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

Document number: 321413-004US
14 March 2013

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.

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 3 (2013.3)

- Intel® C++ Compiler XE 13.0.2
- Intel® Math Kernel Library 11.0 Update 3
- Intel® Threading Building Blocks 4.1 Update 3
- Support for Xcode 4.6
- Xcode 4.3 not supported
- Corrections to reported problems

1.1.2 Update 2 (2013.2)

- No change to compiler or debugger
- –gcc-version is deprecated
- Intel® Math Kernel Library 11.0 Update 2
- Intel® Threading Building Blocks 4.1 Update 2
- Corrections to reported problems

1.1.3 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
- Support for Xcode* 4.5
- Inline assembly and intrinsic support for Intel architecture code named Broadwell
- core_4th_gen_avx added for manual cpu dispatch in Composer XE 2013 Update 1
- Microsoft* loop pragma support
- Corrections to reported problems

1.1.4 Changes since Intel® C++ Composer XE 2011

- 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 update to version 4.1
- Compiler compatibility with clang added (-use-clang-env)
- 32-bit Apple* Mac* system hosts no longer supported
- OS X* 10.8 support added
- Xcode 4.3 and 4.4 support added
- Versions of Xcode* prior to 4.3 are no longer supported
- OS X* 10.6 is no longer supported
- 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
- New Warning Level –w3 and Changes to Warning Levels in Composer XE 2013
- Xcode* integration only installation no longer provided

1.2 Product Contents

*Intel® C++ Composer XE 2013 Update 3 for OS X** includes the following components:

- Intel® C++ Compiler XE 13.0.2 for building applications that run on Intel-based Mac systems running the OS X* operating system
- Intel® Debugger 13.0
- Intel® Integrated Performance Primitives 7.1 Update 1
- Intel® Math Kernel Library 11.0 Update 3
- Intel® Threading Building Blocks 4.1 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 10.8 and Xcode* 4.4, 4.5, or 4.6 and SDK 10.8
  - OS X 10.7 and Xcode* 4.4, 4.5, or 4.6 and SDK 10.7
- If doing command line development with Xcode* 4.4 (or later), the Command Line Tools component of Xcode* 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.

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 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 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 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.

2.3 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.

2.4 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*.

2.5 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
- `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
- `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` – 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 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.

2.6 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

2.7 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.

2.8 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.<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 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.

- 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
- Compatibility with the Clang environment using the –use-clang-env option (note there is currently no support for libc++)

3.1.1 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`.

3.1.2 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).

3.1.3 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.

3.1.4 Support for Intel® Advanced Vector Extensions 2 (Intel® AVX2) Instructions in 2011 Update 7

The compiler in Intel® C++ Composer XE 2011 Update 7 supports use of Intel® AVX2 instructions in inline assembly and in intrinsics in immintrin.h.

3.1.5 Intel® Cilk™ Plus fully supported in 2011 Update 6

Intel® Cilk™ Plus language extensions for implementing task parallelism which includes cilk_spawn, cilk_for, cilk_sync and Cilk Plus reducers are now supported on OS X* in Update 6. Please refer to the documentation for details.

3.1.6 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.

3.1.7 -export and –export-dir deprecated starting in 2011 update 4

The two compiler options –export and –export-dir support the C++ template export feature. This feature initially planned for support in the C++0x standard was dropped from this standard. The Intel compiler is deprecating this feature and will remove it in a future release.

3.1.8 Additional Keywords for `-sox` option, default changed in 2011 update 3

The `-sox` option, which adds information to the object and executable file about compiler options used and procedure profiling information, has been enhanced to let the user request that the list of inlined functions be included and to let the user exclude information about procedure profiling.

The syntax for *`-sox`* is now:

```
-[no]sox
-sox=keyword[,keyword]
```

Where *`keyword`* is one of `inline` or `profile`. If `-sox` is specified with no keywords, only the command line options are included – this is a change from previous releases. To maintain the previous behavior, use `-sox=profile`. Multiple `-sox` options may be specified on the command line – if so, they are interpreted in left-to-right order.

3.1.9 Three intrinsics changed in 2011 update 2

Three intrinsics (_rdrand16_step(), _rdrand32_step(), _rdrand64_step()) have been changed in update 2. The documentation has not been updated with these new changes. These intrinsic return a hardware-generated random value and are declared in the “immintrin.h” header file.

These three intrinsics are mapped to a single RDRAND instruction, generate random numbers of 16/32/64 bit wide random integers.

Syntax

1. extern int _rdrand16_step(unsigned short *random_val);
2. extern int _rdrand32_step(unsigned int *random_val);
3. extern int _rdrand64_step(unsigned __int64 *random_val);

Description

The intrinsics perform one attempt to generate a hardware generated random value using the instruction RDRAND. The generated random value is written to the given memory location and the success status is returned: 1 if the hardware returned a valid random value and 0 otherwise.

Return

A hardware-generated 16/32/64 random value.

Constraints

The _rdrand64_step() intrinsic can be used only on systems with the 64-bit registers support.

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 and Changed in Composer XE 2013

- -vec-report6
- -f[no-]defer-pop
- -f[no-]optimize-sibling-calls
- -fextend-arguments=[32|64]
- -guide-profile=<file|dir>[,[file|dir],…]
- -openmp-link <library>
- -debug [no]pubnames
- -std=c++11 (same as –std=c++0x)
- -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
- -watch=<keyword>
- -nowatch
- -use-clang-env
- -clang-name=<name>
- -clangxx-name=<name>

- -static-libstdc++
- -[no_]pie
- -ipp-link={static|dynamic|static_thread}

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

3.2.2 –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.

3.2.3 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.

3.2.4 -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.

3.3 Other Changes

3.3.1 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.

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 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 Mac 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 support for libc++.

The Intel compiler currently does not support the new libc++ library on OS X* (-stdlib=libc++).

4 Intel® Debugger (IDB)

4.1 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 `idb` command line debugger.

4.2 Compilation Requirements

4.2.1 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 users have two options.

Users may 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
```

Alternatively, users may use the compiler switch -save-temps to prevent the compiler from deleting the .o files it generates:

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

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

4.2.2 Compilation requirements for debugging on OS X* 10.7 (64-bit only)

OS X* 10.7 defaults to building 64-bit executables with Position Independent Executable (PIE) code. However, the Intel Debugger (IDB) does not currently support debugging 64-bit executables built with PIE. To disable PIE, add the following options at the end of the command line:

```
-Wl,-no_pie
```

Or 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.

4.3 Known Issues

4.3.1 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.

4.3.2 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.

4.3.3 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).

4.3.4 Debugging applications that fork

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

4.3.5 Debugging applications that exec

The $catchexecs control variable is not supported.

4.3.6 Snapshots

Snapshots are not yet supported as described in the manual.

4.3.7 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.

4.3.8 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.

4.3.9 Graphical User Interface (GUI)

This version of the debugger does not support the GUI

4.3.10 MPP Debugging Restrictions

MPP debugging is not supported as described in the manual.

4.3.11 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.

4.3.12 Core File Debugging

Debugging core files is not yet supported.

4.3.13 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.

4.3.14 Debugger variable $threadlevel

The manual's discussion of the debugger variable "$threadlevel" says "On Mac 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*.

4.3.15 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.

4.3.16 `$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.

4.3.17 `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.

4.3.18 `$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
```

5 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.

5.1 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.

5.2 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

5.3 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.

6 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

6.1 Notices

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

- PGI compilers are not supported for Intel® MKL running on Mac OS X* 10.8
- 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
- Documentation:
    - The Intel MKL Reference Manual in HTML format is no longer available with the product

6.2 Changes in This Version

6.2.1 What's New in Intel® MKL 11.0 Update 3

- BLAS:
    - Improved serial and multithreaded performance of DGEMM on 2nd and 3rd Generation Intel® Core™ microarchitectures
- Linpack:
    - Updated the Intel® Optimized MP LINPACK Benchmark for Clusters package to HPL 2.1
- Sparse BLAS:
    - Improved performance of DCOOMM on Intel® Advanced Vector Extensions 2 (Intel® AVX2)
- LAPACK:
    - Parallelized ?LASET, ?LACPY, ?LANGE, ?LANSY
- FFT:
    - Improved Complex-to-complex power-of-2 FFT performance on Intel AVX2
- VSL:
    - Improved performance of SFMT19937 Basic Random Number Generator (BRNG) on Intel AVX2
- Cluster FFT:
    - Improved hybrid mode (MPI + OpenMP*) Cluster FFT performance
- Data Fitting:

    - Improved performance of df?construct1d function for linear and Hermite/Bessel/Akima cubic types of splines on Intel® Xeon® X5570 and Intel® Xeon® E5-2690 CPUs series
- **Known Issue:** User application 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 parallel section.

6.2.2 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)
- 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
- ScaLAPACK:
    - Updated version to 2.0.2. New functions introduced include:
        - PxHSEQR: Nonsymmetric Eigenvalue Problem
        - PxSYEVR/PxHEEVR: MRRR (Multiple Relatively Robust Representations) algorithm
- VSL:
    - Supported ICDF (Inverse cumulative distribution function) method in VSL Lognormal RNG
    - Added "const" specifier to declarations of Summary Statistics functions
- 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

6.2.3 What’s New in Intel® MKL 11.0 Update 1

- BLAS:
    - Improved DGEMM and double-precision Level 3 BLAS performance on AMD* Family “Bulldozer” CPUs
- 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
- 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

6.2.4 Changes in Initial Release

- 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:
    - Improved DSYRK/SSYRK performance for 64-bit programs supporting Intel® Advanced Vector Extensions (Intel® AVX)
- LAPACK:
    - Introduced support for LAPACK version 3.4.1
- FFT :
    - 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:
    - 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
- Link Line Advisor:
    - Added Help-Me functionality for selecting architecture (IA-32/Intel® 64) and interface layer (LP64/ILP64)

6.3 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.

7 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.

8 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/

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

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