Intel® Fortran Composer XE 2011 for Linux* Installation Guide and Release Notes

Document number: 321415-003US
14 January 2011

Table of Contents

1 Introduction

This document describes how to install the product, provide a summary of new and changed product features and includes 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 2 (12.0.2)

- Intel® Math Kernel Library updated to 10.3 Update 2
- The way that the Static Security Analysis feature creates data files has changed
- Corrections to reported problems

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 Linux** includes the following components:

- Intel® Fortran Compiler XE 12.0.2 for building applications that run on IA-32 and Intel® 64 architecture systems running the Linux* operating system
- Intel® Debugger 12.0.2
- Intel® Math Kernel Library 10.3 Update2
- On-disk documentation

1.3 System Requirements

For an explanation of architecture names, see
http://software.intel.com/en-us/articles/intel-architecture-platform-terminology

Requirements to develop IA-32 architecture applications

- A PC based on an IA-32 or Intel® 64 architecture processor supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium® 4 processor or later, or compatible non-Intel processor)
- Development for a target different from the host may require optional library components to be installed from your Linux Distribution.
- For the best experience, a multi-core or multi-processor system is recommended
- 1GB of RAM (2GB recommended)
- 2GB free disk space for all features
- One of the following Linux distributions (this is the list of distributions tested by Intel; other distributions may or may not work and are not recommended - please refer to Technical Support if you have questions):
    - Asianux* 3.0
    - Fedora* 12,13
    - Red Hat Enterprise Linux* 4, 5, 6
    - SUSE LINUX Enterprise Server* 10,11
    - Ubuntu* 10.04
    - Debian* 5.0
- Linux Developer tools component installed, including gcc, g++ and related tools
- Library libunwind.so is required in order to use the `-traceback` option. Some Linux distributions may require that it be obtained and installed separately.
- If developing on an Intel® 64 architecture system, some Linux distributions may require installation of one or more of the following additional Linux components: ia32-libs, lib32gcc1, lib32stdc++6, libc6-dev-i386, gcc-multilib

Requirements to develop Intel® 64 architecture applications

- A PC based on an Intel® 64 architecture processor (Intel® Pentium 4 processor or later, or compatible non-Intel processor)
- For the best experience, a multi-core or multi-processor system is recommended
- 1GB of RAM (2GB recommended)
- 2GB free disk space for all features
- 100 MB of hard disk space for the virtual memory paging file. Be sure to use at least the minimum amount of virtual memory recommended for the installed distribution of Linux
- One of the following Linux distributions (this is the list of distributions tested by Intel; other distributions may or may not work and are not recommended - please refer to Technical Support if you have questions):
    - Asianux* 3.0
    - Fedora* 12, 13
    - Red Hat Enterprise Linux* 4, 5, 6
    - SUSE LINUX Enterprise Server* 10.2, 11.1 SP1
    - Ubuntu* 10.04
- Linux Developer tools component installed, including gcc, g++ and related tools

- Library libunwind.so is required in order to use the `-traceback` option. Some Linux distributions may require that it be obtained and installed separately.

Additional requirements to use the Graphical User Interface of the Intel® Debugger

- IA-32 Architecture system or Intel® 64 Architecture system
- Java* Runtime Environment (JRE) 5.0 (also called 1.5)
- A 32-bit JRE must be used on an IA-32 architecture system and a 64-bit JRE must be used on an Intel® 64 architecture system

Notes

- The Intel compilers are tested with a number of different Linux distributions, with different versions of gcc. Some Linux distributions may contain header files different from those we have tested, which may cause problems. The version of glibc you use must be consistent with the version of gcc in use. For best results, use only the gcc versions as supplied with distributions listed above.
- The default for the Intel® compilers is to build IA-32 architecture applications that require a processor supporting the Intel® SSE2 instructions - for example, the Intel® Pentium® 4 processor. A compiler option is available to generate code that will run on any IA-32 architecture processor.
- Compiling very large source files (several thousands of lines) using advanced optimizations such as -O3, -ipo and -openmp, may require substantially larger amounts of RAM.
- The above lists of processor model names are not exhaustive - other processor models correctly supporting the same instruction set as those listed are expected to work. Please refer to Technical Support if you have questions regarding a specific processor model
- Some optimization options have restrictions regarding the processor type on which the application is run. Please see the documentation of these options for more information.

1.3.1 Red Hat Enterprise Linux* 4 Support Deprecated

In a future major release of Intel® Fortran Composer XE, support will be removed for installation and use on Red Hat Enterprise Linux 4. Intel recommends migrating to a newer version of these operating systems.

1.3.2 IA-64 Architecture (Intel® Itanium®) Development Not Supported

This product version does not support development on or for IA-64 architecture (Intel® Itanium®) systems. The version 11.1 compiler remains available for development of IA-64 architecture applications.

1.4 Documentation

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

Optimization Notice
Intel® 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. 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. 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. Notice revision #20101101

1.5 Japanese Language Support

Intel compilers provide support for Japanese language users when the combined English-Japanese product is installed. 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-linux/

1.6 Technical Support

Register your license 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 received your product on DVD, mount the DVD, change the directory (`cd`) to the top-level directory of the mounted DVD and begin the installation using the command:

```
./install.sh
```

If you received the product as a downloadable file, first unpack it into a writeable directory of your choice using the command:

```
tar -xzvf name-of-downloaded-file
```

Then change the directory (`cd`) to the directory containing the unpacked files and begin the installation using the command:

```
./install.sh
```

Follow the prompts to complete installation.

Note that there are several different downloadable files available, each providing different combinations of components. Please read the download web page carefully to determine which file is appropriate for you.

You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions.

2.1 Activation of Purchase after Evaluation Using the Intel Activation Tool

Note for evaluation customers: a new tool Intel Activation Tool "`Activate`" is included in this product release and installed at `/opt/intel/ActivationTool/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/ActivationTool/Activation/Activate`. It will convert your evaluation software to a fully licensed product. To use the tool:

```
$ /opt/intel/ActivationTool/Activation/Activate [SN_here]
```

2.2 Silent Install

For information on automated or "silent" install capability, please see http://software.intel.com/en-us/articles/intel-compilers-for-linux-silent-installation-guides

2.3 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.

2.4 Known Installation Issues

- If you have enabled the Security-Enhanced Linux (SELinux) feature of your Linux distribution, you must change the `SELINUX` mode to `permissive` before installing the Intel Fortran Compiler. Please see the documentation for your Linux distribution for details. After installation is complete, you may reset the `SELINUX` mode to its previous value.
- On some versions of Linux, auto-mounted devices do not have the "exec" permission and therefore running the installation script directly from the DVD will result in an error such as:

  ```
  bash: ./install.sh: /bin/bash: bad interpreter: Permission denied
  ```

  If you see this error, remount the DVD with exec permission, for example:

  ```
  mount /media/<dvd_label> -o remount,exec
  ```

  and then try the installation again.
- The product is fully supported on Ubuntu and Debian Linux distributions for IA-32 and Intel® 64 architecture systems as noted above under System Requirements. Due to a restriction in the licensing software, however, it is not possible to use the Trial License feature when evaluating IA-32 components on an Intel® 64 architecture system under Ubuntu or Debian. This affects using a Trial License only. Use of serial numbers,

license files, floating licenses or other license manager operations, and off-line activation (with serial numbers) is not affected. If you need to evaluate IA-32 components of the product on an Intel® 64 architecture Ubuntu or Debian system, please visit the Intel® Software Evaluation Center (http://www.intel.com/cd/software/products/asmo-na/eng/download/eval/) to obtain an evaluation serial number.

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, and can also perform a "non-root" install in the location of your choice.

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

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 directory containing man pages for the latest installed version
- `Documentation` – symbolic link to the `documentation` directory
- `Samples` – symbolic link to the `samples` directory

- `eclipse_support` – symbolic link to a directory created by the Intel Debugger component that is shared between Intel Fortran and Intel C++. Intel does not provide Eclipse support for Fortran.

Each `composerxe-2011-<n>.<pkg>` directory contains the following directories that reference a specific update of the Intel® Composer XE 2011 compiler:

- `bin` – all executables
- `compiler` – shared libraries and header files
- `debugger` – debugger files
- `Documentation` – documentation files
- `man` – `man` pages
- `mkl` – Intel® Math Kernel Library libraries and header files
- `Samples` – Product samples and tutorial files
- `eclipse_support` –directory created by the Intel Debugger component that is shared between Intel Fortran and Intel C++. Intel does not provide Eclipse support for Fortran.

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.

2.6 Removal/Uninstall

Removing (uninstalling) the product should be done by the same user who installed it (root or a non-root user). If `sudo` was used to install, it must be used to uninstall as well. It is not possible to remove the compiler while leaving any of the performance library components installed.

1. Open a terminal window and set default (`cd`) to any folder outside `<install-dir>`
2. Type the command: `<install-dir>/bin/ia32/uninstall_cprof.sh` (substitute `intel64` for `ia32` as desired)
3. Follow the prompts
4. Repeat steps 2 and 3 to remove additional platforms or versions

If you also have the same-numbered version of Intel® C++ Compiler installed, it may also be removed.

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 Linux* (8.0 and later) may be used in a build with version 12.0. 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 is also compatible with gcc.

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)
- Coarrays
- CODIMENSION attribute
- SYNC ALL statement
- SYNC IMAGES statement

- SYNC MEMORY statement
- CRITICAL and END CRITICAL statements
- LOCK and UNLOCK statements
- ERROR STOP statement
- ALLOCATE and DEALLOCATE may specify coarrays
- Intrinsic procedures IMAGE_INDEX, LCOBOUND, NUM_IMAGES, THIS_IMAGE, UCOBOUND
    - **Note:** ATOMIC_DEFINE and ATOMIC_REF are not supported in this version
- 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, 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, LOCK_TYPE, LOGICAL_KINDS, REAL_KINDS, REAL32, REAL64, REAL128, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, STAT_UNLOCKED

3.2.3 Coarrays

No special procedure is necessary to run a program that uses coarrays; you simply run the executable file. The underlying parallelization implementation is Intel® MPI. Installation of the compiler automatically installs the necessary Intel® MPI run-time libraries to run on shared memory. The Intel® Cluster Toolkit installs the necessary Intel® MPI run-time libraries to run on distributed memory. Use of coarray applications with any other MPI implementation, or with OpenMP*, is not supported.

By default, the number of images created is equal to the number of execution units on the current system. You can override that by specifying the option `/Qcoarray-num-images:<n>` on the ifort command that compiles the main program. You can also specify the number of images in an environment variable `FOR_COARRAY_NUM_IMAGES`.

3.2.3.1 Specifying Shared or Distributed Memory Processing of Coarrays

The documentation for the `-coarray` option currently says:

> Using `/Qcoarray` (Windows*) or `-coarray` (Linux*) with no argument is equivalent to running on multi-node (distributed memory) if an Intel® Cluster Toolkit license is installed or on single node (shared memory) if there is no Intel® Cluster Toolkit license installed.

The implementation has changed since the above text was written. The new behavior is that if `-coarray` is specified without the *`memory`* argument, shared memory is used whether or not the Intel® Cluster Toolkit license is present. To use distributed memory, which requires that a license for Intel® Cluster Toolkit is present, specify `-coarray=distributed`.

3.2.3.2 How to Debug a Coarray Application

The following instructions describe how to debug a Coarray application.

1. Add a stall loop to your application before the area of code you wish to debug, e.g.:

```
LOGICAL VOLATILE :: WAIT_FOR_DEBUGGER
LOGICAL, VOLATILE :: TICK
   :
DO WHILE(WAIT_FOR_DEBUGGER)
  TICK = .NOT. TICK
  END DO
! Code you want to debug is here
!
```

   The use of `VOLATILE` is required to ensure that the loop will not be removed by the compiler. If the problem is only found on one image, you can wrap the loop in `IF (THIS_IMAGE() .EQ. 4) THEN` or the like.
2. Compile and link with debug enabled (`-g`).
3. Create at least N+1 terminal windows on the machine where the application will be running, where N is the number of images your application will have.
4. In a terminal window, start the application.
   `linuxprompt> ./my_app`
5. In each of the other terminal windows, set your default directory to be the same as the location of the application executable. Use the ps command in one of the windows to find out which processes are running your application:

```
linuxprompt> ps -ef | grep 'whoami' | grep my_app
```

   There will be several processes. The oldest is the one you started in step 4 – it has run the MPI launcher and is now waiting for the others to terminate. Do not debug it.

   The others will look like this:

```
<your-user-name>  25653 25650 98 15:06 ?        00:00:49 my_app
<your-user-name>  25654 25651 97 15:06 ?        00:00:48 my_app
<your-user-name>  25655 25649 98 15:06 ?        00:00:49 my_app
```

   The first number is the PID of the process (e.g., 25653 in the first line).

   Call the PIDs of these N processes running "my_app" P1, P2, P3 and so on.

6. In each window other than the first, start your debugger and set it to stop processes when attached:

   ```
   linuxprompt> idb –idb
   (idb) set $stoponattach = 1
   ```

   or

   ```
   linuxprompt> gdb
   ```

7. Attach to one of the processes (e.g. to P1 in window 1, to P2 in window 2, etc.)

   ```
   (idb) attach <P1> my_app
   ```

   or

   ```
   (gdb) attach <P1>
   ```

8. Get execution out of the stall loop:

   (idb) assign WAIT_FOR_DEBUGGER = .FALSE.

   or

   (gdb) set WAIT_FOR_DEBUGGER = .false.

9. You can now debug.

If you are using idb, you can use the multiprocess capability of idb to have only one debugger window instead of N. First, attach to each process and get out of the loop (steps 7 and 8).

```
(idb) attach <P1> my_app
(idb) assign WAIT_FOR_DEBUGGER = .FALSE.
(idb) attach <P2> my_app
(idb) assign WAIT_FOR_DEBUGGER = .FALSE.
(idb) attach <P3> my_app
(idb) assign WAIT_FOR_DEBUGGER = .FALSE.
```

Use the "process" command to switch debugging focus from one process to another:

```
(idb) process <Pn>
```

Processes not focused on will remain in the state they were left in: with breakpoints and watchpoints set but not running.

3.2.3.3 Coarray Known Issues

The following features are known not to work in this version:

- Character data type coarrays
- Coarrays of derived type where the type contains an ultimate component that is ALLOCATABLE or POINTER
- Output (WRITE, PRINT, etc.) of an array slice of a coarray referencing another image. A whole array reference, or a single element works.
- Default initialization of a REAL(16) or COMPLEX(16) coarray
- LOCK and UNLOCK cannot be used on another image.
- STAT= or ERRMSG= arguments on LOCK, UNLOCK, SYNC IMAGES, SYNC MEMORY, or SYNC ALL are not being set correctly.

3.2.4 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.

3.2.5 Other Changes

- The ability to create a source listing file with identifier cross-reference has been added
- Guided auto-parallelism
- 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
- The ability to generate a build dependencies output file has been added

3.2.5.1 Change in Static Security Analysis Behavior

The `inspxe-runsc` command line utility that is distributed with Intel® Composer XE 2011 has been changed. This change only affects users who use Composer XE 2011 to perform Static Security Analysis (SSA). Those that do not use SSA and those that perform SSA without using this utility are unaffected. SSA is only available to users of Intel® Parallel Studio XE 2011 or Intel® C++ Studio XE 2011, so users who do not have those products are unaffected.

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

The versions of `inspxe-runsc` included in Composer XE 2011 and Composer XE 2011 Update 1 generate all the SSA results in a single directory. In the multiple link case this violated

the rule that all the SSA results for one and only one project must be created in the same directory. The updated version of `inspxe-runsc` respects this rule by generating results for each link step in a separate directory. The name of that directory is formed from the name of the file being linked. Thus if a build specification describes a project that builds two executables, file1.out and file2.out, then earlier versions of `inspxe-runsc` would create two results, one for file1 and one for file2, say r000sc and r001sc, in the same directory. The new version of `inspxe-runsc` will also create two results, but the one for file1 will be created in "My Inspector XE results – file1/r000sc" and the one for file2 will be created in "My Inspector XE results – file2/r000sc". The directories containing the results are both created in the same parent directory.

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

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

3.3 New and Changed Compiler Options

Please refer to the compiler documentation for details

- -assume [no]fpe_summary
- -assume [no]old_ldout_format
- -coarray
- -coarray-num-images
- -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
- -guide-file
- -guide-file-append
- -guide-opts
- -guide-par
- -guide-vec
- -list
- -list-line-len
- -list-page-len
- -opt-args-in-regs
- -par-runtime-control
- -prof-value-profiling
- -profile-functions
- -profile-loops-report
- -show=keyword
- -simd
- -standard-semantics

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

3.4 Other Changes and Notes

3.4.1 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`.

3.4.2 Establishing the Compiler Environment

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.

3.4.3 OpenMP* Legacy Libraries Removed

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

3.4.4 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 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
- ERRMSG 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)
- Coarrays
- CODIMENSION attribute
- SYNC ALL statement
- SYNC IMAGES statement
- SYNC MEMORY statement
- CRITICAL and END CRITICAL statements
- LOCK and UNLOCK statements
- ERROR STOP statement
- ALLOCATE and DEALLOCATE may specify coarrays
- Intrinsic procedures IMAGE_INDEX, LCOBOUND, NUM_IMAGES, THIS_IMAGE, UCOBOUND
    - **Note:** ATOMIC_DEFINE and ATOMIC_REF are not supported in this version
- 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, 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, LOCK_TYPE, LOGICAL_KINDS, REAL_KINDS, REAL32, REAL64, REAL128, STAT_LOCKED, STAT_LOCKED_OTHER_IMAGE, STAT_UNLOCKED

4 Intel® Debugger (IDB)

The following notes refer to the Graphical User Interface (GUI) available for the Intel® Debugger (IDB) when running on IA-32 and Intel® 64 architecture systems. In this version, the `idb` command invokes the GUI – to get the command-line interface, use `idbc`.

4.1 Setting up the Java* Runtime Environment

The Intel® IDB Debugger graphical environment is a Java application and requires a Java Runtime Environment (JRE) to execute. The debugger will run with a version 5.0 (also called 1.5).

Install the JRE according to the JRE provider's instructions.

Finally you need to export the path to the JRE as follows:

```
export PATH=<path_to_JRE_bin_dir>:$PATH
```

4.2 Starting the Debugger

To start the debugger, first make sure that the compiler environment has been established as described at Establishing the Compiler Environment. Then use the command:

```
idb
```

or

```
idbc
```

as desired.

Once the GUI is started and you see the console window, you're ready to start the debugging session.

Note: Make sure, the executable you want to debug is built with debug info and is an executable file. Change permissions if required, e.g. `chmod +x <application_bin_file>`

4.3 Additional Documentation

Online help titled *Intel® Compilers / Intel® Debugger Online Help* is accessible from the debugger graphical user interface as `Help > Help Contents`.

Context-sensitive help is also available in several debugger dialogs where a `Help` button is displayed.

4.4 Debugger Features

4.4.1 Main Features of IDB

The debugger supports all features of the command line version of the Intel® IDB Debugger. Debugger functions can be called from within the debugger GUI or the GUI-command line. Please refer to the Known Limitations when using the graphical environment.

4.4.1.1 Threads Window

- Improved Data Sharing Dectection
- Support for OpenMP* 3.0

- Support for Linux* OS synchronization functionsImproved data sharing detection analysis performance

4.4.1.2 Extended Breakpoints Feature

With this feature you can set breakpoints on routines in shared libraries which have not yet been loaded. The requested breakpoint will be realized whenever possible. You'll see unrealized breakpoints marked with a yellow triangle (not having an address, file and symbol name) in the GUI. On the command line those are marked as `<PENDING>`. Any ambiguity is directly resolved and you will get multiple realizations, e.g. requesting a breakpoint for an overloaded function. In the GUI, those are visualized as a tree with the requesting breakpoint as its node. On the command line the requesting breakpoint is marked as `<MULTIPLE>` and its realizations follow. Please note that for the command line this feature is only available in GDB mode.

4.4.1.3 Command `solib-search-path` now Implemented

The command line debugger idbc and the Command window of the GUI debugger now support the existing gdb command `solib-search-path` which is used to look up images or shared libraries when they have not been found in the usual places such as `$LD_LIBRARY_PATH`.

Please invoke the command line help to see the `solib-search-path` command usage:

(idb) help set solib-search-path

(idb) help show solib-search-path

or the abbreviated commands:

(idb) h set sol

(idb) h sho sol

4.4.1.4 New Command for Disassembly Style Display

The IDB debugger now provides two styles of disassembly views in the Assembler window or on the Command windows.

The new commands on the Command window are:

(idb) set disassembly-flavor [att|intel]

(idb) show disassembly-flavor

The commands can also be found by invoking the help:

(idb) help set

(idb) help show

In the GUI/Assembler window right-click 'Change Style' to switch between Intel and ATT style. ATT stands for AT&T style (also known as GNU style).

4.5 Known Problems

4.5.1 Coarray elements cannot be viewed.

The IDB Debugger cannot view coarray elements. Please refer to section 3.2.3.1 'How to Debug a Coarray Application' where a workaround is described.

4.5.2 Signals Dialog not working Signals Dialog not working

The Signals dialog accessible via the GUI dialog Debug / Signal Handling or the shortcut Ctrl+S is not working correctly. Please refer to the Intel® Debugger (IDB) Manual for use of the signals command line commands instead.

4.5.3 Resizing GUI

If the debugger GUI window is reduced in size, some windows may fully disappear. Enlarge the window and the hidden windows will appear again.

4.5.4 `$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.

4.5.5 `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 value prints the innermost –`num` frames in reverse order.

4.5.6 `$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

4.5.7 SIGTRAP error on some Linux* Systems

On some Linux distributions (e.g. Red Hat Enterprise Linux Server release 5.1 (Tikanga)) a SIGTRAP error may occur when the debugger stops at a breakpoint and you continue debugging. As a workaround you may define the SIGTRAP signal as follows on command line:

```
(idb) handle SIGTRAP nopass noprint nostop
SIGTRAP is used by the debugger.
SIGTRAP       No        No      No              Trace/breakpoint trap
(idb)
```

```
Caveat: With this workaround all SIGTRAP signals to the debuggee are
blocked.
```

4.5.8 idb GUI cannot be used to debug MPI processes

The idb GUI cannot be used to debug MPI processes. The command line interface (idbc) can be used for this purpose.

4.5.9 Thread Syncpoint Creation in GUI

While for plain code and data breakpoints the field “Location” is mandatory, thread syncpoints require both “Location” and “Thread Filter” to be specified. The latter specifies the threads to synchronize. Please note that for the other breakpoint types this field restricts the breakpoints created to the threads listed.

4.5.10 Data Breakpoint Dialog

The fields “Within Function” and “Length” are not used. The location to watch provides the watched length implicitly (the type of the effective expression is used). Also “Read” access is not working.

4.5.11 Stack Alignment for IA-32 Architecture

Due to changes in the default stack alignment for the IA-32 architecture, the usage of inferior calls (i.e. evaluation of expressions that cause execution of debuggee code) might fail. This can cause as well crashes of the debuggee and therefore a restart of the debug session. If you need to use this feature, make sure to compile your code with 4 byte stack alignment by proper usage of the `-falign-stack=<mode>` option.

4.5.12 GNOME Environment Issues

With GNOME 2.28, debugger menu icons may not being displayed by default. To get the menu icons back, you need to go to the “System->Preferences->Appearance, Interface” tab and enable, "Show icons in menus". If there is not “Interface” tab available, you can change this with the corresponding `GConf` keys in console as follows:

```
gconftool-2 --type boolean --set /desktop/gnome/interface/buttons_have_icons true

gconftool-2 --type boolean --set /desktop/gnome/interface/menus_have_icons true
```

4.5.13 Accessing Online-Help

On systems where the Online-Help is not accessible from the IDB Debugger GUI Help menu, you can access the web-based debugger documentation from:

http://software.intel.com/en-us/articles/intel-software-technical-documentation

5 Intel® Math Kernel Library

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

5.1 What's New in Intel® MKL 10.3 Update 2

- BLAS: Improved performance of transposition functions on the Intel® Xeon® processor 5600 series
- BLAS: Added examples for transposition routines
- FFT: Added Fortran examples showing how to reduce application footprint by linking only functions with the desired precision
- FFT: Added check for stride consistency on in-place real transforms with CCE storage
- FFT: Expanded threading to new cases for multi-dimensional transforms
- VSL: Improved performance of Multivariate Gaussian random number generator for single- and double-precision on 4-core Intel® Xeon® processors 5500 series
- VML: Improved performance of in-place operation of Add, Mul, and Sub functions on the Intel® Xeon® processor 5500 series
- Bug fixes

5.2 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

5.3 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

5.4 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.

6 Disclaimer and Legal Information