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

Document number: 321413-005US
23 January 2014

Table of Contents

1 Introduction

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

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

1.1 Change History

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

1.1.1 Changes in Update 2

- OS X* 10.9 supported
- Intel® C++ Compiler XE 14.0.2
- Intel® Math Kernel Library 11.1 update 2
- Intel® Integrated Performance Primitives 8.1
- Intel® Threading Building Blocks 4.2 update 3

- New Intel® Cilk™ Plus STL vector reducer in Intel® C++ Composer XE 2013 SP1 update 2
- KMP_DYNAMIC_MODE Environment Variable Support for “asat” Deprecated
- Corrections to reported problems

1.1.2 Changes in Update 1

- Xcode* 5.0 supported
- Intel® C++ Compiler XE 14.0.1
- Intel® Math Kernel Library 11.1 update 1
- Intel® Integrated Performance Primitives 8.0 update 1
- Intel® Threading Building Blocks 4.2 update 1
- New compiler option /Q[a]xMIC-AVX512(-[a]xMIC-AVX512)
- New intrinsic `_allow_cpu_features`
- New Numeric String Conversion Library: `libistrconv`

1.1.3 Changes since Intel® Composer XE 2013

- Online installation
- Intel® C++ Compiler XE 14.0.0
- GNU* Project Debugger (GDB*)
- -use-clang-env enabled by default
- libc++ now supported
- OS X* 10.7 is no longer supported
- Xcode* 4.4 and 4.5 are no longer supported
- Features from C++11 (-std=c++11)
- Partial OpenMP* 4.0 support
- Intel® Cilk™ Plus changes
- __INTEL_COMPILER_UPDATE predefined macro
- Pointer type alignment qualifiers
- Variable definition attributes to avoid false sharing
- -mtune performance tuning option
- New KMP_PLACE_THREADS environment variable
- New __INTEL_PRE_CFLAGS and __INTEL_POST_CFLAGS environment variables
- New –vec-report7 vectorization report level
- -[no-]pch-messages to enable/disable precompiled header diagnostics
- -openmp-simd option added for controlling the enabling/disabling of specific OpenMP* 4.0 features independently of other OpenMP features
- -xATOM_SSE4.2 option added to support Silvermont microarchitecture
- Intel® Math Kernel Library 11.1
- Intel® Integrated Performance Primitives 8.0 update 1
- Intel® Threading Building Blocks 4.2

1.2 Product Contents

*Intel® C++ Composer XE 2013 SP1 Update 2 for OS X** includes the following components:

- Intel® C++ Compiler XE 14.0.2 for building applications that run on Intel-based Mac systems running the OS X* operating system
- GNU* Project Debugger (GDB*) 7.5
- Intel® Debugger 13.0
- Intel® Integrated Performance Primitives 8.1
- Intel® Math Kernel Library 11.1 update 2
- Intel® Threading Building Blocks 4.2 update 3
- Integration into the Xcode* development environment
- On-disk documentation

1.3 System Requirements

- A 64-bit Intel®-based Apple* Mac* system host (development for 32-bit is still supported)
- 1GB RAM minimum, 2GB RAM recommended
- 3GB free disk space
- One of the following combinations of OS X*, Xcode* and the Xcode SDK:
  - OS X 10.9 and Xcode* 5.0
  - OS X 10.8 and Xcode* 4.6
- If doing command line development, the Command Line Tools component of Xcode* is required

Additional requirements to use GNU GDB*

- To use the provided GNU* GDB, Python* version 2.4, 2.6 or 2.7 is required.

Note: Advanced optimization options or very large programs may require additional resources such as memory or disk space.

1.4 Documentation

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

1.5 Samples

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

1.6 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 will be using Xcode*, please make sure that a supported version of Xcode is installed. If you install a new version of Xcode in the future, you must reinstall the Intel C++ Compiler afterwards.

The Command Line Tools component, required for command-line development, is not installed by default. It can be installed using the Components tab of the Downloads preferences panel.

You will need to have administrative or “sudo” privileges to install, change or uninstall the product.

If you received the compiler product on DVD, insert the DVD. Locate the disk image file (`xxx.dmg`) on the DVD and double-click on it. If you received the compiler product as a download, double-click the downloaded file.

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 Online Installation now available in Intel® Composer XE 2013 SP1

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

2.2 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.

2.3 Using a License or Serial Number from Intel® C++ Compiler 11.1 Professional Edition to Install

Serial numbers and licenses distributed for use with the Intel® C++ Compiler 11.1 Professional Edition will not work with Intel® C++ Composer XE 2013. You can obtain a new upgraded license and serial number for free if your current product is active by doing the following:

1. Login to the Intel® Software Development Products Registration Center at http://registrationcenter.intel.com by entering your Login ID and Password in the Registered Users Login section of the web page. You will find a list of all products you have subscriptions for in the "My Products" page.
2. For the current product, you will see the XE product name displayed in addition to the original product name. Clicking the latest update in the download latest update column leads you to the product upgrade page. Click the product name to upgrade.
3. You can now send yourself an email with the updated license file or use the updated serial number to install the C++ Composer XE 2013 product.

2.4 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.

2.5 Xcode* integration-only installation no longer provided

The C++ Composer XE 2013 installation only allows you to install with command-line tools only or with command-line tools and Xcode* integration. There is no longer an option to only install the integration with Xcode*.

2.6 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.

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

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

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

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

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

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

- `Documentation` – symbolic link to the `Documentation` directory
- `Samples` – symbolic link to the `Samples` directory

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

- `bin` – all executables
- `pkg_bin` – symbolic link to `bin` directory
- `compiler` – shared libraries and header files
- `debugger` – debugger files
- `Documentation` – documentation files
- `man` – symbolic link to the `man` directory
- `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

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 SP1 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.

2.7 Installing Intel® Integrated Performance Primitives Cryptography Libraries

The Intel® Integrated Performance Primitives product provides an optional component containing libraries of cryptography routines. Installation and use of the cryptography libraries requires a separate license that is available at no charge from Intel once your license for Intel Integrated Performance Primitives has been registered. Export restrictions apply. For details, please see http://intel.ly/ndrGnR

2.8 Relocating Product After Install

The Xcode integration is relocatable simply by dragging and dropping the Xcode directory tree to another location. If you wish to use `idb` from a command prompt using a relocated Xcode directory tree, please see http://intel.ly/q3Fl3R for additional steps that are required. Note that `idb` is not available from within the Xcode IDE.

2.9 Removal/Uninstall

It is not possible to remove the compiler while leaving any of the performance library components installed.

1. Open Terminal and set default (`cd`) to any folder outside `<install-dir>`
2. Type the command:
   `<install-dir>/composer_xe_2013_sp1.<n>.<pkg>/uninstall_ccompxe.sh`
3. Follow the prompts

If you are not currently logged in as `root` you will be asked for the `root` password.

3 Intel® C++ Compiler

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

3.1 New and Changed Features

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

- New Intel® Cilk™ Plus STL vector reducer in Intel® C++ Composer XE 2013 SP1 update 2
- New intrinsic _allow_cpu_features in Intel® C++ Composer XE 2013 SP1 update 1
- The `this` pointer is now allowed in the Intel® Cilk™ Plus SIMD-enabled function `uniform` clause (i.e. `__declspec(vector(uniform(this)))`) in Intel® C++ Composer XE 2013 SP1 update 1
- New Numeric String Conversion Library libistrconv in Intel® C++ Composer XE 2013 SP1 update 1
- Features from C++11 (-std=c++11)
  - Complete (instead of partial) implementation of initializer lists. See N2672 and N3217.
  - Complete implementation of inline namespaces. See N2535.
  - Complete implementation of non-static data member initializers. See N2756.
  - Complete implementation of generalized constant expressions. See N2235.
  - Complete implementation of unrestricted unions. See N2544.
  - Delegating constructors. See N1986.
  - Rvalue references for *this. See N2439.
  - Raw string literals. See N2442.
  - Conversions of lambdas to function pointers.
  - Implicit move constructors and assignment operators. See N3053.
  - __bases and __direct_bases type traits.
  - The context-sensitive keyword "final" can now be used on a class definition, and "final" and "override" can be used on member function declarations. See N2928, N3206, and N3272.
  - Complete implementation of the "noexcept" specifier and operator. See N3050. Includes the late instantiation of noexcept per core issue 1330.
- Partial OpenMP* 4.0 RC1 and TR1 support

- Intel® Cilk™ Plus changes
- __INTEL_COMPILER_UPDATE predefined macro
- Pointer type alignment qualifiers
- Variable definition attributes to avoid false sharing
- -mtune performance tuning option

3.1.1 New Intel® Cilk™ Plus STL vector reducer in Intel® C++ Composer XE 2013 SP1 update 2

In update 2, a `reducer_vector` class is now provided. The header file "`cilk/reducer_vector.h`" will need to be included. The reducer type is `cilk::reducer< cilk::op_vector<type> >`. See the header file comments for further specifics.

3.1.2 New intrinsic `_allow_cpu_features` in Intel® C++ Composer XE 2013 SP1 update 1

This new intrinsic `_allow_cpu_features([xxx][,xxx])` is added to `immintrin.h`. It tells the compiler that the code region following it may be targeted for processors with the specified feature(s) so some specific optimizations may be performances.

Note: support of this intrinsic is preliminary, not all of the compilers optimization phases can be taken place in the code region.

Please reference Compiler documentation for detailed information with code sample, and the article New intrinsic _allow_cpu_features support for additional information.

3.1.3 The `this` pointer is now allowed in the Intel® Cilk™ Plus SIMD-enabled function uniform clause (i.e. `__declspec(vector(uniform(this)))`) in Intel® C++ Composer XE 2013 SP1 update 1

When a uniform class object calls a SIMD-enabled class member function, explicitly specifying "`uniform(this)`" clause in the callee's SIMD declaration may improve performance (how much depends on how the "`this`" keyword is used inside the callee). The usage model is the same as the uniform clause applied to formal parameters.

Please reference Compiler documentation for detailed information.

3.1.4 New Numeric String Conversion Library `libistrconv` in Intel® C++ Composer XE 2013 SP1 update 1

This New Numeric String Conversion Library, `libistrconv`, provides a collection of routines for converting between ASCII strings and C data types, which are optimized for performance. The new APIs are declared in the header file "`istrconv.h`".

Please reference Compiler documentation for detailed information.

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

Composer XE 2013 SP1 adds partial support for OpenMP* 4.0 features. The features supported as defined in the OpenMP* 4.0 specifications available from http://openmp.org are:

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

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

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

- SIMD enabled function implementation has changed to be more compatible with other vector function implementations in gcc and OpenMP*. This breaks binary compatibility with previous Intel® C++ Compiler versions (13.1 and earlier). You should either rebuild all codes using SIMD enabled functions with the version 14.0 compiler, or use the –vecabi=legacy compiler option to use the previous implementation.
- New multiply reducer defined in `cilk/reducer_opmul.h`
- Three new array notation reduction intrinsics have been added to support bitwise reduction operations:
    - `__sec_reduce_and`
    - `__sec_reduce_or`
    - `__sec_reduce_xor`

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

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

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

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

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

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

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

3.1.9 New __INTEL_COMPILER_UPDATE predefined macro in Composer XE 2013 SP1

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

3.2 New and Changed Compiler Options

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

3.2.1 New compiler option /Q[a]xMIC-AVX512(-[a]xMIC-AVX512) for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instructions support (Update 1)

This option may generate Intel(R) AVX-512 Foundation instructions, Intel(R) AVX-512 Conflict Detection instructions, Intel(R) AVX-512 Exponential and Reciprocal instructions, Intel(R) AVX-512 Prefetch instructions for Intel(R) processors and the instructions enabled with CORE-AVX2. It lets compiler to optimize for Intel(R) processors that support Intel(R) AVX-512 instructions.

Please see Compiler Options section of the documentation for detailed information, and see AVX-512 instructions article for more about the new instructions.

3.2.2 New and Changed in Composer XE 2013 SP1

- -[no-]openmp-simd
- -xATOM_SSE4.2
- -xATOM_SSSE3
- -mlong-double-64
- -mlong-double-80
- -vecabi=<arg>
- -W[no-]pch-messages
- -mtune=<arch>

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

3.2.3 –[no-]openmp-simd added to Composer XE 2013 SP1

This option allows you to enable/disable the SIMD features of OpenMP* 4.0 independently of support of the rest of OpenMP* (enabled with –openmp). When –openmp is specified, –openmp-simd is set as well, allowing the use of this feature.

3.2.4 –use-clang-env is now enabled by default in Composer XE 2013 SP1

The option –use-clang-env to enable the use of clang headers and libraries is now on by default. To use GNU* headers and libraries, please specify –no-use-clang-env.

3.2.5 –mtune added to Composer XE 2013 SP1

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

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

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

3.3 Other Changes

3.3.1 KMP_DYNAMIC_MODE Environment Variable Support for “asat” Deprecated

Support for “`asat`” (automatic self-allocating threads) by the environment variable `KMP_DYNAMIC_MODE` is now deprecated, and will be removed in a future release.

3.3.2 Environment Setup Script Changed

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

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.

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

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

```
// t.cpp
```

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

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

3.3.4 OpenMP* Legacy Libraries Removed

The OpenMP "legacy" libraries have been removed in this release. Only the "compatibility" libraries are provided.

3.4 Sample Notes

3.4.1 Building Tachyon

There are a couple common problems that may come up in the course of building Tachyon. If you use the provided Makefile to do a command-line build of the Tachyon sample, you may get errors about not finding directories or finding pbxcp. If you get these, go to the file common/gui/makefile.gmake, look for where the two variables `XCODE_SDK_SYSROOT` and `PBXCP` are set and change them to point to the location of your OS X 10.6 SDK and the pbxcp binary respectively.

For building from Xcode*, you may run into problems building the build_with_tbb configuration with llvm gcc*. The problem will be that the libtbb.dylib cannot be found. In this case, go to the `Summary->Linked Frameworks and Libraries` section, and manually add the libtbb.dylib library from the `composer_xe_2013.x.xxx/compiler/lib` directory.

3.5 Known Limitations

3.5.1 "No such file or directory" error when compiling Objective-C files in Xcode* 4.6

When the Intel compiler is selected by adding a build rule for C/C++ sources, Xcode* 4.6 will fail to compile Objective-C sources in targets that contain both C/C++ and Objective-C source files with an error similar to the following:

```
Build AST from tmp-cobjc1/myclass.m/Applications/Xcode-
46.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/
clang -x objective-c <…>  myclass.m -o myclass.ast

clang: error: no such file or directory: ' /tmp/tmp-cobjc1/myclass.m'

clang: error: no input files
```

```
Command /Applications/Xcode-
46.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/
clang failed with exit code 1
```

You can resolve this issue by adding another Build Rule for "*.m" and/or "*.mm" files to be compiled by the Intel C++ Compiler.

3.5.2 "Build" or "Run" in Xcode* 5 do not accurately detect and rebuild modified dependencies

When the Intel® C++ Compiler XE 14.0 is selected for a project in Xcode* 5, there is an issue where if a file that other files/projects depend on is modified, for example a header file, then the dependent files or libraries will not be automatically rebuilt with the "Build" or "Run" commands. Instead, a "Clean" needs to be done first to do a full rebuild. This issue is being investigated.

3.5.3 No support for libc++ in Composer XE 2013, but added in Composer XE 2013 SP1

Support for libc++ has been added to Composer XE 2013 SP1.

3.5.4 Creating new project in Xcode 4.6 or above causes hardcoding of –stdlib=libc++

A new project created in Xcode 4.6 or above causes the hardcoding of a setting for –stdlib=libc++ even for projects that have the Intel® C++ Compiler toolset added. So setting the Intel® C++ Compiler field for the C++ Standard Library setting to libstdc++ is ineffective because libc++ overrides the setting. To change this, do the following:

1. Select the project row in the navigator area at the left of the workspace window
2. In the project editor that appears, select the row that represents the project level of build settings
3. You should see the `C++ Standard Library` setting in bold, indicating that it has a custom value in this project
4. Select that row and press the Delete key to remove the customized value
5. The `C++ Standard Library` build property should now have the value `Compiler Default`

Note that you may have to follow the above steps before adding the Intel® C++ Compiler toolset to your project build rules.

4 GNU* Project Debugger (GDB)

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

4.1 Features

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

- Support for Intel® Transactional Synchronization Extensions (Intel® TSX)

4.2 Pre-requisites

In order to use the provided GNU* GDB Python* version 2.4, 2.6 or 2.7 is required.

4.3 Using GNU* GDB

To use any of the above versions of GNU* GDB source the following script:

```
source <install-dir>/bin/debuggervars.[sh|csh]
```

Please make sure to source the above script always before using the debugger.

To start GNU* GDB provided with Intel® Composer XE use the following command:

```
$ gdb-ia
```

This debugger is designed to debug IA-32 or Intel® 64 applications natively. Its use is no different than with traditional GNU* GDB debuggers. There are some extensions, though, which can be found in the documentation.

4.4 Documentation

The documentation for the provided GNU* GDB can be found here:

```
<install-dir>/Documentation/en_US/debugger/gdb/gdb.pdf
```

5 Intel® Debugger (IDB)

Intel® Debugger (IDB) is available as host debugger for IA-32 and Intel® 64 applications.

5.1 Support Deprecated for Intel® Debugger

In a future major release of the product, the Intel® Debugger may be removed. This impacts all components and features described in this section.
New users should use the GNU* GDB debugger components instead.

5.2 Using Intel® Debugger

To start Intel® Debugger provided with Intel® Composer XE use the following command:

`$ idbc` (command line version)

This debugger is designed to debug IA-32 or Intel® 64 applications natively. Additional documentation can be found Documentation section below.

5.3 Documentation

Documentation for the Intel® Debugger can be found here:

```
<install-dir>/Documentation/en_US/debugger/
```

5.4 Compilation Requirements

5.4.1 Debug information stored in object files

Starting with Xcode 2.3, the Dwarf debugging information is stored in the object (.o) files. These object files are accessed by the debugger to obtain information related to the application being debugged and thus must be available for symbolic debugging.

In cases where a program is compiled and linked in one command, such as:

```
$ icc -g -o hello.exe hello.c
```

The object files are generated by the compiler but deleted before the command completes. The binary file produced by this command will have no debugging information. To make such an application debuggable there are two options:

1. Build the application in two steps, explicitly producing a `.o` file:

   ```
   $ icc -c -g -o hello.o hello.c
   $ icc -g -o hello.exe hello.o
   ```

2. Use the compiler switch `-save-temps` to prevent the compiler from deleting the temporary `.o` files it generates:

   ```
   $ icc -g -save-temps -o hello.exe hello.c
   ```

   The debugger does not use the output of the “dsymutil” utility.

5.4.2 Compilation requirements for debugging on OS X* 10.7 and higher (64-bit only)

Starting with OS X* 10.7 built 64-bit executables default to Position Independent Executable (PIE) code. However, the Intel® Debugger does not support debugging 64-bit executables built with PIE. To disable PIE, add `-Wl,-no_pie` to the compiler invocation. If the GNU* linker is used directly, use option `-no_pie`.

If in Xcode*, select “Don’t Create Position Independent Executables” under Build Settings. Note that the `-g` (and optionally `-save-temps` to save your object files) options are also required to build debuggable applications on all OS X* versions.

5.5 Known Issues

5.5.1 Dwarf vs. Stabs Debug Formats

The debugger only supports debugging of executables whose debug information is in Dwarf2 format, and does not support the Stabs debug format. Use the `-gdwarf-2` flag on the compile command to have gcc and g++ generate Dwarf output. The Intel compilers (icc and ifort) produce Dwarf2 debug format with the `-g` flag.

5.5.2 Debug Info from Shared Libraries

The debugger does not read debug information from shared libraries. Therefore you cannot set a breakpoint to symbols like _exit which are part of a system library.

5.5.3 Non-local Binary and Source File Access

The debugger cannot access binary files from a network-mounted file system (such as NFS). The error message will look like this:

```
Internal error: cannot create absolute path for: /home/me/hello
You cannot debug "/home/me/hello" because its type is "unknown".
```

The debugger cannot access source files from a network-mounted file system (such as NFS). The error message will look like this:

```
Source file not found or not readable, tried...
./hello.c
/auto/mount/site/foo/usr1/user_me/c_code/hello.c
(Cannot find source file hello.c)
```

The file-path specified will be correct.

The workaround in both cases is to copy the files to a local file system (i.e., one which is not mounted over the network).

5.5.4 Debugging applications that fork

Debugging the child process of an application that calls fork is not yet supported.

5.5.5 Debugging applications that exec

The $catchexecs control variable is not supported.

5.5.6 Snapshots

Snapshots are not yet supported as described in the manual.

5.5.7 Debugging optimized code

Debugging optimized code is not yet fully supported. The debugger may not be able to see some function names, parameters, variables, or the contents of the parameters and variables when code is compiled with optimizations turned on.

5.5.8 Watchpoints

Watchpoints that are created to detect write access don't trigger when a value identical to the original has been written. These restrictions are due to a limitation in the OS X* operating system.

Because the SIGBUS signal rather than the SIGSEGV signal is used by the debugger to implement watchpoints, you cannot create a signal detector which will catch a SIGBUS signal.

5.5.9 Graphical User Interface (GUI)

This version of the debugger does not support the GUI

5.5.10 MPP Debugging Restrictions

MPP debugging is not supported as described in the manual.

5.5.11 Function Breakpoints

Debugger breakpoints set in functions (using the "stop in" command) may not halt user program execution at the first statement. This is due to insufficient information regarding the function prologue in the generated Dwarf debug information. As a work-around, use the "stop at" command to set a breakpoint on the desired statement.

The compiler generates a call to "__dyld_func_lookup" as part of the prologue for some functions. If you set a breakpoint on this function the debugger will stop there, but local variable values may not be valid. The work-around is to set a breakpoint on the first statement inside the function.

5.5.12 Core File Debugging

Debugging core files is not yet supported.

5.5.13 Universal Binary Support

Debugging of universal binaries is supported. The debugger supports debugging the IA-32 Dwarf sections of binaries on IA-32 and either the IA-32 or the Intel® 64 sections on Intel® 64.

5.5.14 Debugger variable $threadlevel

The manual's discussion of the debugger variable "$threadlevel" says "On OS X*, the debugger supports POSIX threads, also known as pthreads." This sentence might be read as implying that other kinds of threads might be supported. This is not true; only POSIX threads are supported on OS X*.

5.5.15 Open File Descriptors Limitation

Because the debugger opens the .o files of a debuggee to read debug information, you should raise the open file limit.

OS X* limits the number of open file descriptors to 256. You can increase this limit as follows:

```
ulimit -n 2000
```

Please use this command to increase the number of open descriptors before starting the debugger.

This is a workaround until the debugger can better share a limited number of open file descriptors over many files.

5.5.16 `$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 are 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.

5.5.17 `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.

5.5.18 `$stepg0` Default Value Changed

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

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

6 Intel® Integrated Performance Primitives

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

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

For detailed information about IPP see the following links:

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

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

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

6.2 Intel® IPP Code Samples

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

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

7 Intel® Math Kernel Library

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

7.1 Notices

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

- Intel® MKL now provides a choice of components to install. Components necessary for PGI* compiler, Compaq Visual Fortran* Compiler, SP2DP interface, BLAS95 and LAPACK95 interfaces, Cluster support (ScaLAPACK and Cluster DFT) and Intel® Many Integrated Core Architecture (Intel® MIC Architecture) support are not installed unless explicitly selected during installation
- Unaligned Conditional Numerical Reproducibility (CNR) is not available for Intel MKL Cluster components (ScaLAPACK and Cluster DFT)
- Examples for using Intel MKL with Boost* uBLAS and Java* have been removed from product distribution and placed in the following articles:
  - How to use Intel MKL with Java*
  - How to use Boost* uBLAS with Intel MKL
- **Known Issue:** User applications on OS X* linked with the libmkl_rt.so library where the first call to Intel MKL was made in a parallel section will crash with a segmentation fault or with either of these messages:
  `"malloc: *** error for object xxxxx: pointer being freed was not allocated *** set a breakpoint in malloc_error_break to debug"`
  or
  `"malloc: *** error for object xxxxx: double free !!! *** set a breakpoint in malloc_error_break to debug"`
  Workaround: Call any Intel MKL function before the parallel section

7.2 Changes in This Version

7.2.1 What's New in Intel MKL 11.1 update 2

- Introduced support for Intel® Atom™ processors
- BLAS:
  - Improved performance of ?GEMM for m==1 or n==1 on all Intel architectures
  - Improved performance of ?SYMM on Intel MIC Architecture
  - Improved {S/D}GEMM single thread performance on small matrices for 64-bit processors supporting Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Vector Extensions 2 (Intel® AVX2)
  - Improved DGEMV performance for 64-bit processors supporting Intel AVX2
  - Improved threaded performance of {S,D,C,Z}GEMV for notrans:n>>m and trans:m>>n on all Intel architectures
  - Improved DSYR2K performance for 64-bit processors supporting Intel AVX and Intel AVX2

    - Improved DTRMM performance on small matrices (A matrix size <= 10) for 64-bit processors supporting Intel AVX and Intel AVX2
    - Reduced stack usage for ZHEMM and ZSYRK
- LAPACK:
    - Improved performance of (S/D)SYRDB and (D/S)SYEV for large dimensions and UPLO=L when eigenvectors are needed
    - Improved performance of ?GELQF,?GELS and ?GELSS for underdetermined case (M
    - Improved performance of ?GEHRD,?GEEV and ?GEES
- Sparse BLAS:
    - Improved Sparse BLAS level 2 and 3 performance for systems supporting Intel® Streaming SIMD Extensions 4.2 (Intel® SSE4.2), Intel AVX, and Intel AVX2 instruction sets
- PARDISO:
    - Improved memory estimation of out-of-core portion size for reordering algorithm leading to improved factorization-solving step performance in OOC mode
- VML:
    - Added v[d|s]Frac function computing fractional part for each vector element
- VSL RNG:
    - Improved performance of MT2203 BRNG on CPUs supporting Intel AVX and Intel AVX2 instruction sets
- VSL Summary Statistics:
    - Added support for computation of group/pooled (VSL_SS_GROUP_MEAN/VSL_SS_POOLED_MEAN) mean estimates

7.2.2 What's New in Intel MKL 11.1 update 1

- Introduced support for Intel® AVX-512 instructions set with limited set of optimizations
- BLAS:
    - Improved performance of DSDOT, and added support for multiple threads, on all 64-bit Intel processors supporting Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Vector Extensions 2 (Intel® AVX2)
    - Improved handling of denormals on the diagonal in *TRSM
    - Improved SGEMM performance for small N and large M and K on Intel® Many Integrated Core Architecture (Intel® MIC Architecture)
    - Improved parallel performance of *HEMM on all Intel processors supporting Intel® SSE4.2 and later
    - Improved parallel performance of 64-bit *SYRK/*HERK on all Intel processors supporting Intel® SSSE3 and later
    - Improved serial performance of 64-bit {D,S}SYRK on all Intel processors supporting Intel® SSE4.2 and later
    - Improved performance of DTRSM on Intel® MIC Architecture
    - Enhanced Intel® Optimized HPL Benchmark runmultiscript capabilities for Intel processors supporting Intel® AVX
    - Improved Intel® Optimized HPL Benchmark performance on Intel® MIC Architecture
- LAPACK

    - Decreased memory utilization for parallel LAPACK functions (OR/UN)M(QR/RQ/QL/LQ)
    - Decreased stack memory utilization in LAPACK functions
    - Improved performance of (S/D)SYRDB and (S/D)SYEV for large dimensions when eigenvalues are only needed
- ScaLAPACK
    - Updated PBLAS headers to mix default NETLIB and MKL complex datatypes
- DFT: Optimized complex-to-complex and real-to-complex transforms
- Transposition: Improved performance of mkl_?omatcopy routines on tall and skinny matrices
- DFTI interface and FFTW wrappers are now thread safe. setting NUMBER_OF_USER_THREADS parameter when using MKL DFT from parallel regions became optional.

7.2.3 What's New in Intel MKL 11.1

- Conditional Numerical Reproducibility : Introduced support for Conditional Numerical Reproducibility (CNR) mode on unaligned data
- Introduced Clang compiler support on OS X*
- Improved performance of CNR=AUTO mode on recent AMD* systems
- BLAS:
    - Improved performance of [S/D]GEMV on all Intel processors supporting Intel® SSE4.2 and later
    - Optimized [D/Z]GEMM and double-precision Level 3 BLAS functions on Intel® Advanced Vector Extensions 2 (Intel® AVX2)
    - Optimized [Z/C]AXPY and [Z/C]DOT[U/C] on Intel® Advanced Vector Extensions (Intel® AVX) and Intel AVX2
    - Optimized sequential version of DTRMM on Intel MIC Architecture
    - Tuned DAXPY on Intel AVX2
- LAPACK:
    - Improved performance of (S/D)SYRDB and (S/D)SYEV for large dimensions when only eigenvalues are needed
    - Improved performance of xGESVD for small sizes like M,N<10
- VSL:
    - Added support and examples for mean absolute deviation
    - Improved performance of Weibull Random Number Generator (RNG) for alpha=1
    - Added support of raw and central statistical sums up to the 4th order, matrix of cross-products and median absolute deviation
    - Added a VSL example designed by S. Joe and F. Y. Kuo illustrating usage of Sobol QRNG with direction numbers which supports dimensions up to 21,201
    - Improved performance of SFMT19937 Basic Random Number Generator (BRNG) on Intel MIC Architecture
- DFT:
    - Improved performance of double precision complex-to-complex transforms on Intel MIC Architecture
    - Optimized complex-to-complex DFT on Intel AVX2
    - Optimized complex-to-complex 2D DFT on Intel® Xeon processor E5 v2 series

    - Improved performance for workloads specific to GENE application on Intel Xeon E5-series (Intel AVX) and on Intel AVX2
    - Improved documentation data layout for DFTI compute functions
    - Introduced scaling in large real-to-complex FFTs
- Data Fitting:
    - Improved performance of df?Interpolate1D and df?SearchCells1D functions on Intel Xeon processors and Intel MIC Architecture
    - Improved performance of df?construct1d function for linear and Hermite/Bessel/Akima cubic types of splines on Intel MIC Architecture, Intel® Xeon® processor X5570 and Intel® Xeon® processor E5-2690
- Transposition
    - Improved performance of in-place transposition for square matrices
- Examples and tests for using Intel MKL are now packaged as an archive to shorten the installation time

7.3 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.

8 Intel® Threading Building Blocks

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

9 Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

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

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

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

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

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

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

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

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

Please consult the licenses included in the distribution for details.