Intel® C++ Compiler Professional Edition 11.1 for Mac OS* X Installation Guide and Release Notes

Document number: 321413-002US
2 September 2009

Table of Contents

1 Introduction

1.1 Change History

This section highlights important changes in product updates. For a list of corrections to reported problems, please read the README.TXT file for each update that is available for download from the Intel® Software Development Products Registration Center.

Update 2

- Added mention of new compiler option `-mkl`
- Corrections to reported problems

Update 1

- Note added about change in behavior of `-O0`
- Corrections to reported problems

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.

1.2 Product Contents

Intel® C++ Compiler Professional Edition 11.1 for Mac OS X* includes the following components:

- Intel® C++ Compilers for building applications that run on Intel-based Mac systems running the Mac OS* X operating system
- Intel® Debugger
- Intel® Integrated Performance Primitives
- Intel® Math Kernel Library
- Intel® Threading Building Blocks
- Integration into the Xcode* development environment
- On-disk documentation

1.3 System Requirements

- An Intel®-based Apple* Mac* system
- 1GB RAM minimum, 2GB RAM recommended
- 3GB free disk space
- Mac OS* X 10.5.6 and Xcode* 3.1.2, or Mac OS* X 10.5.7 and Xcode* 3.1.3, or Mac OS* X 10.5.8 and Xcode* 3.1.3
- gcc* 4

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 Japanese Language Support

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

If you wish to use Japanese-language support on an English-language operating system, or English-language support on a Japanese-language operating system, you will find instructions at http://software.intel.com/en-us/articles/changing-language-setting-to-see-english-on-a-japanese-os-environment-or-vice-versa-on-mac-os-x/

1.6 Technical Support

If you did not register your compiler during installation, please do so at the Intel® Software Development Products Registration Center. 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

If you are installing the product for the first time, please be sure to have the product serial number available as you will be asked for it during installation. A valid license is required for installation and use.

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.

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 (`m_cproc_p_11.1.xxx.dmg`) on the DVD and double-click. If you received the compiler product as a download, double-click the downloaded file, which will have a name of the form `m_cproc_p_11.1.xxx.dmg`.

Follow the prompts to complete installation.

2.1 Installation Folders

The 11.1 product installs into a different arrangement of folders than in previous versions. The new arrangement is shown in the diagram below. Not all folders will be present in a given installation.

- `<root>/intel/Compiler/11.1/`*`xxx`*`/`
  - `bin`
    - `ia32`
    - `intel64`
  - `include`
    - `ia32`
    - `intel64`
  - `lib`
  - `perf_headers`
  - `Frameworks`
    - `ipp`
    - `mkl`
    - `tbb`
  - `Documentation`
  - `man`
  - `Samples`

Where `<root>` is `/opt` by default, *`xxx`* is the three-digit update number and the folders under `bin`, `include` and `lib` are used as follows:

- `ia32`: Compilers that build applications that run on a 32-bit Intel-based Mac OS* X system
- `intel64`: Compilers that build applications that run on a 64-bit Intel-based Mac OS* X system (also referred to as Intel® 64 architecture)

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

2.2 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://software.intel.com/en-us/articles/intel-integrated-performance-primitives-cryptography-library/

2.3 Relocating Product After Install

If you wish to move the installed product's command line interface to a different location on disk, you can do so using a supplied script.

1. Open a terminal window
2. Change directory (cd) to the compiler install folder (for example, `/opt/intel/Compiler/11.1/xxx`)
3. Type the command:
   `./move_cproc.sh <new-install-location>`
   where `<new-install-location>` is the new directory path

This script will move all the files and update symlinks, environment variables and startup scripts as needed. If you have both Intel C++ and Intel Fortran installed in the old path, both products will be moved to the new location.

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://software.intel.com/en-us/articles/running-idb-from-command-line-after-relocating-xcode-environment/ for additional steps that are required. Note that `idb` is not available from within the Xcode IDE.

2.4 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>/uninstall_cproc.sh`
3. Follow the prompts

If you are not currently logged in as `root` you will be asked for the `root` password. If you also have the same-numbered version of Intel® Fortran Compiler installed, it may also be removed.

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

Please refer to the compiler documentation for details

- Additional features from C++ 0x
- C++ lambda functions
- Decimal floating point
- valarray implementation using IPP option
- #pragma vector_nontemporal

- #pragma unroll_and_jam
- Support for OpenMP* 3.0
- The default mode of the C++ compiler now more closely matches the default mode of gcc. Some C99 features, such as mixed declarations and code, may no longer be turned on by default, but can be enabled using -std=c99

3.2 New and Changed Compiler Options

- -mkl[=lib]

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

3.2.1 –O0 no longer implies –mp

In version 11.1, the `-O0` option for disabling optimizations no longer implies `-mp` for maximizing floating-point precision. The `-mp` switch is deprecated, so we recommend using an explicit `-fp-model` option for applications that are sensitive to floating-point precision changes.

3.3 Other Changes

3.3.1 Optimization Reports Disabled by Default

As of version 11.1, the compiler no longer issues, by default, optimization report messages regarding vectorization, automatic parallelization and OpenMP threaded loops. If you wish to see these messages you must request them by specifying `-diag-enable vec`, `-diag-enable par` and/or `-diag-enable openmp`, or by using `-vec-report`, `-par-report` and/or `-openmp-report`.

Also, as of version 11.1, optimization report messages are sent to `stderr` and not `stdout`.

3.3.2 Environment Setup Script Changed

The icc.sh (icc.csh) script, used to set up the command-line build environment, changed in version 11.0. In previous versions, you chose the target platform by selecting either the cc or cce directory root. In version 11.1, there is one version of these scripts and they now take an argument to select the target platform.

The command takes the form:

`source /opt/intel/Compiler/11.1/`*`xxx`*`/bin/iccvars.sh `*`argument`*

Where *`xxx`* is the package identifier and *`argument`* is either `ia32` or `intel64` as described above under Installation Folders. If you have installed the compiler into a different path, make the appropriate adjustments in the command. Establishing the compiler environment also establishes the Intel® Debugger (idb) environment.

3.3.3 OpenMP* Libraries Default to “compat”

In version 10.1, a new set of OpenMP* libraries was added that allowed applications to use OpenMP code from both Intel and gcc* compilers. These “compatibility” libraries can provide higher performance than the older “legacy” libraries. In version 11.x, the compatibility libraries

are used by default for OpenMP applications, equivalent to `-openmp-lib compat`. If you wish to use the older libraries, specify `-openmp-lib legacy`

The “legacy” libraries will be removed in a future release of the Intel compilers.

4 Intel® Debugger (IDB)

4.1 Known Problems

4.1.1 Dwarf vs. Stabs Debug Formats

The debugger only supports debugging of executables whose debug information is in Dwarf 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 Dwarf debug format with the `-g` flag.

4.1.2 Compilation Requirements

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.

4.1.3 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).

4.1.4 Debugging applications that fork

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

4.1.5 Debugging applications that exec

The $catchexecs control variable is not supported.

4.1.6 Snapshots

Snapshots are not yet supported as described in the manual.

4.1.7 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.

4.1.8 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 Mac 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.

4.1.9 Graphical User Interface (GUI)

This version of the debugger does not support the GUI

4.1.10 MPP Debugging Restrictions

MPP debugging is not supported as described in the manual.

4.1.11 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.

4.1.12 Core File Debugging

Debugging core files is not yet supported.

4.1.13 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.

4.1.14 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 Mac OS* X.

4.1.15 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.

Mac OS* 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.

4.1.16 `$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.

4.1.17 `info stack` Usage

The debugger command "info stack" does not currently support negative frame counts in the optional syntax below:

```
  info stack [num]
```

A positive frame count num will print the innermost num frames. A negative or zero count will print no frames rather than the outermost num frames.

4.1.18 `$stepg0` Default Value Changed

The debugger has changed the default value of the debugger variable `$stepg0` from 1 to 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 have compatibility 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 the Intel® Integrated Performance Primitives as part of Intel C++ Compiler Professional Edition.

5.1 New and Changed Features

- Support Intel® Advanced Vector Extensions (Intel® AVX)
- Support Intel® Core™ i7 processor with new optimization and threading control/optimization
- 3D Image Processing: 3D Geometric Transforms, 3D Filters
- New Data Compression Functions APIs
- New Intel IPP Crypto support to RSA_SSA1.5 and RSA_PKCSv1.5
- Unified Image Classes (UIC) to add PNG format support and new features to support DXT1, DXT3, DXT5 texture compression
- Advanced lighting functions including Spherical Harmonic and Perlin Noise generation Functions
- New video coding areas improvement including Scene Analyzer in MPEG-2, Intensity Compensation & Overlap Smoothing in VC1
- Samples for Signal Processing, Image Processing, String Processing and C++, C# language support have been added in \Samples folder. Others can be downloadable by clicking “Free Code Samples” at http://software.intel.com/en-us/intel-ipp
- The deprecated APIs have been added more reference information in reference manual and header files.

5.2 Known Limitations

The context-sensitive help may fail to work for some flavors of the generated functions (with the “ippg” prefix).

5.3 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://software.intel.com/en-us/articles/download-ipp-cryptography-libraries/

5.4 Intel® IPP Code Samples

The Intel® IPP code samples are organized into downloadable packages for Windows*, Linux* and Mac OS* at
http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-code-samples/

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 the Intel® Math Kernel Library as part of Intel C++ Compiler Professional Edition.

6.1 Changes in This Version

6.1.1 New features

- LAPACK 3.2
  - 238 new LAPACK functions
  - Extra Precise Iterative Refinement
  - Non-Negative Diagonals from Householder QR factorization
  - High Performance QR and Householder Reflections on Low-Profile Matrices
  - New fast and accurate Jacobi SVD
  - Routines for Rectangular Full Packed format
  - Pivoted Cholesky
  - Mixed precision iterative refinement (Cholesky)
  - More robust DQDS algorithm
- Introduced implementation of the DZGEMM Extended BLAS function (as described at http://www.netlib.org/blas/blast-forum/). See the description of the `*gemm` family of functions in the BLAS section of the reference manual.
- PARDISO now supports real and complex, single precision data

6.1.2 Usability/Interface improvements

- Sparse matrix format conversion routines which convert between the following formats:
  - CSR (3-array variation) ↔ CSC (3-array variation)
  - CSR (3-array variation) ↔ diagonal format
  - CSR (3-array variation) ↔ skyline
- Fortran95 BLAS and LAPACK compiled module files (.mod) are now included
  - Modules are pre-built with the Intel Fortran Compiler and are located in the include directory (see Intel® MKL User's Guide for full path)
  - Source is still available for use with other compilers
  - Documentation for these interfaces can be found in the Intel® MKL User's Guide
- The FFTW3 interface is now integrated directly into the main libraries

    - Source code is still available to create wrappers for use with compilers not compatible with the default Intel® Fortran compiler convention for name decoration
    - See Appendix G of the Reference Manual for information
- DFTI_DESCRIPTOR_HANDLE now represents a true type name and can now be referenced as a type in user programs
- Added parameter to Jacobi matrix calculation routine in the optimization solver domain to allow access to user data (see the description of the djacobix function in the reference manual for more information)
- Added an interface mapping calls to single precision BLAS functions in Intel® MKL (functions with “s” or “c” initial letter) to 64-bit floating point precision functions has been added on 64-bit architectures (See “sp2dp” in the Intel® MKL User Guide for more information)
- Compatibility libraries (also known as “dummy” libraries) have been removed from this version of the library

6.1.3 Performance improvements

- Further threading in BLAS level 1 and 2 functions for Intel® 64 architecture
    - Level 1 functions (vector-vector): (CS,ZD,S,D)ROT, (C,Z,S,D)COPY, and (C,Z,S,D)SWAP
        - Increase in performance by up to 1.7-4.7 times over version 10.1 Update 1 on 4-core Intel® Core™ i7 processor depending on data location in cache
        - Increase in performance by up to 14-130 times over version 10.1 Update 1 on 24-core Intel® Xeon® processor 7400 series system, depending on data location in cache
    - Level 2 functions (matrix-vector): (C,Z,S,D)TRMV, (S,D)SYMV, (S,D)SYR, and (S,D)SYR2
        - Increase in performance by up to 1.9-2.9 times over version 10.1 Update 1 on 4-core Intel® Core™ i7 processor, depending on data location in cache
        - Increase in performance by up to 16-40 times over version 10.1 Update 1 on 24-core Intel® Xeon® processor 7400 series system, depending on data location in cache
- Introduced recursive algorithm in 32-bit sequential version of DSYRK for up to 20% performance improvement on Intel® Core™ i7 processors and Intel® Xeon® processors in 5300, 5400, and 7400 series.
- Improved LU factorization (DGETRF) by 25% over Intel MKL 10.1 Update 1 for large sizes on the Intel® Xeon® 7460 Processor; small sizes are also dramatically improved
- BLAS *TBMV/*TBSV functions now use level 1 BLAS functions to improve performance by up to 3% on Intel® Core™ i7 processors and up to 10% on Intel® Core™2 processor 5300 and 5400 series.
- Improved threading algorithms to increase DGEMM performance

    - up to 7% improvement on 8 threads and up to 50% on 3,5,7 threads on the Intel® Core™ i7 processor
    - up to 50% improvement on 3 threads on Intel® Xeon® processor 7400 series.
- Threaded 1D complex-to-complex FFTs for non-prime sizes
- New algorithms for 3D complex-to-complex transforms deliver better performance for small sizes (up to 64x64x64) on 1 or 2 threads
- Implemented high-level parallelization of out-of-core (OOC) PARDISO when operating on symmetric positive definite matrices.
- Reduced memory use by PARDISO for both in-core and out-of-core on all matrix types
- PARDISO OOC now uses less than half the memory previously used in Intel MKL 10.1 for real symmetric, complex Hermitian, or complex symmetric matrices
- Parallelized Reordering and Symbolic factorization stage in PARDISO/DSS
- Up to 2 times better performance (30% improvement on average) on Intel® Core® i7 and Intel® Core™2 processors for the following VML functions: v(s,d)Round, v(s,d)Inv, v(s,d)Div, v(s,d)Sqrt, v(s,d)Exp, v(s,d)Ln, v(s,d)Atan, v(s,d)Atan2
- Optimized versions of the following functions available for Intel® Advanced Vector Extensions (Intel® AVX)
    - BLAS: DGEMM
    - FFTs
    - VML: exp, log, and pow
    - See important information in the Intel® MKL User's Guide regarding the `mkl_enable_instructions()` function for access to these functions

6.2 Known Issues

A full list of the known limitations of this release can be found in the Knowledge Base for the Intel® MKL at http://software.intel.com/en-us/articles/known-limitations-in-intel-mkl-10-2

6.3 Notices

The following change is planned for future versions of Intel MKL. Please contact Technical Support if you have concerns:

- Content in the libraries containing `solver` in the filenames will be moved to the core library in a future version of Intel MKL. These `solver` libraries will then be removed.

6.4 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 (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.

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. Some FFT functions in this release of the Intel® MKL DFTI have been generated by the UHFFT software generation system under license from University of Houston. 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

This section summarizes changes, new features and late-breaking news about Intel® Threading Building Blocks (Intel® TBB) as part of Intel® C++ Compiler Professional Edition.

- Unhandled exceptions in the user code executed in the context of TBB algorithms or containers may lead to segmentation faults when Intel(R) C++ Compiler 10.x is used with glibc 2.3.2, 2.3.3, or 2.3.4.
- To allow more accurate results to be obtained with Intel® Thread Checker or Intel® Thread Profiler, download the latest update releases of these products before using them with Intel® Threading Building Blocks.
- If you are using Intel® Threading Building Blocks and OpenMP* constructs mixed together in rapid succession in the same program, and you are using Intel compilers for your OpenMP* code, set KMP_BLOCKTIME to a small value (e.g., 20 milliseconds to improve performance. This setting can also be made within your OpenMP* code via the kmp_set_blocktime() library call. See the compiler OpenMP* documentation for more details on KMP_BLOCKTIME and kmp_set_blocktime().
- In general, non-debug ("release") builds of applications or examples should link against the non-debug versions of the Intel® Threading Building Blocks libraries, and debug builds should link against the debug versions of these libraries. See the Tutorial in the product documentation sub-directory for more details on debug vs. release libraries.

8 Disclaimer and Legal Information