now contains Intel® C++ Compiler XE 13.0. The following features are new or significantly enhanced in this version. For more information on these features, please refer to the documentation.

- Support for Intel® MIC Architecture, both via offload (pragmas/keywords) and native (with –mmic) compilation
    - Language Extensions for Offload
    - Offload Extensions for Data Marshaling (Non-shared memory) method
    - Offload Extensions for Virtual-Shared memory method
- Improved support for 3rd Generation Intel® Core™ processor family (-xCORE-AVX-I and –axCORE-AVX-I) and future Intel processors supporting Intel® Advanced Vector Extensions 2 (Intel® AVX2) (-xCORE-AVX2 and –axCORE-AVX2)
- Features from C++11 (-std=c++0x)

    - Additional type traits
    - Uniform initialization
    - Generalized constant expressions (partial support)
    - noexcept
    - Range based for loops
    - Conversions of lambdas to function pointers
    - Implicit move constructors and move assignment operators
    - Support for C++11 features in gcc 4.6 and 4.7 headers
- Out-of-bounds memory checking (-check-pointers)
- More options for controlling the analysis for Static Analysis (-diag-enable sc-{full|concise|precise})

4.2.1 Support for Upcoming OpenMP* SIMD and accelerator features added in Composer XE 2013 Update 2

Composer XE 2013 Update 2 adds support for certain preliminary OpenMP* features. One, support for the SIMD pragmas, directives, and clauses as defined in the OpenMP* 4.0 Public Review Release Candidate 1 specification available from http://openmp.org, is now supported for Linux*. Another, support for directives for attached accelerators as defined in the TR1 - Technical Report on Directives for Attached Accelerators available from http://openmp.org, is also now supported for Linux*. For more information, see http://intel.ly/W7CHjb.

4.2.2 KMP_PLACE_THREADS Environment Variable added in Composer XE 2013 Update 2

This environment variable allows the user to simplify the specification of the number of cores and threads per core used by an OpenMP* application, as an alternative to writing explicit affinity settings or a process affinity mask.

Syntax
```
value = ( int [ "C" | "T" ] [ delim ] | delim ) [ int [ "T" ]
[ delim ] ] [ int [ "O" ] ];
delim = "," | "x";
```

Effect
Specifies the number of cores, with optional offset value and number of threads per core to use. The "C" indicates cores, "T" indicates threads, "O" is used to specify an offset. Either cores or threads should be specified. If omitted, the default value is the available number of cores (threads).

Examples
`5C,3T,1O` - use 5 cores with offset 1, 3 threads per core
`5,3,1` - same as above
`24` - use first 24 cores, all available threads per core
`2T` - use all cores, 2 threads per core
`,2` - same as above
`3x2` - use 3 cores, 2 threads per core
`4C12O` - use 4 cores with offset 12, all available threads per core

4.2.3 New __INTEL_PRE_CFLAGS and __INTEL_POST_CFLAGS environment variables added in Composer XE 2013 Update 2

Composer XE 2013 Update 2 adds a facility to specify Intel compiler command line options via host system environment variables. This is an extension to the facility already provided in the compiler configuration files icc.cfg and icpc.cfg. Command line options can be inserted in the "prefix" position using the `__INTEL_PRE_CFLAGS` environment variable or in the postfix position using the `__INTEL_POST_CFLAGS` environment variable. The command line will be built as follows:

```
icc <PRE flags> <flags from configuration file> <flags from the
compiler invocation> <POST flags>
```

4.2.4 Inline assembly and intrinsic support for Intel architecture code named Broadwell added to Composer XE 2013 Update 1

Some new instructions have been added in the upcoming Intel architecture code named Broadwell. Composer XE 2013 Update 1 has added inline assembly and intrinsic support for these instructions. Intrinsics are defined in `immintrin.h`.

```
extern int _rdseed16_step(unsigned short *random_val);
extern int _rdseed32_step(unsigned int *random_val);
extern int _rdseed64_step(unsigned __int64 *random_val);
```

These intrinsics generate random numbers of 16/32/64 bit wide random integers. These intrinsics are mapped to the hardware instruction `RDSEED`. The generated random value is written to the given memory location and the success status is returned - 1if the hardware returned a valid random value, and 0 otherwise.

The difference between `rdseed` and `rdrand` intrinsics is that `rdseed` intrinsics meet the NIST SP 800-90B and NIST SP 800-90C standards, while the `rdrand` meets the NIST SP 800-90A standard.

```
extern unsigned char _addcarry_u32(unsigned char c_in, unsigned int
src1, unsigned int src2, unsigned int *sum_out);
```

```
extern unsigned char _addcarry_u64(unsigned char c_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *sum_out);
```

The intrinsic computes the sum of two 32/64 bit wide integer values (`src1, src2`) and a carry-in value. The carry-in value is considered 1 for any non-zero `c_in` input value or 0 otherwise. The sum is stored to a memory location referenced by `sum_out` argument:

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

The intrinsic does not perform validness check of a memory address pointed by `sum_out` thus it cannot be used to find out if a sum produces carry-out without storing result of the sum. The return value of the intrinsic is a carry-out value generated by sum. The sum result is stored into memory location pointed by `sum_out` argument.

```
extern unsigned char _subborrow_u32(unsigned char b_in, unsigned int
src1, unsigned int src2, unsigned int *diff_out);
```

```
extern unsigned char _subborrow_u64(unsigned char b_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *diff_out);
```

The intrinsic computes the sum of a 32/64 bit wide unsigned integer value `src2` and a borrow-in value and then subtracts the result of the sum from the 32/64 bit wide unsigned integer value `src1`. The borrow-in value is considered 1 for any non-zero `b_in` input value or 0 otherwise. The difference is then stored to a memory location referenced by `diff_out` argument:

```
*diff_out = src1 + (src2 + (b_in !=0 ? 1 : 0))
```

The intrinsic does not perform validness check of a memory address pointed by `diff_out` thus it cannot be used to find out if a subtraction produces borrow-out without storing the result of the subtraction. The return value of the intrinsic is a borrow-out value generated by subtraction. The result of the subtraction is stored into memory location pointed by the `diff_out` argument.

```
extern unsigned char _addcarryx_u32(unsigned char c_in, unsigned int
src1,   unsigned int src2, unsigned int *sum_out);
```

```
extern unsigned char _addcarryx_u64(unsigned char c_in, unsigned
__int64 src1,   unsigned __int64 src2, unsigned __int64 *sum_out);
```

The intrinsic computes sum of two 32/64 bit wide integer values (`src1, src2`) and a carry-in value. The carry-in value is considered 1 for any non-zero `c_in` input value or 0 otherwise. The sum is stored to a memory location referenced by `sum_out` argument:

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

The intrinsic does not perform validness check of a memory address pointed by `sum_out` thus it cannot be used to find out if a sum produces carry-out without storing the result of the sum. The intrinsic is translated to either a `ADCX` or `ADOX` instruction depending on compiler's decision. By their design these instructions allow running of two interleaved add-with-carry instruction sequences in parallel via using `ADCX` and `ADOX` instructions for these sequences respectively. The return value of the intrinsic is the carry-out value generated by the sum. The sum result is stored into memory location pointed by the `sum_out` argument.

New `_MM_HINT_ET0` hint to `_mm_prefetch` instrinsic

The `_MM_HINT_ET0` hint makes the intrinsic being lowered to the instruction `PREFETCHW` which is supported by the Intel architecture code name Broadwell. Check if the target CPU supports the instruction `PREFETCHW` before using `_MM_HINT_ET0`.

4.2.5 core_4th_gen_avx added for manual cpu dispatch in Composer XE 2013 Update 1

The cpuid "`core_4th_gen_avx`" is now supported for use with the `cpu_dispatch` and `cpu_specific` manual cpu dispatch mechanisms. This cpuid targets processors that support Intel® Advanced Vector Extensions 2 (Intel® AVX2).

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.

4.2.6.1 The command line utility "inspxe-runsc.exe" changed since 2011 update 2

This utility is distributed with Intel® Composer XE 2011 and has been changed since 2011 update 2. This change only affects users who use Composer XE 2011 to perform Static Analysis. Those that do not use Static Analysis and those that perform Static Analysis without using this utility are unaffected. Static Analysis is only available to users of Intel® Parallel Studio XE, Intel® C++ Studio XE, or Intel® Fortran Studio XE versions 2011 or 2013, so users who do not have those products are unaffected.

Inspxe-runsc executes a **build specification,** a description of how an application is built. Usually build specification files are generated by observing a build as it executes and recoding the compilations and links that are performed. Inspxe-runsc repeats these actions using the Intel compiler in Static Analysis mode. Static Analysis results are generated at the link step so a build specification that describes a build with more than one link step will generate more than one Static Analysis result when inspxe-runsc is invoked.

The versions of inspxe-runsc included in Composer XE 2011 and Composer XE 2011 Update 1 generate all the Static Analysis results in a single directory. In the multiple link case this violated the rule that all the Static Analysis results for one and only one project must be created in the same directory. The updated version of inspxe-runsc respects this rule by generating results for each link step in a separate directory. The name of that directory is formed from the name of the file being linked. Thus if a build specification describes a project that builds two executables, file1.exe and file2.exe, then earlier versions of inspxe-runsc would create two results, one for file1 and one for file2, say r000sc and r001sc, in the same directory. The new version of inspxe-runsc will also create two results, but the one for file1 will be created in "My Inspector XE results – file1\r000sc" and the one for file2 will be created in "My Inspector XE results – file2\r000sc". The directories containing the results are both created in the same parent directory.

Inspxe-runsc has a command line switch, -result-dir (-r), that specifies where results are to be created. The meaning of this switch has changed. Previous this would name the directory where the result itself, say r000sc, would be created. Now it names the parent directory where the “My Inspector XE Results - name” directory or directories will be created. So the directory named in the –r switch is effectively two levels up from the results themselves.

The change to inspxe-runsc effectively moves the result directory, and user action is required to adapt to this change. Those using scripts that invoke inspxe-runsc with the –r switch must update their scripts to reflect the new interpretation of the –r switch argument described earlier. Users must move their old result files into the new directory so that Static Analysis results produced by earlier versions of inspxe-runsc share the same directory as results produced by the new version of inspxe-runsc. Users that had been using inspxe-runsc with a build specification with only one link step should move their old results into a directory of the form “My Inspector XE results – name”. If this is not done, then all the problems in the newly created result will appear to be “New”. Users that had been using inspxe-runsc with a build specification with multiple link steps have been having various issues with Static Analysis that will be resolved by using the new utility. Such users are best advised to copy the most recent into their old results into each of the new “My Inspector XE results – name” directories. This offers the best chance that some old problem state information will be correctly applied to new results when they are created in the future.

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

- -vec-report6, -vec-report7
- -W[no-]pch-messages
- -f[no-]defer-pop
- -f[no-]optimize-sibling-calls
- -mmic
- -fextend-arguments=[32|64]
- -guide-profile=<file|dir>[,[file|dir],…]
- -openmp-link <library>
- -opt-prefetch-distance=N[,N]
- -debug [no]pubnames
- -grecord-gcc-switches
- -fno-merge-constants
- -check-pointers=<arg>
- -check-pointers-dangling=<arg>
- -std=c++11 (same as –std=c++0x)
- -[no-]check-pointers-undimensioned

- -no-]check-uninit functionality expanded to –check=<keyword>[,<keyword>…]. Use –check:[no]uninit for original functionality.
- -w3
- -W[no-]unused-parameter
- -W[no-]invalid-pch
- -noerror-limit removed
- -diag-enable sc-{full|concise|precise}
- -diag-enable sc-single-file
- -diag-enable sc-enums
- -watch=<keyword>
- -nowatch
- -offload-attribute-target=<name>
- -offload-option,<target>,<tool>, “option list”
- -no-offload
- -fimf-domain-exclusion=classlist[:funclist]
- -ipp-link={static|dynamic|static_thread}
- -fms-dialect=11
- -static-libstdc++
- -[no-]pie
- -opt-streaming-cache-evict=[0|1|2|3]
- -opt-threads-per-core=[1|2|3|4]
- -[no-]opt-assume-safe-padding

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

4.3.2 -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.

4.3.3 -W[no-]pch-messages added to Composer XE 2013 Update 2

Functionality to enable or disable diagnostics related to pre-compiled headers has been added to update 2.

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

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

4.3.5 –check-pointers=w added to Composer XE 2013 Update 1

Functionality has been added in update 1 to perform write-only error checking in Pointer Checker.

4.3.6 –opt-assume-safe-padding, –opt-streaming-cache-evict and –opt-threads-per-core added in Composer XE 2013 Update 1

For more information, look in the section on Intel® Many Integrated Core architecture here.

4.3.7 -ipp-link option

This option is used with -ipp to indicate which variant of the Intel® Integrated Performance Primitives libraries should be used. There are three options, static to link against the static single-threaded libraries, dynamic to link against the dynamic libraries, or static-thread to link against the static multithreaded libraries. Note that the static multithreaded libraries are only available in a separate package.

4.3.8 –fimf-domain-exclusion

For more information, look in the section on Intel® Many Integrated Core architecture here.

4.4 Other Changes

4.4.1 Support for Microsoft loop pragma syntax added to Composer XE 2013 Update 1

Support for the Microsoft Visual C++ 2012* compiler's `#pragma loop [hint_parallel(n),no_vector,ivdep]` is added for Composer XE 2013 Update 1.

4.4.2 New Warning Level –w3 and Changes to Warning Levels in Composer XE 2013

Here's the new warning levels as listed in "`icc -help`":

```
-w<n>     control diagnostics
        n = 0    enable errors only (same as -w)
        n = 1    enable warnings and errors (DEFAULT)
        n = 2    enable verbose warnings, warnings and errors
        n = 3    enable remarks, verbose warnings, warnings and errors
```

Previously, remarks were listed under –w2. This has been changed so that remarks are now enabled under the new warning level –w3.

4.4.3 Binary compatibility change with __regcall functions and elemental functions (i.e. __declspec(vector))

Intel® C++ Composer XE 2013 introduces an incompatibility with previous compiler versions in the way the __regcall calling convention is handled. Starting from this version, the RBX register is considered to be a callee-save register for __regcall routines, whereas in previous versions it was true only for IA32 targets. This could cause run-time fails if binaries built with different versions of the compilers are used together, so the compiler changes the name decoration scheme for __regcall routines using a new __regcall2__ prefix for mangling __regcall routines (previously the prefix was __regcall__). Binaries built with different versions of the compiler will therefore not link together successfully.

If you have functions declared with the __regcall interface or with the __declspec(vector) elemental function interface, code with these functions built with C++ Composer XE 2013 will not link with code with these functions built with earlier compilers. If you use these declarations, make sure to rebuild all necessary code with C++ Composer XE 2013.

4.4.4 New libirng library for vectorizing random number generator functions added to Composer XE 2013

The compiler can now automatically vectorize the drand48 family of random number generator functions provided by the C standard library. A new library, libirng.a and libirng.so, has been added to implement this support.

4.4.5 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.

4.4.6 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`

4.4.7 Intel® Cilk™ Plus "scalar" Clause removed

The "scalar" clause used optionally with Intel® Cilk™ Plus elemental functions is removed in this release. Please use the functionally equivalent "uniform" clause instead.

4.4.8 Intel® Cilk™ Plus Array Notations Semantics Change in 2011 update 6

In Intel® C++ Composer XE 2011, an Intel® Cilk™ Plus array section assignment like the following:

```
a[:] = b[:] + c[:];
```

could potentially generate temporary copies of the results, impacting performance.

Starting in Intel® C++ Composer XE 2011 update 6, if an array section on the right hand side of an assignment (in the example given, b[:] or c[:]) partially overlaps the array section on the left hand side (in the example given, a[:]) in memory, this assignment will be undefined, and it is up to the programmer to assure that there is no partial overlap in memory on assignments in order to get defined behavior.

An exception to this is if the array sections completely overlap, for example:

```
a[:] = a[:] + 3;
```

Since array a overlaps itself completely, this summation will work as expected.

4.5 Known Issues

4.5.1 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.

4.5.2 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

4.5.3 Static Analysis Known Issues

4.5.3.1 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 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`.

5.1 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

5.2 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
```

5.3 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>`

5.4 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.

5.5 Debugger Features

5.5.1 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.

5.5.2 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.

5.6 Known Issues and Changes

5.6.1 Default .gdbinit script on Pardus* systems may cause the Debugger crash

If you encounter a debugger crash when starting `idbc` or `idb`, you may add the option `-nx` to bypass the default `.gdbinit` script.

5.6.2 No thread info available on Pardus* systems

Due to an issue with the default `libthread_db.so` library on Pardus* systems, the debugger cannot detect thread info when debugging multithreaded applications.

5.6.3 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.

5.6.4 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.

5.6.5 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.

5.6.6 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.

5.6.7 Watchpoints now using processor debug registers (hardware based) in Composer XE 2011 Update 6

Switching from the Intel® Composer XE 2011 (IDB 12.0) to Composer XE 2011 Update 6 (IDB 12.1) debugger, watchpoint support is now entirely supported by using the processor debug registers. Hence their possible configurations are specified by the underlying processor architecture. 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.

5.6.8 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.

5.6.9 Command line parameter –parallel not supported

The debugger command line parameter –parallel is not supported on the shell command prompt nor on the Console Window of the Debugger GUI.

5.6.10 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.

5.6.11 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.

5.6.12 `$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.

5.6.13 `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.

5.6.14 `$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
```

5.6.15 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.

5.6.16 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.

5.6.17 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.

5.6.18 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.

5.6.19 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.

5.6.20 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
```

5.6.21 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

6 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.

6.1 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.

6.1.1 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.

6.2 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 13.0 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 13.0 User and Reference Guides`.

6.2.1 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.

6.3 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.

6.3.1 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.

6.4 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
```

6.5 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.

6.6 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, and 13.0. On Intel® 64 architecture systems, only compiler versions 11.0, 11.1, 12.0, 12.1, and 13.0 are supported.

7 Intel® Integrated Performance Primitives

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

7.1 Intel® IPP static threaded Libraries are Available as a Separate Download

If you require the static threaded version of the Intel® IPP libraries, they are no longer provided in the default Composer XE package. There should be a separate package available from the same area where you downloaded the Composer XE package that contains these libraries.

7.2 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

7.3 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.

8 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

8.1 Notices

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

- Removed Intel MKL GNU Multiple Precision* (GMP) function interfaces
- Disabled timing function mkl_set_cpu_frequency() to perform useful work — use mkl_get_max_cpu_frequency(), mkl_get_clocks_frequency(), and mkl_get_cpu_frequency() as described in the Intel MKL Reference Manual
- Removed MKL_PARDISO constant — used MKL_DOMAIN_PARDISO to specify the PARDISO domain with the mkl_domain_set_num_threads() function
- Removed special backward compatibility functions for convolution and correlation functions in Intel MKL 10.2 update 4
- Removed support for Intel® Pentium® III processor. The minimal supported instruction set will be Intel® Streaming SIMD Extensions 2 (Intel® SSE2).
- Documentation:
  - The Intel MKL Reference Manual in HTML format is no longer available with the product
  - Man pages and Eclipse* help integration are no longer provided

8.2 Changes in This Version

8.2.1 What's New in Intel® MKL 11.0 Update 2

- Introduced Intel MKL Extended Eigensolver:

  Intel MKL Extended Eigensolver is a high performance package for solving symmetric standard or generalized symmetric-definite eigenvalue problems on matrices in dense, LAPACK banded, and sparse (CSR) formats. It is based on an innovative fast and stable numerical algorithm named Feast (See Attributions section below)
- BLAS:
  - Optimized [C/Z]HERK for native execution on the Intel® Many Integrated Core Architecture (Intel® MIC Architecture)
  - Optimized BLAS Level-3 subroutine, xSYMM (all precisions) for automatic offload (AO) on Intel MIC Architecture
- Sparse BLAS:
  - Improved performance of 0-based DCSRMM significantly
- LAPACK:
  - Improved performance of parallel versions of x (OR/UN)(M/G)(LQ/QL/QR/RQ) functions significantly
  - Optimized LU (xGETRF), Cholesky (xPOTRF), and QR (xGEQRF) factorization functions for automatic offload on Intel MIC Architecture
  - Improved LU and SMP Linpack performance for 60-cores on Intel MIC Architecture

- ScaLAPACK:
    - Updated version to 2.0.2. New functions introduced include:
        - PxHSEQR: Nonsymmetric Eigenvalue Problem
        - PxSYEVR/PxHEEVR: MRRR (Multiple Relatively Robust Representations) algorithm
- FFT:
    - Improved performance of complex-to-complex power-of-2 1D and 2D FFTs on Intel MIC Architecture
    - Improved performance of real-to-complex power-of-two and odd size 1D FFTs on Intel MIC Architecture
    - Decreased DFTI descriptor commit time on Intel MIC Architecture
    - Added FFTW interface wrapper libraries support for Intel MIC Architecture
- VSL:
    - Supported ICDF (Inverse cumulative distribution function) method in VSL Lognormal RNG
    - Added "const" specifier to declarations of Summary Statistics functions
    - Improved performance of Wichmann-Hill BRNG on Intel MIC Architecture
- Data Fitting:
    - Improved performance of df[d/s]Interpolate1D, df[d/s]InterpolateEx1D, df[d/s]SearchCells1D, df[d/s]SearchCellsEx1D routines for default/quasi-uniform partition, sorted interpolation sites in scalar (number of interpolation sites is 1) and vector cases for Intel® Xeon® processor X5570 and Intel® Xeon® processor E5-2600
    - Supported DF_DISABLE_CHECK_FLAG parameter in dfiEditVal editor to improve performance for small number of interpolation sites (fewer than one dozen) by disabling checking of the correctness of parameters in Data Fitting routines
    - Added "const" specifier to declarations of functions
- Transposition:
    - Parallelized general out-of-place matrix transposition (mkl_?omatcopy2), improving its performance significantly
- Service functions:
    - Added mkl_peak_mem_usage function which provides information about peak memory amount used by Intel MKL Memory Allocator
    - Added mkl_calloc and mkl_realloc functions extending MKL Memory Allocator functionality to standard C library memory allocation API
- Enhanced SMP LINPACK with residual check:

  It returns error code 1 if a failure is detected and prints conclusion if resulting residuals are ok to pass precision check or not. Please note that residuals might slightly vary from run-to-run on the same matrix if conditional numerical reproducibility mode is not turned on. The check ensures that results are reliable

8.2.2 What's New in Intel® MKL 11.0 Update 1

- BLAS:
    - Optimized [S/D/C/Z]SYMM for native execution on the Intel® Many Integrated Core Architecture (Intel® MIC Architecture)
    - Improved DGEMM and double-precision Level 3 BLAS performance on AMD* Family "Bulldozer" CPUs

- Sparse BLAS: Greatly improved CSRMV performance for complex conjugate transpose & Hermitian cases on Intel® MIC Architecture
- LAPACK:
  - Optimized ?(SY/HE)TRD, ?(OR/UN)M(LQ/QL/QR/RQ), ?(OR\UN)GQR,?GE(QR/LQ/RQ/QL)F functions for native performance on Intel® MIC Architecture
  - Improved ?GETRF and SMP LINPACK benchmark native performance on Intel® MIC Architecture
  - Optimized ?GETRF, ?GEQRF, ?PORTF functions for automatic offload on the Intel® MIC Architecture
- PARDISO: Imaginary part of the diagonal values for Hermitian matrices are ignored
- Cluster FFT:
  - Improved hybrid Cluster FFT (MPI + OpenMP*) performance up to 2 times
  - A new Cluster FFT algorithm (Segment of Interest FFT) that uses less communication was implemented for 1D FFTs and it can be enabled by setting the environment variable "MKL_CFFT_SOI_ENABLE" to "YES" or "1" — see more info in the Intel® MKL documentation
- VSL:
  - Added support of VSL_SS_METHOD_FAST_USER_MEAN method for computation of descriptive Summary Statistics estimates given user-provided mean
  - Improved performance of VSL_SS_METHOD_FAST method for computations of descriptive Summary Statistics estimates on Intel® Xeon® processor E5-2690 CPU
  - Improved performance of Summary Statistics algorithms for computation of raw and central moments, and variance-covariance estimates on Intel® MIC Architecture
  - Improved performance of MT2203 and WH BRNGs on Intel® MIC Architecture
- Transposition: Improved performance of Out-of-place transposition on 2nd generation Intel® Core™ microarchitecture (up to 7x)
- Service functions: Removed seven service functions with obsolete names (see more details in this article on obsolete service functions removed)
- Bug fixes

8.2.3 Changes in Initial Release

- Intel MKL now has support for Intel® Xeon Phi™ coprocessor based on the Intel® Many Integrated Core Architecture (Intel® MIC Architecture) on Linux* only. There are three Intel MKL usage models on Intel Xeon Phi coprocessor: automatic offload, compiler assisted offload and native execution. Most of Intel MKL has been ported to run natively on these coprocessors. A smaller number of functions have been optimized to automatically divide their computational work between the host and Intel Xeon Phi coprocessor, a feature called automatic offload (AO). Read the Intel MKL User's Guide for more information. Most standard Intel MKL functions run on Intel Xeon Phi coprocessor except the Poisson library, Iterative sparse solvers, and Trust region solvers.

- Conditional Bitwise Reproducibility (CBWR): New functionality in Intel MKL now allows you to balance performance with reproducible results by allowing greater flexibility in code branch choice and by ensuring algorithms are deterministic. See the Intel MKL User's Guide for more information. Refer to the CBWR Knowledge Base Article for more information.
- Intel MKL also introduces optimizations using the new Intel® Advanced Vector Extensions 2 (AVX2) including the new FMA3 instructions. See the Knowledge Base article on support for Intel® AVX2
- BLAS:
  - Optimized [S/D/C/Z]GEMM, [S/D/C/Z]TRMM, [S/D/C/Z]TRSM, [S/D/C/Z]SYRK, [S/D]GEMV, [S/D]AXPY, [S/D]DOT for native execution and ?TRMM, ?TRSM, ?GEMM functions for automatic offload on the Intel MIC Architecture
  - Improved DSYRK/SSYRK performance for 64-bit programs supporting Intel® Advanced Vector Extensions (Intel® AVX)
- LAPACK:
  - Optimized [S/D]GETRF, [S/D]POTRF, [S/D]GEQRF, [S/D]GELQF, [S/D]GEQLF, and [S/D]GERQF for native execution on Intel MIC Architecture
  - Introduced support for LAPACK version 3.4.1
- FFT :
  - Optimized single- and double-precision real-to-complex and complex-to-complex one-, two-, and three-dimensional fast Fourier transforms for native execution on Intel MIC Architecture
  - Added configuration parameter DFTI_THREAD_LIMIT which limits the number of threads per descriptor
  - Added support for 1D real-to-complex transforms with sizes given by 64-bit prime integers
- VML /VSL:
  - Optimized complex SinCos and CIS functions for native execution on Intel MIC Architecture
  - Optimized MT19937, MT2203, MRG32k3a BRNGs, and discrete Uniform and Geometric RNGs for native execution on Intel MIC Architecture
  - Improved performance of viRngGeometric on Intel® Advanced Vector Extensions (Intel® AVX)
  - Implemented threading in Data Fitting Integrate1d function
- Transposition: Parallelized in-place transposition of square matrices with leading dimensions greater than the matrix size for single and double precisions improving its performance significantly
- Implemented local threading control function (mkl_set_num_threads_local) which increases flexibility in threading control
- The mklvars.* script no longer sets $FPATH in environment and no longer exports internal variable MKL_TARGET_ARCH (this change will not impact users as the Intel compiler no longer requires these variables)

- Link Tool: Added Intel MIC Architecture support
- Link Line Advisor:
    - Added Help-Me functionality for selecting architecture (IA-32/Intel® 64) and interface layer (LP64/ILP64)
    - Added Intel MIC Architecture support

8.3 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.

9 Intel® Threading Building Blocks

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

10 Disclaimer and Legal Information