Table of Contents

1 Introduction .......................................................................................................................... 3
  1.1 Change History ................................................................................................................. 3
  1.2 Product Contents ............................................................................................................. 3
  1.3 System Requirements ..................................................................................................... 3
  1.4 Documentation ............................................................................................................... 4
  1.5 Technical Support .......................................................................................................... 4

2 Installation ............................................................................................................................ 5
  2.1 Activation of Purchase after Evaluation Using the Intel Activation Tool .................... 5
  2.2 Using a License Server ................................................................................................... 6
  2.3 Installation Folders .......................................................................................................... 6
  2.4 Relocating Product After Install .................................................................................. 7
  2.5 Removal/Uninstall ......................................................................................................... 7

3 Intel® Fortran Compiler ...................................................................................................... 7
  3.1 Compatibility .................................................................................................................. 7
    3.1.1 Stack Alignment Change for REAL(16) and COMPLEX(16) Datatypes ..................... 8
  3.2 New and Changed Features ............................................................................................ 8
    3.2.1 Features from Fortran 2003 ...................................................................................... 8
    3.2.2 Features from Fortran 2008 ...................................................................................... 8
    3.2.3 Static Security Analysis Feature (formerly Source Checker) Requires Intel® Inspector XE .......................................................................................................................... 9
    3.2.4 Other Changes .......................................................................................................... 9
  3.3 New and Changed Compiler Options ............................................................................... 9
  3.4 Other Changes ................................................................................................................ 10
    3.4.1 Environment Setup Script Changed ....................................................................... 10
3.4.2 RANF Portability Function Is Now an Intrinsic

3.5 Known Issues

3.5.1 Errors on Mac OS X 10.6 “Snow Leopard” When COMMON Block Shares Name With Library Routine

3.6 Fortran 2003 and Fortran 2008 Feature Summary

4 Intel® Debugger (IDB)

4.1 Compilation Requirements

4.2 Known Problems

4.2.1 Dwarf vs. Stabs Debug Formats

4.2.2 Debug Info from Shared Libraries

4.2.3 Non-local Binary and Source File Access

4.2.4 Local variables may not be visible

4.2.5 Printing Fortran REAL*16 variables

4.2.6 Debugging applications that fork

4.2.7 Debugging applications that exec

4.2.8 Fortran alternate entry points

4.2.9 Snapshots

4.2.10 Debugging optimized code

4.2.11 Watchpoints

4.2.12 Fortran modules and commons

4.2.13 Graphical User Interface (GUI)

4.2.14 MPP Debugging Restrictions

4.2.15 Function Breakpoints

4.2.16 Core File Debugging

4.2.17 Universal Binary Support

4.2.18 Debugger variable $threadlevel

4.2.19 Open FileDescriptors Limitation

4.2.20 $cdir, $cwd Directories

4.2.21 info stack Usage

4.2.22 $stepg0 Default Value Changed

5 Intel® Math Kernel Library

5.1 What’s New in Intel® MKL 10.3 Update 1

5.2 What’s New in Intel® MKL 10.3
1 Introduction
This document describes how to install the product, provides a summary of new and changed product features and notes about features and problems not described in the product documentation.

Intel® Fortran Composer XE 2011 is the next release of the product formerly called Intel® Fortran Compiler Professional Edition.

1.1 Change History
This section highlights important changes in product updates.

Update 1 (12.0.1)

- Intel® Math Kernel Library updated to 10.3 Update 1
- Corrections to reported problems

Product Release (12.0.0)

- Initial product release

1.2 Product Contents
Intel® Fortran Composer XE 2011 for Mac OS* X includes the following components:

- Intel® Fortran Compiler XE 12.0.1 for building applications that run on Intel-based Mac* systems running the Mac OS* X operating system
- Intel® Debugger 12.0.1
- Intel® Math Kernel Library 10.3 Update 1
- Integration into the Xcode* development environment (Limited Feature)
- On-disk documentation

1.3 System Requirements

- An Intel®-based Apple* Mac* system
- 1GB RAM minimum, 2GB RAM recommended
- 2GB free disk space
- Mac OS* X 10.5.8 and Xcode* 3.1.4, or Mac OS* X 10.6.5 and Xcode* 3.2.4
- 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.

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® compilers, associated libraries and associated development tools may include or utilize options that optimize for instruction sets that are available in both Intel® and non-Intel microprocessors (for example SIMD instruction sets), but do not optimize equally for non-Intel microprocessors. In addition, certain compiler options for Intel compilers, including some that are not specific to Intel micro-architecture, are reserved for Intel microprocessors. For a detailed description of Intel compiler options, including the instruction sets and specific microprocessors they implicate, please refer to the “Intel® Compiler User and Reference Guides” under “Compiler Options.” Many library routines that are part of Intel® compiler products are more highly optimized for Intel microprocessors than for other microprocessors. While the compilers and libraries in Intel® compiler products offer optimizations for both Intel and Intel-compatible microprocessors, depending on the options you select, your code and other factors, you likely will get extra performance on Intel microprocessors.</td>
</tr>
<tr>
<td>Intel® compilers, associated libraries and associated development tools 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 Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® 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.</td>
</tr>
<tr>
<td>While Intel believes our compilers and libraries are excellent choices to assist in obtaining the best performance on Intel® and non-Intel microprocessors, Intel recommends that you evaluate other compilers and libraries to determine which best meet your requirements. We hope to win your business by striving to offer the best performance of any compiler or library; please let us know if you find we do not.</td>
</tr>
</tbody>
</table>

Notice revision #20101101

1.5 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
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 Fortran 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 (xxx.dmg) on the DVD and double-click. 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 Activation of Purchase after Evaluation Using the Intel Activation Tool
Note for evaluation customers: a new tool Intel Activation Tool “ActivationTool” is included in this product release and installed at /opt/intel/composerxe-2011.x.xxx/Activation directory.

If you installed the product using an Evaluation license or serial number (SN), or using the “Evaluate this product (no serial number required)” option during installation, and then purchased the product, you can activate your purchase using the Intel Activation Tool at /opt/intel/composerxe-2011.x.xxx/Activation/ActivationTool. It will convert your evaluation software to a fully licensed product. To use the tool:

$ /opt/intel/composerxe-2011.x.xxx/Activation/ActivationTool [SN_here]
2.2 Using a License Server

If you have purchased a “floating” license, see http://software.intel.com/en-us/articles/licensing-setting-up-the-client-floating-license/ 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.

2.3 Installation Folders

The compiler installs, by default, under /opt/intel – this is references as <install-dir> in the remainder of this document. You are able to specify a different location. If Xcode integration is installed, a second copy of these files is present under /Developer/opt/intel.

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

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
- mkl – symbolic link to the directory for the latest installed version of Intel® Math Kernel Library
- composerxe – symbolic link to the composerxe-2011 directory
- composerxe-2011 – directory containing symbolic links to subdirectories for the latest installed Intel® Composer XE 2011 compiler release
- composerxe-2011-<n>.<pkg> - physical directory containing files for a specific compiler version. <n> is the update number, and <pkg> is a package build identifier.

Each composerxe-2011 directory contains the following directories that reference the latest installed Intel® Composer XE 2011 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
- mkl – symbolic link to the mkl 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 composerxe-2011-<n>.<pkg> directory contains the following directories that reference a specific update of the Intel® Composer XE 2011 compiler:
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 2011 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.

### 2.4 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://software.intel.com/en-us/articles/running-idb-from-command-line-after-relocating-xcode-environment/](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.

### 2.5 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:
   ```bash
   <install-dir>/compilerpro-12.0.<n>.<pkg>/uninstall_cprof.sh
   ```
3. Follow the prompts

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

### 3 Intel® Fortran Compiler

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

#### 3.1 Compatibility

In general, object code and modules compiled with earlier versions of Intel Fortran Compiler for Mac OS* X may be used in a build with version 12. Exceptions include:
• Sources that use the CLASS keyword to declare polymorphic variables must be recompiled.
• Objects built with the multi-file interprocedural optimization (-ipo) option must be recompiled.
• Objects that use the REAL(16) or REAL*16 datatypes must be recompiled.
• Objects built for the Intel® 64 architecture with a compiler version earlier than 10.0 and that have module variables must be recompiled. If non-Fortran sources reference these variables, the external names may need to be changed to remove an incorrect leading underscore.
• Modules that specified an ATTRIBUTES ALIGN directive and were compiled with versions earlier than 11.0 must be recompiled. The compiler will notify you if this issue is encountered.

3.1.1 Stack Alignment Change for REAL(16) and COMPLEX(16) Datatypes
In previous releases, when a REAL(16) or COMPLEX(16) (REAL*16 or COMPLEX*32) item was passed by value, the stack address was aligned at 4 bytes. For improved performance, the version 12 compiler aligns such items at 16 bytes and expects received arguments to be aligned on 16-byte boundaries.

This change primarily affects compiler-generated calls to library routines that do computations on REAL(16) values, including intrinsics. If you have code compiled with earlier versions and link it with the version 12 libraries, or have an application linked to the shared version of the Intel run-time libraries, it may give incorrect results.

In order to avoid errors, you must recompile all Fortran sources that use the REAL(16) and COMPLEX(16) datatypes.

3.2 New and Changed Features

3.2.1 Features from Fortran 2003
• FINAL subroutines
• GENERIC keyword for type-bound procedures
• A generic interface may have the same name as a derived type
• Bounds specification and bounds remapping list on a pointer assignment

3.2.2 Features from Fortran 2008
• Maximum array rank has been raised to 31 dimensions (Fortran 2008 specifies 15)
• CONTIGUOUS attribute
• MOLD keyword in ALLOCATE
• DO CONCURRENT
• NEWUNIT keyword in OPEN
• G0 and G0.d format edit descriptor
• Unlimited format item repeat count specifier
• A CONTAINS section may be empty
• Intrinsic procedures BESSEL_J0, BESSEL_J1, BESSEL_JN, BESSEL_YN, BGE, BGT, BLE, BLT, DSHIFTL, DSHIFTR, ERF, ERFC, ERFC_SCALED, GAMMA, HYPOT, IALL, IANY, IPARITY, IS_CONTIGUOUS, LEADZ, LOG_GAMMA, MASKL, MASKR, MERGE_BITS, NORM2, PARITY, POCPNT, POPPAR, SHIFTA, SHIFTL, SHIFTR, STORAGE_SIZE, TRAILZ,
• Additions to intrinsic module ISO_FORTRAN_ENV: ATOMIC_INT_KIND, ATOMIC_LOGICAL_KIND, CHARACTER_KINDS, INTEGER_KINDS, INT8, INT16, INT32, INT64, LOGICAL_KINDS, REAL_KINDS, REAL32, REAL64, REAL128

3.2.3 Static Security Analysis Feature (formerly Source Checker) Requires Intel® Inspector XE

The “Source Checker” feature, from compiler version 11.1, has been enhanced and renamed “Static Security Analysis”. The compiler options to enable Static Security Analysis remain the same as in compiler version 11.1 (for example, -Qdiag-enable sc), but the results are now written to a file that is interpreted by Intel® Inspector XE rather than being included in compiler diagnostics output.

3.2.4 Other Changes

• The ability to create a source listing file with identifier cross-reference has been added
• The ability to generate a build dependencies output file has been added
• An option to use math library functions that are faster but return results with less precision or accuracy
• An option to use math library functions that return consistent results across different models and manufacturers of processors

3.3 New and Changed Compiler Options

Please refer to the compiler documentation for details

• -assume [no]fpe_summary
• -assume [no]old_ldout_format
• -fzero-initialized-in-bss
• -fimf-absolute-error
• -fimf-accuracy-bits
• -fimf-arch-consistency
• -fimf-max-error
• -fimf-precision
• -fvar-tracking
• -fvar-tracking-assignments
• -gen-dep
• -gen-depformat
• -guide
• -guide-data-trans
For a list of deprecated compiler options, see the Compiler Options section of the documentation.

3.4 Other Changes

3.4.1 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 `xxx` is the package identifier and `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® C++ Compiler.

3.4.2 RANF Portability Function Is Now an Intrinsic
The RANF function in the portability library is a non-standard random number generator. As of the version 12.0 compiler, RANF is an intrinsic function with a new, higher-performance implementation. If your program has added USE IFPORT to provide access to RANF, no changes will be seen and you will get the older version. If your program does not have USE IFPORT, or you add INTRINSIC RANF, you will get the new version that returns a different sequence, for a given seed, than the older version. The portability subroutine SRAND is still used to set the seed for RANF. Intel recommends use of the standard intrinsic RANDOM_NUMBER, but RANF is provided for compatibility with applications already using it.
3.5 Known Issues

3.5.1 Errors on Mac OS X 10.6 “Snow Leopard” When COMMON Block Shares Name With Library Routine

On Mac OS X 10.6 “Snow Leopard”, Fortran programs that declare COMMON blocks may have unexpected behavior, including “bus error”, if the COMMON block shares a name with a routine in a static library that is linked against. For example, a COMMON block named SEED will cause problems because the Intel Fortran run-time library contains a portability routine named SEED.

Apple has confirmed that this is caused by a defect in the linker supplied with Mac OS 10.6. To follow the issue and its eventual resolution, please refer to Apple “RADAR” issue 7890410 at http://developer.apple.com/ - a free membership in the Apple Mac Developer Program is required. As a workaround, rename the COMMON block so as not to duplicate a name in a static library.

3.6 Fortran 2003 and Fortran 2008 Feature Summary

The Intel Fortran Compiler supports many features that are new in Fortran 2003. Additional Fortran 2003 features will appear in future versions. Fortran 2003 features supported by the current compiler include:

- The Fortran character set has been extended to contain the 8-bit ASCII characters ~ \ [ ] ` ^ { } | # @
- Names of length up to 63 characters
- Statements of up to 256 lines
- Square brackets [ ] are permitted to delimit array constructors instead of (/ /)
- Structure constructors with component names and default initialization
- Array constructors with type and character length specifications
- A named PARAMETER constant may be part of a complex constant
- Enumerators
- Allocatable components of derived types
- Allocatable scalar variables
- Deferred-length character entities
- PUBLIC types with PRIVATE components and PRIVATE types with PUBLIC components
- ERMSG keyword for ALLOCATE and DEALLOCATE
- SOURCE= keyword for ALLOCATE (Polymorphic source not supported)
- Type extension
- CLASS declaration
- Polymorphic entities
- Inheritance association
- Deferred bindings and abstract types
- Type-bound procedures
- TYPE CONTAINS declaration
- ABSTRACT attribute
• DEFERRED attribute
• NON_OVERRIDABLE attribute
• GENERIC keyword for type-bound procedures
• FINAL subroutines
• ASYNCHRONOUS attribute and statement
• BIND(C) attribute and statement
• PROTECTED attribute and statement
• VALUE attribute and statement
• VOLATILE attribute and statement
• INTENT attribute for pointer objects
• Reallocation of allocatable variables on the left hand side of an assignment statement when the right hand side differs in shape or length (requires option -assume realloc_lhs if not deferred-length character)
• Bounds specification and bounds remapping on a pointer assignment
• ASSOCIATE construct
• SELECT TYPE construct
• In all I/O statements, the following numeric values can be of any kind: UNIT=, IOSTAT=
• NAMELIST I/O is permitted on an internal file
• Restrictions on entities in a NAMELIST group are relaxed
• Changes to how IEEE Infinity and NaN are represented in formatted input and output
• FLUSH statement
• WAIT statement
• ACCESS='STREAM' keyword for OPEN
• ASYNCHRONOUS keyword for OPEN and data transfer statements
• ID keyword for INQUIRE and data transfer statements
• POS keyword for data transfer statements
• PENDING keyword for INQUIRE
• The following OPEN numeric values can be of any kind: RECL=
• The following READ and WRITE numeric values can be of any kind: REC=, SIZE=
• The following INQUIRE numeric values can be of any kind: NEXTREC=, NUMBER=, RECL=, SIZE=
• Recursive I/O is allowed in the case where the new I/O being started is internal I/O that does not modify any internal file other than its own
• IEEE Infinities and NaNs are displayed by formatted output as specified by Fortran 2003
• BLANK, DECIMAL, DELIM, ENCODING, IOMSG, PAD, ROUND, SIGN, SIZE I/O keywords
• DC, DP, RD, RC, RN, RP, RU, RZ format edit descriptors
• In an I/O format, the comma after a P edit descriptor is optional when followed by a repeat specifier
• Rename of user-defined operators in USE
• INTRINSIC and NON_INTRINSIC keywords in USE
• IMPORT statement
• Allocatable dummy arguments
• Allocatable function results
• PROCEDURE declaration
• Procedure pointers
• ABSTRACT INTERFACE
• PASS and NOPASS attributes
• The COUNT_RATE argument to the SYSTEM_CLOCK intrinsic may be a REAL of any kind
• Execution of a STOP statement displays a warning if an IEEE floating point exception is signaling
• MAXLOC or MINLOC of a zero-sized array returns zero if the option -assume noold_maxminloc is specified.
• Type inquiry intrinsic functions
• COMMAND_ARGUMENT_COUNT intrinsic
• EXTENDS_TYPE_OF and SAME_TYPE_AS intrinsic functions
• GET_COMMAND intrinsic
• GET_COMMAND_ARGUMENT intrinsic
• GET_ENVIRONMENT_VARIABLE intrinsic
• IS_IOSTAT_END intrinsic
• IS_IOSTAT_EOR intrinsic
• MAX/MIN/MAXVAL/MINVAL/MAXLOC/MINLOC intrinsics allow CHARACTER arguments
• MOVE_ALLOC intrinsic
• NEW_LINE intrinsic
• SELECTED_CHAR_KIND intrinsic
• The following intrinsics take an optional KIND= argument: ACHAR, COUNT, IACHAR, ICHAR, INDEX, LBOUND, LEN, LEN_TRIM, MAXLOC, MINLOC, SCAN, SHAPE, SIZE, UBOUND, VERIFY
• ISO_C_BINDING intrinsic module
• IEEE_EXCEPTIONS, IEEE_ARITHMETIC and IEEE_FEATURES intrinsic modules
• ISO_FORTRAN_ENV intrinsic module

Fortran 2003 features not yet supported include:

• User-defined derived type I/O
• Parameterized derived types
• A polymorphic SOURCE= specifier for ALLOCATE

The Intel® Fortran Compiler also supports some features from the Fortran 2008 standard. Additional features will be supported in future releases. Fortran 2008 features supported by the current version include:

• Maximum array rank has been raised to 31 dimensions (Fortran 2008 specifies 15)
• CONTIGUOUS attribute
• MOLD keyword in ALLOCATE
• DO CONCURRENT
• NEWUNIT keyword in OPEN
• G0 and G0.d format edit descriptor
• Unlimited format item repeat count specifier
• A CONTAINS section may be empty
• Intrinsic procedures BESSEL_J0, BESSEL_J1, BESSEL_JN, BESSEL_YN, BGE, BGT, BLE, BLT, DSHIFTL, DSHIFTR, ERF, ERFC, ERFC_SCALED, GAMMA, HYPOT, IALL, IANY, IPARITY, IS_CONTIGUOUS, LEADZ, LOG_GAMMA, MASKL, MASKR, MERGE_BITS, NORM2, PARITY, POPCNT, POPPAR, SHIFTL, SHIFTR, STORAGE_SIZE, TRAILZ,
• Additions to intrinsic module ISO_FORTRAN_ENV: ATOMIC_INT_KIND, ATOMIC_LOGICAL_KIND, CHARACTER_KINDS, INTEGER_KINDS, INT8, INT16, INT32, INT64, LOCK_TYPE, LOGICAL_KINDS, REAL_KINDS, REAL32, REAL64, REAL128, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, STAT_UNLOCKED

4 Intel® Debugger (IDB)

4.1 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:

    ifort -g -o hello.exe hello.f90

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:

    ifort -c -g -o hello.o hello.f90
    ifort -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:

    ifort -g -save-temps -o hello.exe hello.f90

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

4.2.1 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.

4.2.2 Debug Info from Shared Libraries
The debugger does not read debug information from shared libraries. Therefore you can not set a breakpoint to symbols like `_exit` which are part of a system library.

4.2.3 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.f90

/auto/mount/site/foo/usr1/user_me/f_code/hello.f90

(Cannot find source file hello.f90)

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).

4.2.4 Local variables may not be visible
The linker on Mac OS X 10.5.4 (and subsequent versions) does not always issue definitions of local variables into the debug information in the executable. We do not have a characterization of when this occurs. The end result is that the variable is not visible or is visible but incorrectly evaluated.

The instances we have seen have involved local arrays in Fortran programs which were allocated in the `.bss` segment by the compiler. A work-around is to change the source to make the variable be global rather than local. In Fortran this is most easily done by putting the variable into a module or common block. Intel and Apple are working together to resolve this issue.
4.2.5  Printing Fortran REAL*16 variables
The debugger does not print the correct value for Fortran REAL*16 variables.

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

4.2.7  Debugging applications that exec
The $catchexecs control variable is not supported.

4.2.8  Fortran alternate entry points
Formal parameters of alternate entry points are not visible from within the debugger if they are not also formal parameters of the main entry point.

4.2.9  Snapshots
Snapshots are not yet supported as described in the manual.

4.2.10 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.

4.2.11 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.

4.2.12 Fortran modules and commons
A globally defined Fortran module should be rescoped with a double percent (%%) when referred to. For example, to set a breakpoint in the subroutine bar contained in a globally defined module foo, do

(idb) stop in foo%%bar

Please refer to the following section in the manual for the rescoping syntax:

Looking Around the Code, the Data and Other Process Information >
Looking at the Data >
The print Command

If you try to access (print, etc.) a Fortran module or common using the name in the source code, the debugger may not be able to find it. As a workaroud, the you can try prepending '_' to the name. For example, in the source code, if you have a common called "com":

(idb) print _com
4.2.13 Graphical User Interface (GUI)
This version of the debugger does not support the GUI.

4.2.14 MPP Debugging Restrictions
MPP debugging is not supported as described in the manual.

4.2.15 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.

4.2.16 Core File Debugging
Debugging core files is not yet supported.

4.2.17 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.

4.2.18 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.

4.2.19 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.

4.2.20 $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.

4.2.21 info stack Usage

The GDB mode 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.

4.2.22 $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® Math Kernel Library

This section summarizes changes, new features and late-breaking news about this version of Intel® Math Kernel Library.

5.1 What's New in Intel® MKL 10.3 Update 1

- PARDISO/DSS: Added true F90 overloaded API (see the Intel® MKL reference manual for more information)
- PARDISO: Improved the statistical reporting to be more reader friendly
- Sparse BLAS: Improved performance of ?BSRMM functions on the latest Intel& processors
- FFTs: Support for negative strides
- FFT examples: Added examples for split-complex FFTs in C and Fortran using both the DFTI and FFTW3 interfaces
- VML: Improved performance of real in-place Add/Sub/Mul/Sqr functions on systems supporting SSE2 and SSE3
- Poisson Library: Changed the default behavior of the Poisson library functions from sequential to threaded operation
- Bug fixes

5.2 What's New in Intel® MKL 10.3

- BLAS
  - New functions for computing 2 matrix-vector products at once: [D/S]GEM2VU, [Z/C]GEM2VC
• New functions for computing mixed precision general matrix-vector products: 
  [DZ/SC]GEMV
• New function for computing the sum of two scaled vectors: *AXPBY
• Intel® AVX optimizations in key functions: SMP LINPACK, level 3 BLAS, 
  DDOT, DAXPY

• LAPACK
  • New C interfaces for LAPACK supporting row-major ordering
  • Integrated Netlib LAPACK 3.2.2 including one new computational routine 
    (*GEQRFP) and two new auxiliary routines (*GEQR2P and *LARFGP) and 
    the earlier LAPACK 3.2.1 update
  • Intel® AVX optimizations in key functions: DGETRF, DPOTRF, DGEQRF

• PARDISO
  • Improved performance of factor and solve steps in multi-core environments
  • Introduced the ability to solve for sparse right-hand sides and perform partial 
    solves—produces partial solution vector
  • Improved performance of the out-of-core (OOC) factorization step
  • Support for zero-based (C-style) array indexing
  • Zeros on the diagonal of the matrix are no longer required in sparse data 
    structures for symmetric matrices
  • New ILP64 PARDISO interface allows the use of both LP64 and ILP64 
    versions when linked to the LP64 libraries
  • The memory required for storing files on the disk in OOC mode can now be 
    estimated just after reordering

• Sparse BLAS
  • Format conversion functions now support all data types (single and double 
    precision for real and complex data) and can return sorted or unsorted 
    arrays

• FFTs
  • Intel AVX optimizations in all 1D/2D/3D FFTs
  • Improved performance of 2D and 3D mixed-radix FFTs for single and double 
    precision data for all systems supporting the SSE4.2 instruction set
  • Support for split-complex data represented as two real arrays introduced for 
    2D/3D FFTs
  • Support for 1D complex-to-complex transforms of large prime lengths

• VML
  • A new function for computing (ax+b)/(cy+d) where a, b, c, and d are scalars, 
    and x and y are real vectors: v[s/d]LinearFrac()
  • Intel AVX optimizations for real functions
  • A new mode for setting denormals to zero, overflow support for complex 
    vectors, and for every VML function a new function with an additional 
    parameter for setting the accuracy mode

• VSL
  • A set of new Summary Statistics functions was added covering basic 
    statistics, covariance and correlation, pooled, group, partial, and robust 
    covariance/correlation, quantiles and streaming quantiles, outliers detection 
    algorithm, and missing values support
    • Performance optimized algorithms: MI algorithm for support of 
      missing values, TBS algorithm for computation of robust covariance, 
      BACON algorithm for detection of outliers, ZW algorithm for 
      computation of quantiles (streaming data case), and 1PASS 
      algorithm for computation of pooled covariance
- Improved performance of SFMT19937 Basic Random Number Generator (BRNG)
- Intel® AVX optimizations: MT19937 and MT2203 BRNGs
- Added runtime dispatching dynamic libraries allowing link to a single interface library which loads dependent libraries dynamically at runtime depending on runtime CPU detection and/or library function calls
- The custom dynamic libraries builder now uses the runtime dispatching dynamic libraries on the Linux* and Mac OS* X operating systems
- A new directory structure has been established to simplify integration of Intel MKL with the Intel® Parallel Studio XE family of products and directories formerly designated as "em64t" are now designated by the "intel64" tag
- The sparse solver functionality has been fully integrated into the core Intel MKL libraries and the libraries with "solver" in the filename have been removed from the product

5.3 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 (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. Hammerling, 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. Hammerling, 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.
6  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:


Celeron, Centrino, Intel, Intel logo, Intel386, Intel486, Intel Atom, Intel Core, Itanium, MMX, Pentium, VTune, 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.

Copyright © 2010 Intel Corporation. All Rights Reserved.