Agenda

Time	Title
09:00-09:15	Welcome, Introduction
09:15-09:45	Overview – Intel Processor Architecture Evolution
09:45-10:15	Intel's Software Development offerings at a glance
10:15-11:15	Strategies for Parallelism with Intel® Threading Methodologies
11:15-12:30	**Intel® Composer XE – the powerful compiler and performance libraries collection**
12:30-13:30	*Break*
13:30-14:00	Intel® Inspector XE - Detect memory and threading errors
14:00-14:30	Intel® Amplifier XE - Understand performance issues
14:30-15:00	Intel® Advisor XE – Get started with parallelization
15:00-16:00	Intel Development Tools for Multi-threading in a typical Development Cycle – Life-Demo – Wrap-up
16:00- …	Q&A, opens, discussion

Intel® Composer XE
Overview

Hubert Haberstock
Technical Consulting Engineer
Intel GmbH

What is the Composer XE?

- New name for the Intel compilers
  - Intel® Composer XE for Windows*|Linux*|Mac OS* X (formerly Intel® Compiler Suite Professional Edition for Windows*|Linux*|Mac OS* X)
  - Intel® C++ Composer XE for Windows*|Linux*|Mac OS* X (formerly Intel® Compiler Professional Edition for Windows*|Linux*|Mac OS* X)
  - Intel® Visual Fortran Composer XE for Windows* (formerly Intel® Visual Fortran Compiler Professional Edition for Windows*)
  - Intel® Fortran Composer XE for Linux*|Mac OS* X (formerly Intel® Visual Fortran Compiler Professional Edition for Linux*|Mac OS* X)
- Part of the new product bundle Intel® C++|Fortran|Parallel Studio XE
  - but also available separately
- Includes performance libraries, debugger, debugger extension, new threading methodologies

Intel Composer XE 2013 SP1 - Overview
Compilers, Performance Libraries, Debugging Tools

Intel® C++ Composer XE 2013 SP1

- Intel® C++ Compiler XE 14.0
- Intel® Debugger + gdb add-ons
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
- Intel® Math Kernel Library
- Intel® Integrated Performance Primitives

Intel® Fortran Composer XE 2013 SP1

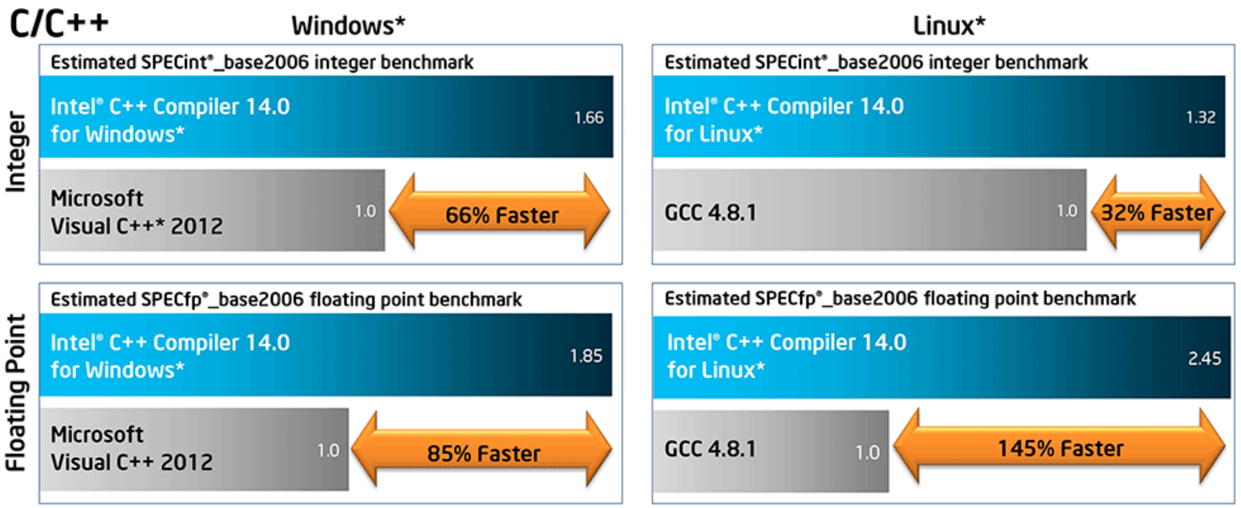
- Intel® Fortran Compiler XE 14.0
- Intel® Debugger + gdb add-ons
- Intel® Math Kernel Library
- Intel® Integrated Performance Primitives

Windows*, Linux*, Mac OS*

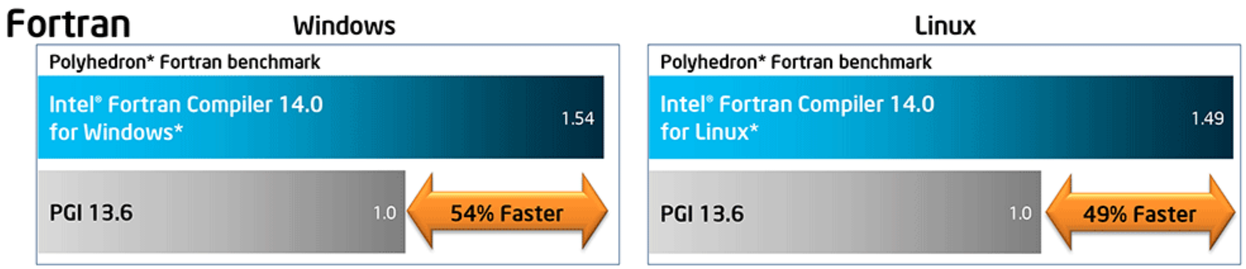
- **Leading Performance Optimizing Compilers**
  - Intel® C++ with Intel® Cilk™ Plus and Fortran Compiler
  - Intel® Integrated Performance Primitives
  - Intel® Math Kernel Library libraries
- **Standard Support**
  - OpenMP* 4.0
  - Support for key parts of the latest Fortran and C++ standards, Visual Studio* Shell for Visual Fortran*
- **Compatibility – Mix and Match**
  - Windows: Visual* C++ and Visual Studio* 2008, 2010, 2012
  - Linux*, OS X*, gcc and, for C++ Eclipse & Xcode for Mac
- **Architecture Support**
  - Intel compatible IA processors
  - Intel® Xeon Phi™ product family,

Continued compiler performance leadership on C++ and Fortran

Industry Leading Performance using the Intel® C/C++ and Fortran Compilers
(Higher is Better)



Configuration Info - SW Versions: Intel® C/C++ 14.0, Microsoft Visual C++ 2012 (Windows), GCC 4.8.1 (Linux); Hardware: HP ProLiant DL360 G8*, 2 x Intel® Xeon® processor E5-2670 (2.60GHz, 20480KB LLC, TurboBoost is on, HyperThreading is on), 128GB RAM, SAS; Windows Operating System: Windows 7 Enterprise, Service pack 1; Linux Operating System: Red Hat Enterprise Linux Server release 6.2 (Santiago), Kernel 2.6.32-220.el6.x86_64. ||| Compiler Options: INT Speed (Linux): Intel compiler 14.0- C: -xAVX -ipo -O3 -no-prec-div -static -parallel -opt-prefetch -auto-p32. C++: xAVX -ipo -O3 -no-prec-div -opt-prefetch -auto-p32. GCC 4.8.1 - C: -m64 -Ofast -ffast-math -flto -march=native -mfpmath=sse -funroll-all-loops -static -ftree-parallelize-loops=16. C++: -m64 -Ofast -ffast-math -flto -march=native -mfpmath=sse -funroll-all-loops. ||| FP Speed (Linux): Intel Compiler 14.0- C: -xAVX -ipo -O3 -no-prec-div -static -parallel -opt-prefetch -ansi-alias. C++-xAVX -ipo -O3 -no-prec-div -static -opt-prefetch -ansi-alias. GCC 4.8.1 - C:-m64 -Ofast -ffast-math -flto -march=native -mfpmath=sse -funroll-all-loops -static -ftree-parallelize-loops=16. C++: -m64 -Ofast -ffast-math -flto -march=native -mfpmath=sse -funroll-all-loops. ||| INT Speed (Windows): Intel compiler 14.0- C: -Qvc11 -Qstd=c99 -QxAVX -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qparallel -Qauto-ilp32. C++: -Qvc11 -QxAVX -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qcxx_features -Qauto-ilp32. Visual C++* 2012- C: /O2 /Ob2 /GL /arch:AVX /favor:EM64T /fp:fast /Qpar. C++: /O2 /Ob2 /GL /arch:AVX /favor:EM64T /fp:fast /Qpar -EHsc -GR. ||| FP Speed (Windows): Intel compiler 14.0- C: -Qvc11 -Qstd=c99 -QxAVX -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qansi-alias -Qparallel. C++: -Qvc11 -QxAVX -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qcxx_features -Qansi-alias. Fortran: -QxAVX -Qipo -O3 -Qprec-div- -Qopt-prefetch -Qparallel.



Configuration Info - Compiler Versions: Intel® Fortran 14.0, PGI 13.6; Hardware: Blue Hills ATX Media IVB Desktop DZ77BH-55K-IDD; Intel® Core™ i7-3770K CPU @ 3.50GHz, TurboBoost is on, HyperThreading is off, 16GB RAM; Windows Operating System: Windows 7 Enterprise, Service pack 1; Linux Operating System: Red Hat Enterprise Linux Server release 6.3 (Santiago); Kernel 2.6.32-279.el6.x86_64 ; Compiler Options (Windows and Linux): Intel Fortran compiler 14.0: ifort -O3 -fast -parallel -ipo -no-prec-div , PGI 13.6: pgf95 -fastsse -Munroll=n:4 -Mipa=fast,inline -Mconcur=bind . Polyhedron benchmark (www.polyhedron.com) performed by Intel Corp on August 20, 2013.

Optimization Notice

Leadership Application Performance

Intel® C, C++ & Fortran Compilers

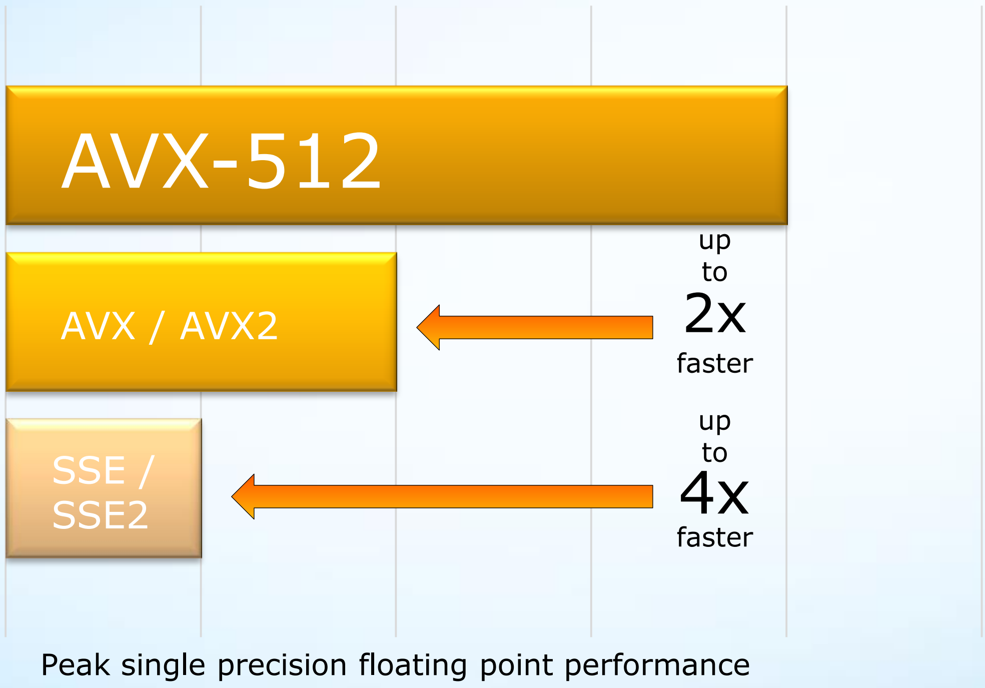
- More Performance for your C++ applications
  - Just recompile
  - Uses Intel® AVX and Intel® AVX2 instructions
  - NEW Intel® Xeon Phi™ product family support (Linux)
  - NEW Intel® Cilk™ Plus: Tasking and vectorization

- More Performance for your Fortran applications
  - Just recompile
  - Intel® Xeon Phi™ product family: Linux compiler, debugger support
  - NEW Access to Intel® AVX and Intel® AVX2 instructions (-xa or /Qxa)
  - NEW Auto-parallelizer & directives to access SIMD instructions
  - Coarrays & synchronization constructs support parallel programming
  - Loop optimization directives: VECTOR, PARALLEL, SIMD
  - NEW More control over array data alignment (align arrayNbytes)
  - NEW New in 2013 XE SP1 release: more Fortran 2008 support

NEW

Up to 4x Faster Performance

with Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Support



Intel® Compilers and Intel® Math Kernel Library will be updated in Q4 with AVX-512 support

- Significant leap to 512-bit SIMD support
- Increased compatibility with AVX
- One byte longer EVEX prefix, enabling additional functionality
- First implemented in the <u>future</u> Intel® Xeon Phi™ coprocessor, code named **Knights Landing**

Enables higher performance for the most demanding computational tasks

Expanded C++ 11 support

New in SP1 (14.0 compilers)

- Unrestricted unions (Linux*, OS X*)
- Non-static data member initializers
- Explicit Virtual Overrides
- Allowing move constructors to throw
- Defining Move Special Member Functions
- Inline namespaces
- Rvalue references v2

Full list of actual C++ 11 support: http://software.intel.com/en-us/articles/c0x-features-supported-by-intel-c-compiler

Full C++11 Support planned for next release

Excellent Support for C++ 11 on Windows* and Linux*

Standards Conformance – C99

- Full support:
  - Restricted pointers (`restrict` keyword).
  - Variable-length Arrays
  - Flexible array members
  - Complex number support (`_Complex` keyword)
  - Hexadecimal floating-point constants
  - Compound literals
  - Designated initializers
  - Mixed declarations and code
  - Macros with a variable number of arguments
  - Inline functions (`inline` keyword)
  - Boolean type (`_Bool` keyword)
  - `long double` is 64 bit, not 128 bit (only Linux*)
- Full list of actual C99 support: http://software.intel.com/en-us/articles/c99-support-in-intel-c-compiler

Optimization Notice

Excellent Fortran 2008 Support

Compilers & Libraries

- Maximum array rank has been raised to 31 dimensions (Fortran 2008 specifies 15)
- Recursive type may have ALLOCATABLE components
- 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
- 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
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

Support for Submodules highly requested

- Planned, but not committed yet

Full list of actual Fortran support: http://software.intel.com/en-us/articles/intel-fortran-compiler-support-for-fortran-language-standards

Leadership F2008 Support on Linux*, Windows* & OSX*

Optimization Notice

Expanded Fortran Capabilities in SP1

- Fortran 2008
  - ATOMIC_DEFINE and ATOMIC_REF
  - initialization of polymorphic INTENT(OUT) dummy arguments
  - standard handling of G format and of printing the value zero
  - polymorphic source allocation

- Co-array now supports Intel® Xeon Phi™ Coprocessor

Intel Fortran: Leadership Performance with a Leading Feature Set

Optimization Notice

FORTRAN 2003 Support

Added in Intel® Fortran Composer XE 2013 SP1 (aka 14.0, released Q3/2013)

- User-defined derived type I/O
- Parts of the missing intrinsics

FORTRAN 2003 features still missing today:

- Parameterized derived types (soon)
- Transformational intrinsics, such as MERGE and SPREAD, in initialization expression (later)

Complete FORTRAN 2003 support

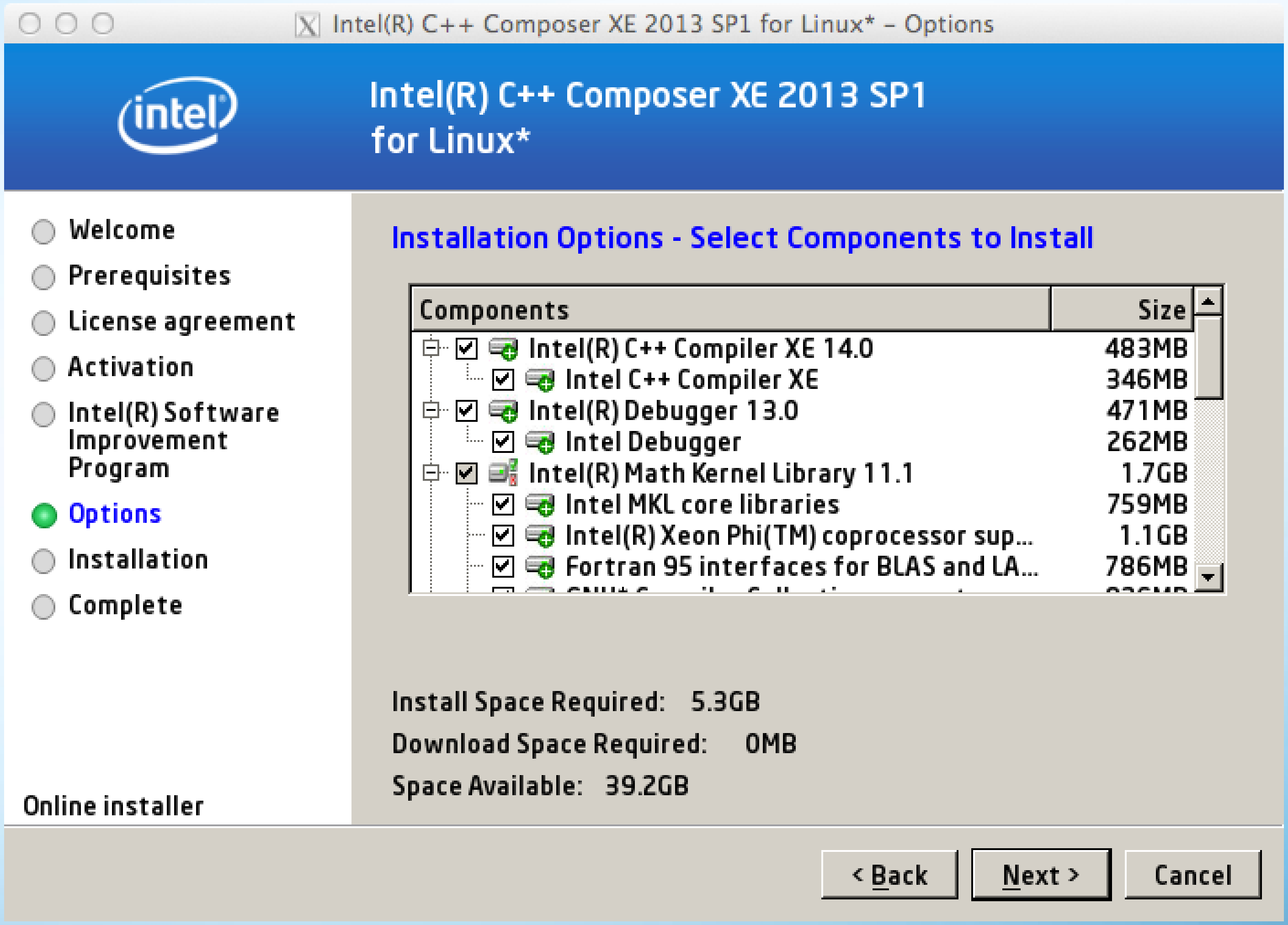
- Scheduled for next major release (15.0, Q4'2014)

Full list of actual Fortran support: http://software.intel.com/en-us/articles/intel-fortran-compiler-support-for-fortran-language-standards

New Installer Features

- Online installer: A small installation program ~4 MB
    - Select the components to install
    - Components are downloaded and installed on demand from the Intel® Registration and Download Center
    - Requires an internet connection, may require proxy setting. For details, see http://software.intel.com/en-us/articles/how-to-set-system-proxy
    - Internet connection speed will impact installation time
- Offline: Full packages of Composer XE and Libraries available for offline installations
- New GUI installation for Linux* available, --gui-mode

Optional GUI installer on Linux*


Intel(R) C++ Composer XE 2013 SP1 for Linux* – Options

Intel(R) C++ Composer XE 2013 SP1 for Linux*

- Welcome
- Prerequisites
- License agreement
- Activation
- Intel(R) Software Improvement Program
- Options
- Installation
- Complete

Online installer

Installation Options - Select Components to Install

Components	Size
☑ Intel(R) C++ Compiler XE 14.0	483MB
☑ Intel C++ Compiler XE	346MB
☑ Intel(R) Debugger 13.0	471MB
☑ Intel Debugger	262MB
☑ Intel(R) Math Kernel Library 11.1	1.7GB
☑ Intel MKL core libraries	759MB
☑ Intel(R) Xeon Phi(TM) coprocessor sup...	1.1GB
☑ Fortran 95 interfaces for BLAS and LA...	786MB

Install Space Required: 5.3GB

Download Space Required: 0MB

Space Available: 39.2GB

Intel® XEON Phi Support

Intel® Many Integrated Core (Intel® MIC) support added since 2013 release (compiler version 13.0)

- Only Intel® Xeon Phi processor family (code name Knights Corner / KNC), not KNF anymore
- One product only ! (no additional license required)
- Needs MPSS stack installed on host system to compile (not only for linking)
  - Very likely to be changed in future release

Enhancement of Compiler Reporting

Intel Compilers offers two switches to get compiler reports:

`-vec-report` for vectorization

`-opt-report` for loop transformations, inlining, prefetching, …

An initiative has started to enhance both reports. It will take some time to implement all enhancements requests we received and which we consider doable.

Some items improved or to be improved in future releases:

- More details on vectorization success or failure like
  - Unrolling
  - Remainder loop
  - Alignment

-Automated mapping of information to source code

-Optimization report will be enhanced too / will be made easier to read

Vectorization Report

Provides details on vectorization success & failure:

Linux*, Mac OS* X: `-vec-report<n>`, Windows*: `/Qvec-report<n>`

`n`	Diagnostic Messages
0	**Tells the vectorizer to report no diagnostic information. Useful for turning off reporting in case it was enabled on command line earlier.**
1	**Tells the vectorizer to report on vectorized loops. [default if `n` missing]**
2	**Tells the vectorizer to report on vectorized and non-vectorized loops.**
3	**Tells the vectorizer to report on vectorized and non-vectorized loops and any proven or assumed data dependences.**
4	**Tells the vectorizer to report on non-vectorized loops.**
5	**Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.**
6	**Tells the vectorizer to emit greater detail when reporting on vectorized and non-vectorized loops and any proven or assumed data dependences: Introduced by compiler 13.0**
7	**Very sophisticated vectorization report including support of source code annotation using Python helper scripts: Introduced by compiler 13.1**

Level 6 Vectorization Report Sample

```
 6:   forall( i= 1:n)
 7:     a(i)= a(i)-b(i)*d(i)
 8:     c(i)= a(i)+c(i)
 9:   endforall
10:
11:   do i=1,n
12:      a(i)= a(i)-b(i)*d(i)
13:      c(i)= a(i)+c(i)
14:   enddo
```

-vec-report3

```
l2.f90(7): (col. 5) remark: LOOP WAS VECTORIZED.
l2.f90(8): (col. 5) remark: LOOP WAS VECTORIZED.
l2.f90(11): (col. 3) remark: LOOP WAS VECTORIZED.
```

-vec-report6

```
l2.f90(7): (col. 5) remark: vectorization support: unroll factor set to 2.
l2.f90(7): (col. 5) remark: LOOP WAS VECTORIZED.
l2.f90(8): (col. 5) remark: vectorization support: unroll factor set to 2.
l2.f90(8): (col. 5) remark: LOOP WAS VECTORIZED.
l2.f90(11): (col. 3) remark: vectorization support: unroll factor set to 2.
l2.f90(11): (col. 3) remark: LOOP WAS VECTORIZED.
```

Level 7 Vectorization Reporting

Provides very detailed and powerful vectorization reporting including summary statistic and source code annotation

- Details and required Python scripts available here
- Not documented in compiler manual yet !
- Python release must be 2.6.5 or younger

Sample usage:

```
ifort -c -vec-report7 loop.f90  2>&1 |
./vecanalysis/vecanalysis.py -list
```

This results in statistics displayed and an annotated source file `loop_ver.f90` to be created

Some Intel-specific Extensions

- Available today / Composer XE 2013 SP1
  - Btrace: Get stack trace even in case stack corrupted
  - PDBX: Parallel Debug Extensions - Data Race Detection
  - TSX: Intel® Transactional Memory Extension support
  - Pointer Checking integration
- Being worked on for future release
  - Full (Intel-) FORTRAN support as provided by Intel® IDB
  - Full VLA (variable length array) debugging
    - In collaboration with community ("GBD Archer project") and Redhat
    - Needed for FORTRAN but also for C99

Note: No own, specific debugger GUI as for Intel® IDB

- But Eclipse-based GUI integration

New Instruction Support

3rd Generation Intel® Core™architecture code name Ivy Bridge:

- Extension core-avx-i for –x switches
- Enables support for all intrinsics introduced by Ivy Bridge (some 6) including RDRAND and 16 bit floating point instructions
- No implicit code generation for new instructions

4th Generation Intel® Core™ architecture code name Haswell:

- Extension core-avx2 for –x switches
- Enables intrinsic support (like for Intel® TSX) and implicit code generation
- FMA instruction will be used by default but can be disabled using switch –no-fma
  - -valuable for FP consistence / rounding issues

New Instruction Support : Broadwell

RDSEED	provide reliable seeds for pseudo-random number generator
ADCX, ADOX	large integer arithmetic addition
PREFETCHW	extending SW prefetch

- Supported via intrinsics :
  - Intel Compilers 13.0 Update 2
  - GNU GCC 4.8

Manual Code Dispatch: A Note

- Intel ® TSX is an optional feature for Haswell
  - Not all models will have this feature
  - Thus two CPU "guard" names are/will be available

    core_4th_gen_avx

    core_4th_gen_avx_tsx

```
__declspec(cpu_specific(core_4th_gen_avx))
void dispatch_func()
{
    printf("\Code here can assume HSW NI but not TSX\n");
}


__ declspec(cpu_specific(core_4th_gen_avx_tsx))
void dispatch_func()
{
    printf("\Code here can assume TSX to be available\n");
}
```

Intel® Visual Fortran Composer XE 2013 with IMSL* for Windows*

- Same 13.1 compiler that is available separately
  - Highly optimizing Fortran compiler featuring scalable multi-threading with OpenMP, and introducing coarray Fortran (part of the Fortran 2008 standard), including new parallelism models
  - VAX Fortran and Compaq Visual Fortran compatibility
  - In addition to IMSL, it includes Intel® Math Kernel Library (Intel® MKL)
- Rogue Wave IMSL* 6.0 Math Library highlights
  - Over 1,000 mathematical and statistical algorithms for developers of Fortran applications
  - Support for shared memory and distributed memory computing environments
  - High performance linear programming optimizer
  - ScaLAPACK integration for MPI, LAPACK integration for SMP
  - New probability density functions and inverses
  - Time series and forecasting additions
  - New Deployment Licensing terms and pricing

Intel® Composer XE for Linux*
Release 4th Sept 2013

Intel® Composer XE 2013 SP1 for Linux* (32/64 bit)	
• Intel® C++ Composer XE Intel® C++ Compiler XE for applications (icc / icpc / idb / gdb-ia)	2013 SP1 14.0.0.080 Build 20130728
• Intel® Fortran Composer XE Intel® C++ Compiler XE for applications (ifort / idbc)	2013 SP1 14.0.0.080 Build 20130728
• Intel® Integrated Performance Primitives (Intel® IPP)	8.0 Update 1
• Intel® Math Kernel Library (Intel® MKL)	11.1
• Intel® Threading Building Blocks (Intel® TBB)	4.1 Update 3
• MPSS package for linking to Intel® Xeon Phi™	2013 Update 3.3

- The 13.0 compilers (Intel® C++ and Fortran Compiler XE 13.0) introduced a large list of new features. All following updates improve stability & compatibility.
- Latest "minor" version is 14.0 (released 4th Sept. 2013)
- Note: Intel® Debugger is deprecated with the next major version.
  - But still needed for full Fortran debugging support

Intel® Composer XE for Windows* / OS X*

Intel® Composer XE 2013 SP1 for Windows* (32/64 bit)	
• Intel® C++ Composer XE Intel® C++ Compiler XE for applications (icl)	2013 SP1 14.0.0.103 Build 20130728
• Intel® Visual Fortran Composer XE (incl. Microsoft Visual Studio Shell and Libraries*) Intel® Visual Fortran Compiler XE for applications (ifort)	2013 SP1 14.0.0.013 Build 20130728
• Intel® Integrated Performance Primitives (Intel® IPP)	8.0 Update 1
• Intel® Math Kernel Library (Intel® MKL)	11.1
• Intel® Threading Building Blocks (Intel® TBB)	4.2
• MPSS package for linking to Intel® Xeon Phi™	2013 Update 3.1

Intel® Composer XE 2013 SP1 for OS X* (32/64 bit)	
• Intel® C++ Composer XE (ICC/ICPC/IDB) Intel® C++ Compiler XE for applications (icc / icpc / idb)	2013 SP1 14.0.0.074 Build 20130728
• Intel® Fortran Composer XE (IFORT/IDB) Intel® C++ Compiler XE for applications (ifort / idb)	2013 SP1 14.0.0.074 Build 20130728

- OS X* versions do not contain Intel® Integrated Performance Primitives!
- Development decoupled from Linux*/Windows* products (see change in version)

Key Files Supplied with Compilers

Windows*

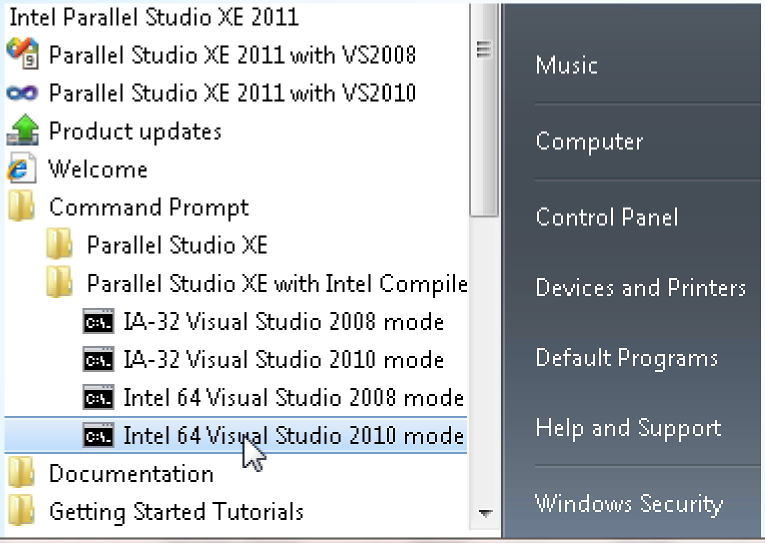
- Intel compiler
  - `icl.exe`, `ifort.exe`: C/C++ compiler, Fortran compiler drivers
  - `mcpcom.exe`, `fortcom.exe`: C/C++ or Fortran Compiler
  - `icl.cfg`, `ifort.cfg`: Default compiler options
  - `compilervars.bat`: Setup command window build environment (C/C++ and Fortran)
- Linker driver
  - `xilink.exe`: Invokes `link.exe`
- Intel include files, libraries
- Re-distributable files
  - `<install-dir>\ComposerXE-2011\redist\`

Linux*, Mac OS* X

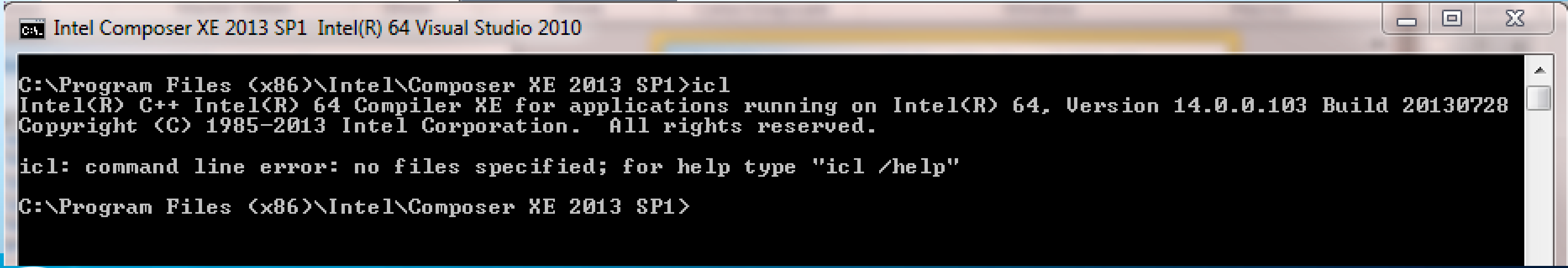
- Intel compiler
  - `icc`, `ifort`: C/C++ compiler, Fortran compiler
  - `mcpcom`, `fortcom`: C/C++ or Fortran Compiler
  - `compilervars.(c)sh`: Source scripts to setup the complete compiler/debugger/libraries environment (C/C++ and Fortran)
- Linker driver
  - `xild`: Invokes `ld`
- Intel include files, libraries
- Intel debugger
  - `idbc` (Command Line) Debugger, `idb` (GUI) Debugger (Linux* only)
- GDB* Enhancements
  - `gdb-ia` with btrace support and data race detection

Command Line Build Environment - Windows

- Sets the environment variables for command line builds – common for C/C++ and Fortran

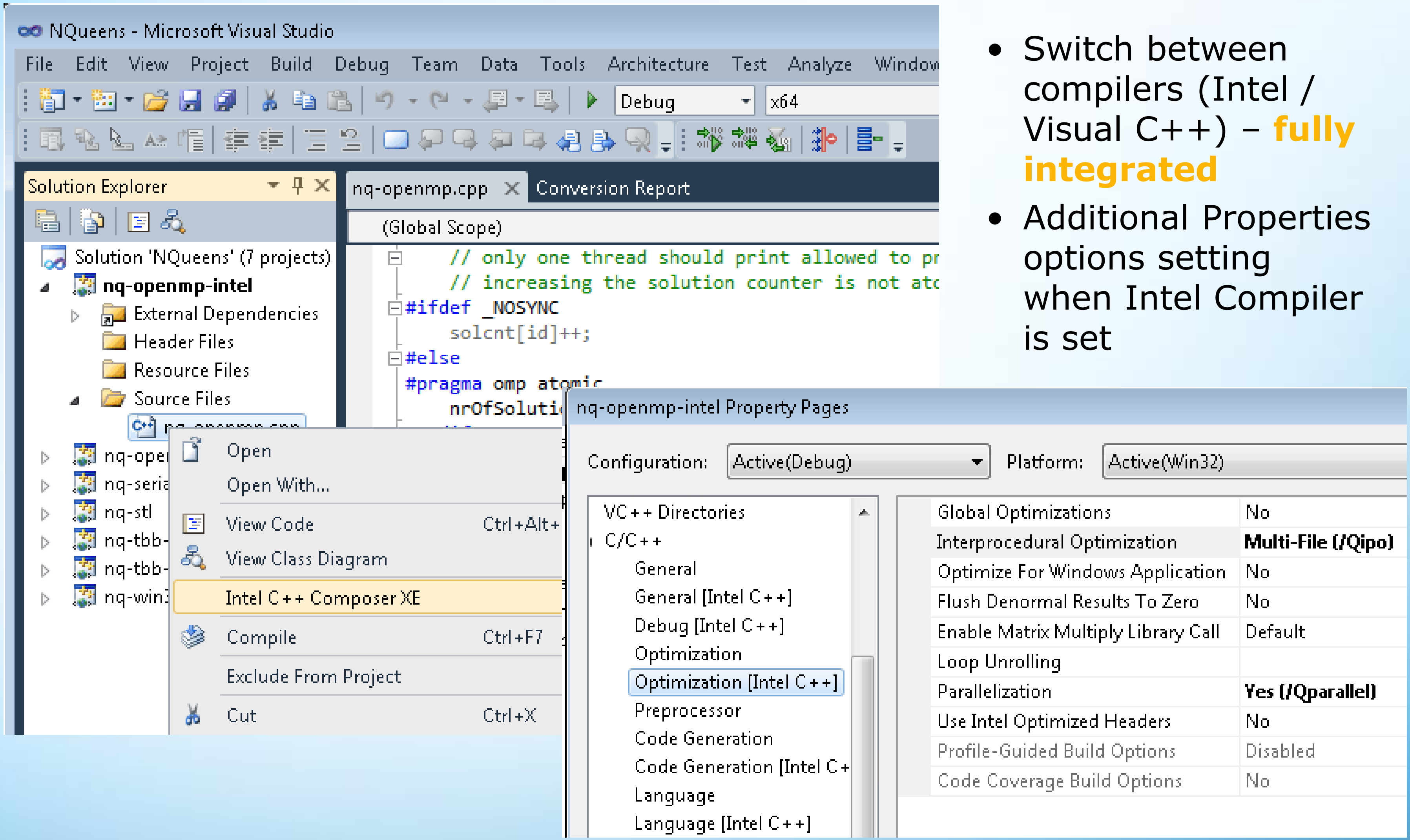

Intel Parallel Studio XE 2011
Parallel Studio XE 2011 with VS2008
Parallel Studio XE 2011 with VS2010
Product updates
Welcome
Command Prompt
Parallel Studio XE
Parallel Studio XE with Intel Compile
IA-32 Visual Studio 2008 mode
IA-32 Visual Studio 2010 mode
Intel 64 Visual Studio 2008 mode
Intel 64 Visual Studio 2010 mode
Documentation
Getting Started Tutorials
Music
Computer
Control Panel
Devices and Printers
Default Programs
Help and Support
Windows Security

- Run C/C++/Fortran compilers from command line



```
C:\Program Files (x86)\Intel\Composer XE 2013 SP1>icl
Intel(R) C++ Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 14.0.0.103 Build 20130728
Copyright (C) 1985-2013 Intel Corporation.  All rights reserved.

icl: command line error: no files specified; for help type "icl /help"

C:\Program Files (x86)\Intel\Composer XE 2013 SP1>
```

Visual Studio* 2010/2012 Integration



```
    // only one thread should print allowed to pr
    // increasing the solution counter is not ato
#ifdef _NOSYNC
    solcnt[id]++;
#else
#pragma omp atomic
    nrOfSoluti
```

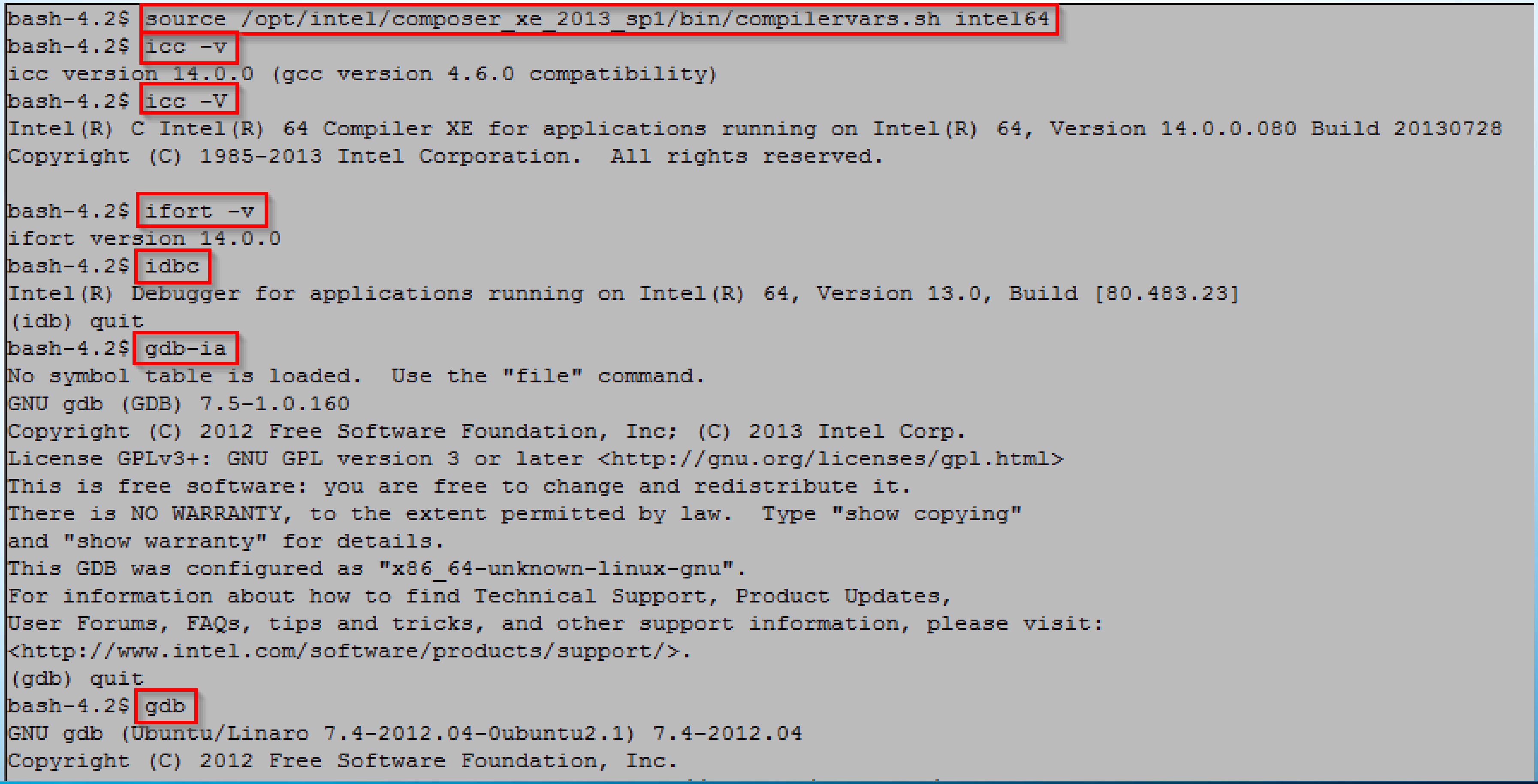
Configuration: Active(Debug)   Platform: Active(Win32)

Global Optimizations	No
Interprocedural Optimization	**Multi-File (/Qipo)**
Optimize For Windows Application	No
Flush Denormal Results To Zero	No
Enable Matrix Multiply Library Call	Default
Loop Unrolling	
Parallelization	**Yes (/Qparallel)**
Use Intel Optimized Headers	No
Profile-Guided Build Options	Disabled
Code Coverage Build Options	No

- Switch between compilers (Intel / Visual C++) – **fully integrated**
- Additional Properties options setting when Intel Compiler is set

Command Line Build Environment - Linux

- Sets the environment variables for command line builds – common for C/C++ and Fortran



```
bash-4.2$ source /opt/intel/composer_xe_2013_sp1/bin/compilervars.sh intel64
bash-4.2$ icc -v
icc version 14.0.0 (gcc version 4.6.0 compatibility)
bash-4.2$ icc -V
Intel(R) C Intel(R) 64 Compiler XE for applications running on Intel(R) 64, Version 14.0.0.080 Build 20130728
Copyright (C) 1985-2013 Intel Corporation.  All rights reserved.

bash-4.2$ ifort -v
ifort version 14.0.0
bash-4.2$ idbc
Intel(R) Debugger for applications running on Intel(R) 64, Version 13.0, Build [80.483.23]
(idb) quit
bash-4.2$ gdb-ia
No symbol table is loaded.  Use the "file" command.
GNU gdb (GDB) 7.5-1.0.160
Copyright (C) 2012 Free Software Foundation, Inc; (C) 2013 Intel Corp.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
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/>.
(gdb) quit
bash-4.2$ gdb
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
```

Optimization Notice

Common Optimization Switches

	Windows*	**Linux***
Disable optimization	/Od	-O0
Optimize for speed (no code size increase)	/O1	-O1
Optimize for speed (default)	/O2	-O2
High-level loop optimization	/O3	-O3
Create symbols for debugging	/Zi	-g
Multi-file inter-procedural optimization	/Qipo	-ipo
Profile guided optimization (multi-step build)	/Qprof-gen /Qprof-use	-prof-gen -prof-use
Optimize for speed across the entire program	/fast (same as: /O3 /Qipo /Qprec-div- /QxHost)	-fast (same as: -ipo –O3 -no-prec-div -static -xHost)
OpenMP 3.0 support	/Qopenmp	-openmp
Automatic parallelization	/Qparallel	-parallel

Compiler Reports – Optimization Report

Compiler switch:

`/Qopt-report-phase[:phase]` (Windows*)

`-opt-report-phase[=phase]` (Linux*, Mac OS* X)

,`phase`' can be:

- `ipo_inl` - Interprocedural Optimization Inlining Report
- `ilo` – Intermediate Language Scalar Optimization
- `hpo` – High Performance Optimization
- `hlo` – High-level Optimization
- `all` – All optimizations (not recommended, output too verbose)

Control the level of detail in the report:

`/Qopt-report[0|1|2|3]` (Windows*)

`-opt-report[0|1|2|3]` (Linux*, MacOS* X)

If you do not specify the option, no optimization report is being generated; if you do not specify the level (i.e. /Qopt-report, -opt-report) level 2 is being used by the compiler.

Optimization Report Example

```
icc -O3 -opt-report-phase=hlo -opt-report-phase=hpo
icl /O3 /Qopt-report-phase:hlo /Qopt-report-phase:hpo
```

```
…
LOOP INTERCHANGE in loops at line: 7 8 9
Loopnest permutation ( 1 2 3 ) --> ( 2 3 1 )
…
Loop at line 8 blocked by 128
Loop at line 9 blocked by 128
Loop at line 10 blocked by 128
…
Loop at line 10 unrolled and jammed by 4
Loop at line 8 unrolled and jammed by 4
…
…(10)…   loop was not vectorized: not inner loop.
…(8)…    loop was not vectorized: not inner loop.
…(9)…    PERMUTED LOOP WAS VECTORIZED
…
```

icc –vec-report2 (icl /Qvec-report2) for just the vectorization report

Case study: Matrix calculation

```
#include <stdio.h>
#include <time.h>

#define NUM 1024
main()
{
	clock_t start, stop;
	int num;

	// initialize the arrays with data
	init_arr(3,-2,1,a);
	init_arr(-2,1,3,b);

	//start timing the matrix multiply code
	printf("NUM:%d\n",NUM);
	start = clock();
	multiply_d(a,b,c);
	stop = clock();

	// print simple test case of data to be sure multiplication is correct
	if (NUM < 5) {
		print_arr("a", a);
		print_arr("b", b);
		print_arr("c", c);
	}

	// print elapsed time
	printf("Elapsed time = %lf seconds\n",((double)(stop - start)) / CLOCKS_PER_SEC);

}
```

Software & Services Group
Developer Products Division

Optimization Notice

Case study: Matrix calculation

```
//routine to initialize an array with data
void init_arr(double row, double col, double off, double a[][NUM])
{
	int i,j;

	for (i=0; i< NUM;i++) {
		for (j=0; j<NUM;j++) {
				a[i][j] = row*i+col*j+off;
		}
	}
}

// routine to print out contents of small arrays
void print_arr(char * name, double array[][NUM])
{
	int i,j;

	printf("\n%s\n", name);
	for (i=0;i<NUM;i++){
		for (j=0;j<NUM;j++) {
				printf("%g\t",array[i][j]);
		}
		printf("\n");
	}
}
```

Case study: Matrix calculation

```
static double  a[NUM][NUM], b[NUM][NUM], c[NUM][NUM];
void multiply_d(double a[][NUM], double  b[][NUM], double  c[][NUM]);

void multiply_d(double a[][NUM], double  b[][NUM], double  c[][NUM])
{
    int i,j,k;
    double temp;
    for(i=0;i<NUM;i++) {
        for(j=0;j<NUM;j++) {
            for(k=0;k<NUM;k++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

Case study: Matrix calculation

- Compile the program matrix.c/matrix.f90 without optimization and with /O1 – compare results (e.g.):

```
icl /Od matrix.c /o noopt_matrix.exe
icl /O1 matrix.c /o O1_matrix.exe
```

- Run and time the executable

```
noopt_matrix.exe          >>    30,25 sec.
O1_matrix.exe             >>     9,6 sec.
```

[Intel® Core™ i7, 4-Core 64-bit, 1,6GHz]

High-Level Optimizer (HLO)

- Compiler switches:
  `/O2, /O3` (Windows*), `-O2, -O3` (Linux*)
- Loop level optimizations
  - loop unrolling, cache blocking, prefetching
- More aggressive dependency analysis
  - Determines whether or not it's safe to reorder or parallelize statements
- Scalar replacement
  - Goal is to reduce memory by replacing with register references

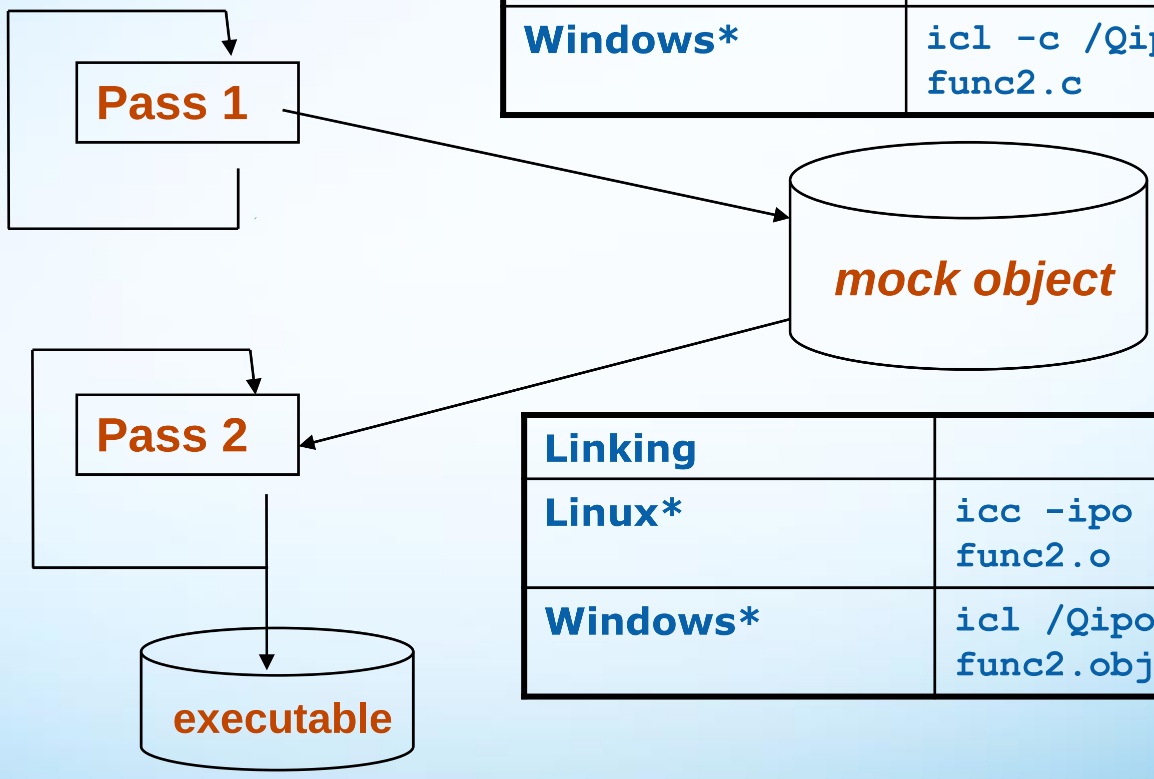
Interprocedural Optimizations (IPO)
Multi-pass Optimization

- Interprocedural optimizations performs a static, topological analysis of your application!
- ip: Enables inter-procedural optimizations for current source file compilation
- ipo: Enables inter-procedural optimizations across files
  - Can inline functions in separate files
  - Especially many small utility functions benefit from IPO

Windows*	Linux*
/Qip	-ip
/Qipo	-ipo

Enabled optimizations:
- Procedure inlining (reduced function call overhead)
- Interprocedural dead code elimination, constant propagation and procedure reordering
- Enhances optimization when used in combination with other compiler features

Interprocedural Optimizations (IPO)
Usage: Two-Step Process



Compiling	
Linux*	`icc -c -ipo main.c func1.c func2.c`
Windows*	`icl -c /Qipo main.c func1.c func2.c`

Linking	
Linux*	`icc -ipo main.o func1.o func2.o`
Windows*	`icl /Qipo main.o func1.o func2.obj`

Inlining Functions

When the compiler inlines a function call, the function's code gets inserted into the caller's instruction stream

Benefits:

- Reducing overhead of calling a function
  - writing the registers and parameters to/from stack
  - restore the registers when the function returns.
- Improving performance because the optimizer can procedurally integrate the called function and can do better optimizations
  - sub-expression elimination
  - copy propagation

Drawbacks:

- Overuse of inlining can actually make programs slower. Depending on a function's size, inlining it can cause the code size to increase, resulting in more cache misses and more pressure on the instruction cache
- The speed benefits of inline functions tend to diminish as the function grows in size. At some point the overhead of the function call becomes small compared to the execution of the function body, and the benefit is lost.

Techniques for Inlining Functions

- Compiler Switches
  - Increase information provided to the compiler
    `-ipo, -prof_use` (Linux), `/Qipo, /Qprof-use` (Windows)
  - Change Compiler Heuristics
    `-inline-factor=n` (default=100), `/Qinline-factor=n`
    `-inline-level=0|1|2, /ob0|1|2`
- Inlining source code features
  - Microsoft* C/C++
    Keywords: `inline, __inline, __forceinline`
  - GCC C/C++
    `__attribute__((always_inline))`
    `__attribute__((noinline))`

Case study: Include IPO

- Compile the program matrix.c/matrix.f90 with optimization and with including /Qipo – compare results:

```
icl /O3 /Qipo matrix.c /Qopt-report-phase:ipo /o O3_ipo_matrix.exe
icl /O3 /Qipo /QxHost matrix.c /o O3_ipo_xHost_matrix.exe
```

[Don't expect much with IPO on this sample (why?)]

- Run and time the executable

```
O2_novec_matrix.exe      >>      2,12 sec.
O3_novec_matrix.exe      >>      1,6 sec.
O2_matrix.exe            >>      0,98 sec.
O3_matrix.exe            >>      0,83 sec.
O3_ipo_matrix.exe        >>      0,82 sec.
O3_ipo_xHost_matrix.exe  >>      0,75 sec.
```

[Intel® Core™ i7, 4-Core 64-bit, 1,6GHz]

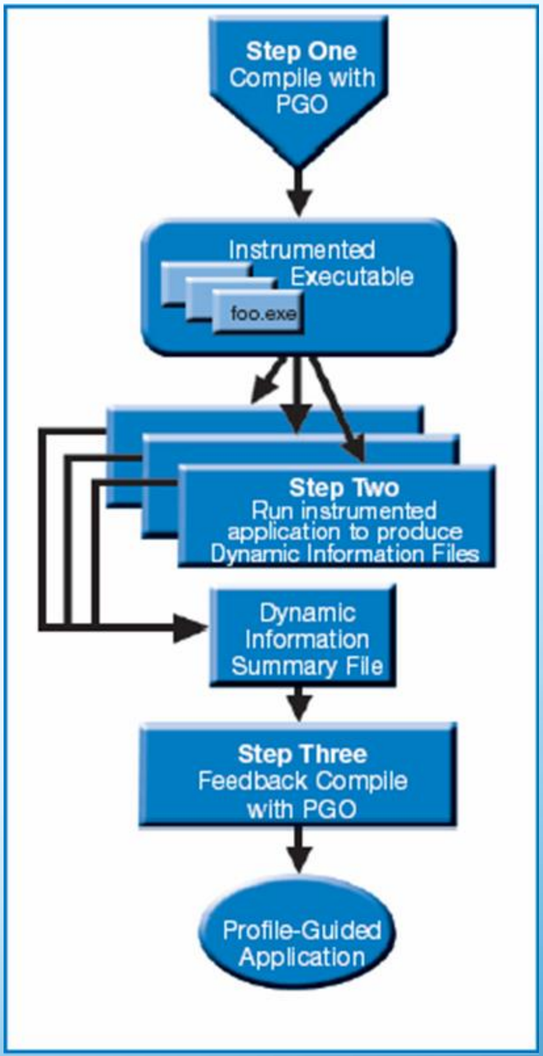
Profile-Guided Optimizations (PGO)

- Static analysis leaves many questions open for the optimizer like:
  - How often is x > y
  - What is the size of count
  - Which code is touched how often

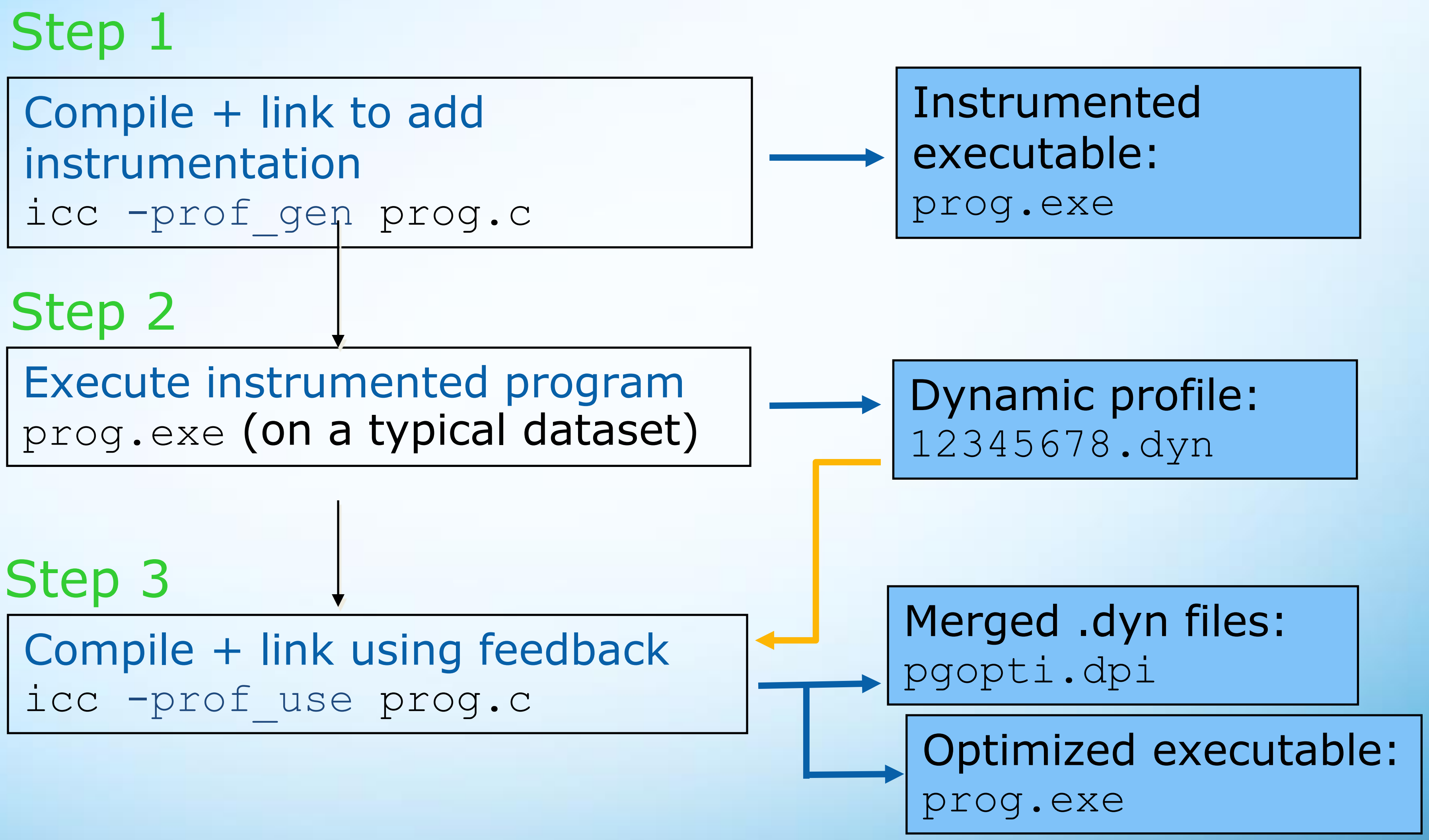
```
if (x > y)
        do_this();
 else
        do that();
```

```
for(i=0; i<count; ++I
do_work();
```

- Use execution-time feedback to guide (final) optimization
- Enhancements with PGO:
  - More accurate branch prediction
  - Basic block movement to improve instruction cache behavior
  - Better decision of functions to inline (help IPO)
  - Can optimize function ordering
  - Switch-statement optimization
  - Better vectorization decisions


Step One
Compile with PGO
Instrumented Executable
foo.exe
Step Two
Run instrumented application to produce Dynamic Information Files
Dynamic Information Summary File
Step Three
Feedback Compile with PGO
Profile-Guided Application

PGO Usage: Three Step Process



Step 1

Compile + link to add instrumentation

```
icc -prof_gen prog.c
```

Instrumented executable:

```
prog.exe
```

Step 2

Execute instrumented program

`prog.exe` (on a typical dataset)

Dynamic profile:

```
12345678.dyn
```

Step 3

Compile + link using feedback

```
icc -prof_use prog.c
```

Merged .dyn files:

```
pgopti.dpi
```

Optimized executable:

```
prog.exe
```

Case study: Use PGO

- Compile the program matrix.c/matrix.f90 with /Qprof-gen /Qprof-use – compare results:

```
icl /Qprof-gen matrix.c /o pgen_matrix.exe
pgen_matrix.exe      [Why does this take so long?]
icl /Qprof-use /O3 /QxHost matrix.c /o puse_matrix.exe
```

- Run and time the executable

```
O2_novec_matrix.exe      >>     2,12 sec.
O3_novec_matrix.exe      >>     1,6 sec.
O2_matrix.exe            >>     0,98 sec.
O3_matrix.exe            >>     0,83 sec.
O3_ipo_matrix.exe        >>     0,82 sec.
O3_ipo_xHost_matrix.exe  >>     0,75 sec.
puse_matrix.exe          >>     0,32 sec.
```

[Intel® Core™ i7, 4-Core 64-bit, 1,6GHz]

Automatic Compiler Vectorization
Processor Specific Optimizations

- Compiler vectorizer switch:
  `/Qvec` (Windows*), `-vec` (Linux*)
  - Implicitly included with optimization switch /O2, -O2 or higher
  - Switch off autovectorizer explicitly with /Qvec-, -vec-
  - Automatically generates vector instructions using the multiple SIMD instruction set extensions SSE/SSE2/SSE3/SSSE3/SSE4.x/AVX
  - Operates at once with one instruction on, e.g.:
    4 float / 2 double values or 4 x 32-bit / 8 x 16-bit integers
- Processor specific extensions switches:
  Default switch is –msse2 (Windows), `/arch:SSE2` (Linux)
  - Activated implicitly with optimization switch /O2, -O2 or higher

  `/arch:<extension>` (Microsoft compatible), `-m<extension>` (Linux*, Mac OS* X)
  - No Intel processor check (but with sse4.x)
  - No Intel specific optimizations

  `/Qx<SIMDextension>, -x<SIMDextension>`
  - Code generation for specific target hardware
  - Enables additional extensions for Intel processors

  `/Qax<SIMDextension>, -ax<SIMDextension>`
  - Code generation for specific target hardware and extensions for Intel processors
  - Autodispatch switch **a** for generating additional default code path

  Special switch `/QxHost, -xHost`
  - Automatically gives access to all the latest features of the processor you are working on
- `-mtune=<ARCH>` option on Linux*/OS X* to specify cpu targeting without generating instructions exclusive to that cpu

Processor-specific Compiler Switches

(Intel® 64 and IA-32)

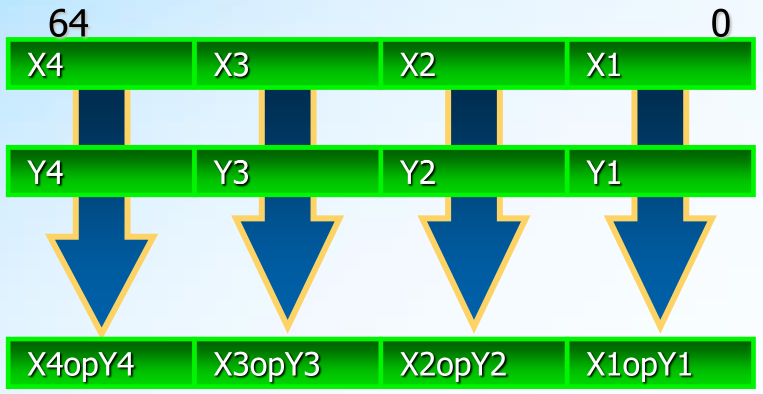
Windows	Linux
/Qxsse2	-xsse2
/Qxsse3	-xsse3
/Qxssse3	-xssse3
/Qxsse4.1	-xsse4.1
/Qxsse4.2	-xsse4.2
/QxAVX	-xavx
/QxHOST	-xHost
/Qxsse3_atom	-xsse3_ATOM

- Implies an Intel CPU ID check
- Runtime message if try to run on unsupported processor

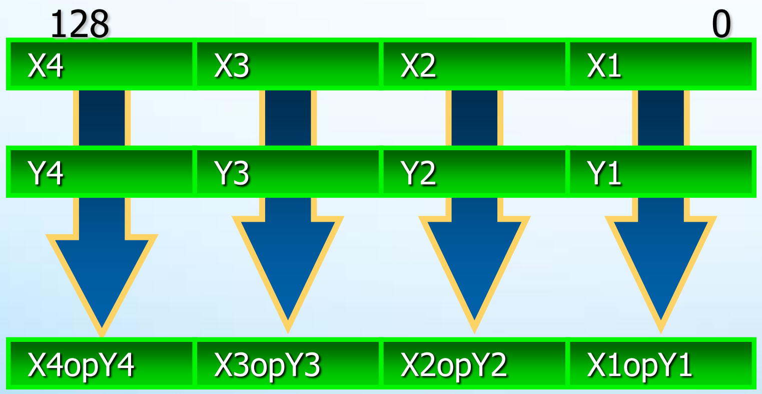
Windows	Linux
/arch:IA32	-mia32
/arch:sse2	-msse2 (default!)
/arch:sse3	-msse3
/arch:ssse3	-mssse3

- No CPU ID check (but with sse4.x)
- Intel and non-Intel processors
- Illegal instruction error if run on unsupported processor
- At least Pentium4 required

SIMD Types in Processors from Intel [1]

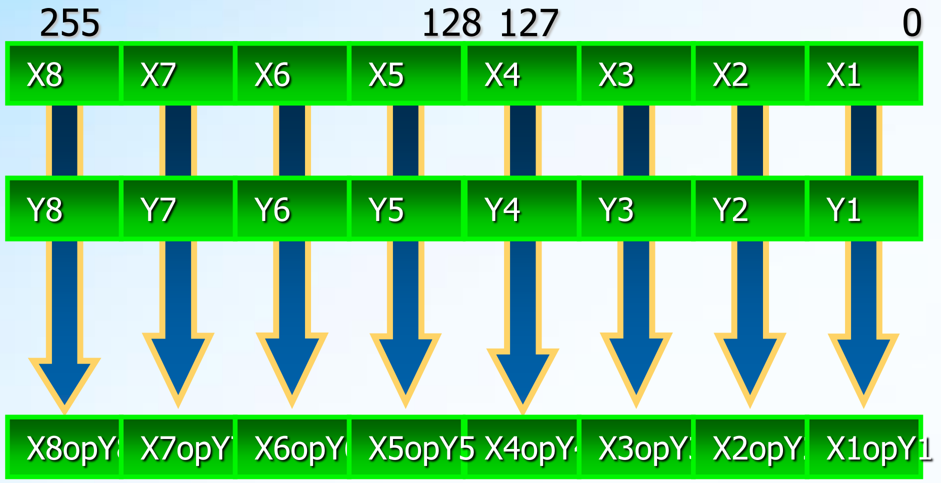


MMX™
Vector size: 64bit
Data types: 8, 16 and 32 bit integers
VL: 2,4,8
For sample on the left: Xi, Yi 16 bit integers



Intel® SSE
Vector size: 128bit
Data types:
  8,16,32,64 bit integers
  32 and 64bit floats
VL: 2,4,8,16
Sample: Xi, Yi bit 32 int / float

SIMD Types in Processors from Intel [2]



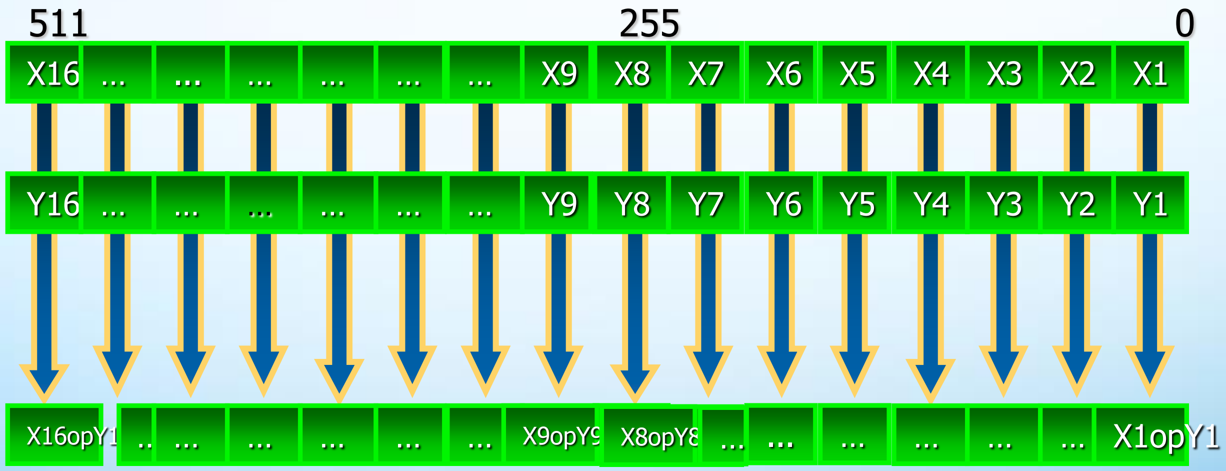
Intel® AVX

Vector size: 256bit

Data types: 32 and 64 bit floats

VL: 4, 8, 16

Sample: Xi, Yi 32 bit int or float



511
255
0
X16 … … … … … … X9 X8 X7 X6 X5 X4 X3 X2 X1
Y16 … … … … … … Y9 Y8 Y7 Y6 Y5 Y4 Y3 Y2 Y1
X16opY16 … X9opY9 X8opY8 … X1opY1

Intel® MIC

Vector size: 512bit

Data types:

32 and 64 bit integers

32 and 64bit floats

(some support for 16 bits floats)

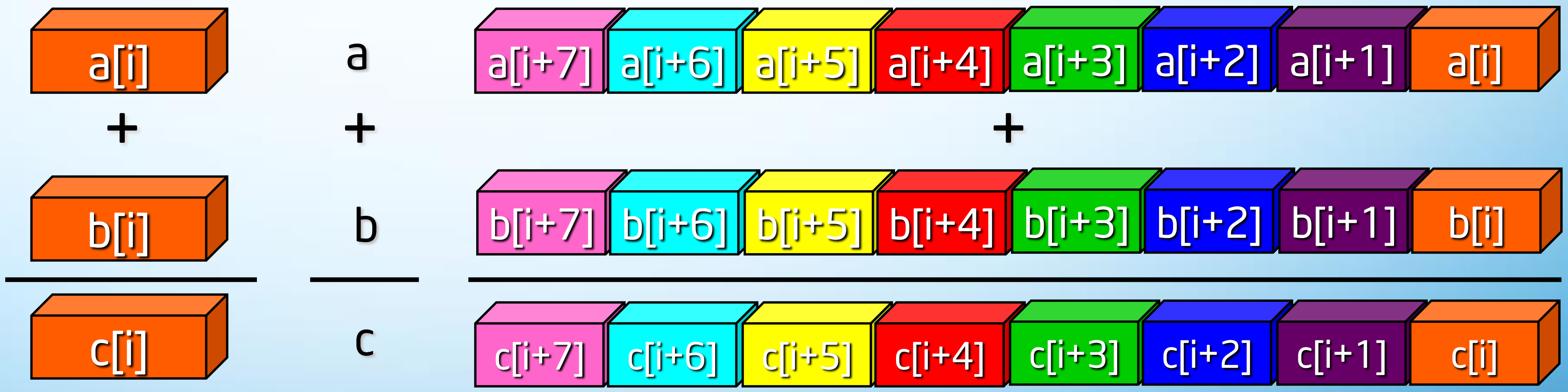
VL: 8,16

Sample: 32 bit float

Auto-Vectorization
SIMD – Single Instruction Multiple Data

- Scalar mode
  - one instruction produces one result
- SIMD processing
  - with SSE or AVX instructions
  - one instruction can produce multiple results

```
for (i=0;i<=MAX;i++)
      c[i]=a[i]+b[i];
```



Scalar and Packed SSE Instructions

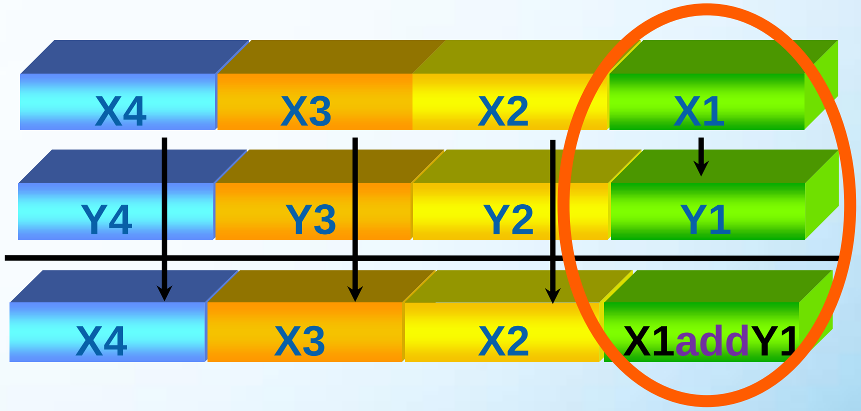
The “vector” form of SSE instructions operating on multiple data elements simultaneously are called packed – thus vectorized SSE code means use of packed instructions

- Most of these instructions have a scalar version too operating only one element only

addss Scalar Single-FP Add

single precision FP data

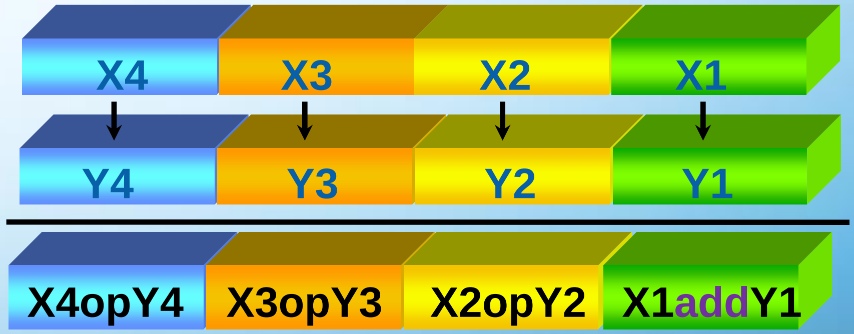
scalar execution mode



addps Packed Single-FP Add

single precision FP data

packed execution mode



References

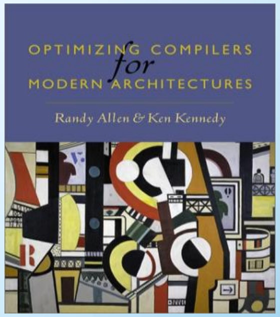
[1] Aart Bik: "The Software Vectorization Handbook"
  – http://www.intel.com/intelpress/sum_vmmx.htm

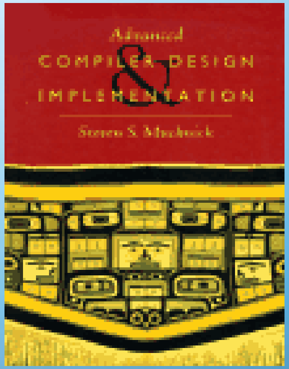
[2] Randy Allen, Ken Kennedy: "Optimizing Compilers for Modern Architectures: A Dependence-based Approach"

[3] Steven S. Muchnik, "Advanced Compiler Design and Implementation"

[4] Intel Software Forums, Knowledge Base, White Papers, Tools Support - see http://software.intel.com
Sample Articles :

- software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers/
- software.intel.com/en-us/articles/requirements-for-vectorizable-loops/
- software.intel.com/en-us/articles/performance-tools-for-software-developers-intel-compiler-options-for-sse-generation-and-processor-specific-optimizations/


OPTIMIZING COMPILERS for MODERN ARCHITECTURES
Randy Allen & Ken Kennedy



Compiler Reports – Vectorization Report

Compiler switch:

`/Qvec-report<n>` (Windows)

`-vec-report<n>` (Linux)

Set diagnostic level dumped to stdout

n=*0*: No diagnostic information
n=1: (Default) Loops successfully vectorized
n=*2*: Loops not vectorized – and the reason why not
n=3: Adds dependency Information
n=4: Reports only non-vectorized loops
n=5: Reports only non-vectorized loops and adds dependency info

- Note:
  For Linux, `-opt_report_phase hpo` provides additional diagnostic information for vectorization

Case study: Use auto-vectorizer

- Compile the program matrix.c/matrix.f90 with optimization /O[2|3], vectorizer and create report:

```
icl /O2 matrix.c /Qvec-report2 /o O2_matrix.exe
icl /O3 matrix.c /Qvec-report2 /o O3_matrix.exe
icl /O3 /QxHost /Qvec-report2 matrix.c /o O3_xHost_matrix.exe
icl /fast matrix.c /o fast_matrix.exe
```

[Do all loops vectorize? Is there room for improvement?]

- Run and time the executable

```
noopt_matrix.exe         >>     30,25 sec.
O1_matrix.exe            >>      9,6 sec.
O2_novec_matrix.exe      >>      2,12 sec.
O3_novec_matrix.exe      >>      1,6 sec.
O2_matrix.exe            >>      0,98 sec.
O3_matrix.exe            >>      0,83 sec.
O3_xHost_matrix.exe      >>      0,78 sec.
fast_matrix.exe          >>      0,78 sec.
```

[Intel® Core™ i7, 4-Core 64-bit, 1,6GHz]

Optimization Notice

Compiling for Intel® AVX

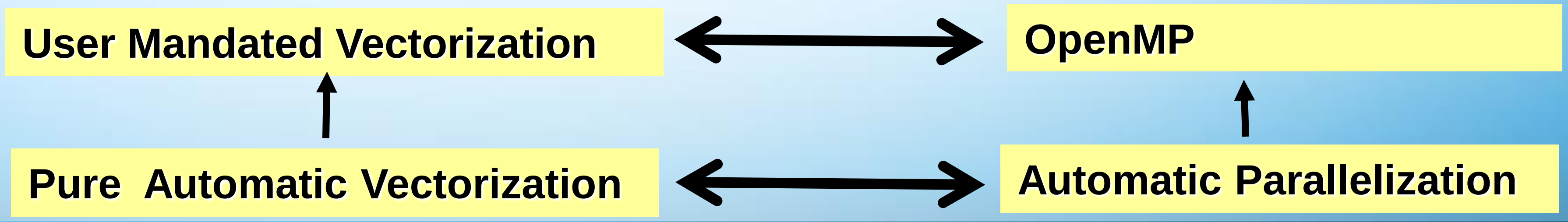
- Compile with –xavx (/Qxavx on Windows*)
  - Vectorization works just as for SSE
- More FP loops can be vectorized than with SSE
  - Individually masked data elements
  - More powerful data rearrangement instructions
- Can create both SSE and AVX code paths in one binary
- Switches Linux: -axavx Windows: /Qaxavx
  - use additional –x (/Qx) switches to modify the default SSE code path; e.g –axavx –xsse4.2 to target SSE4.2 ("Nehalem") and AVX ("SandyBridge")

User-Mandated Vectorization

User-mandated vectorization is based on a new **SIMD Directive** (or “pragma”)

- The SIMD directive provides additional information to compiler to enable vectorization of loops (at this time only inner loop)
- Supplements automatic vectorization but differently to what traditional directives like IVDEP, VECTOR ALWAYS do, the SIMD directive is more a command than a hint or an assertion: The compiler heuristics are completely overwritten as long as a clear logical fault is not being introduced

Relationship similar to OpenMP versus automatic parallelization:


User Mandated Vectorization
OpenMP
Pure Automatic Vectorization
Automatic Parallelization

Positioning of SIMD Vectorization
A Cilk Plus Feature

Fully automatic vectorization

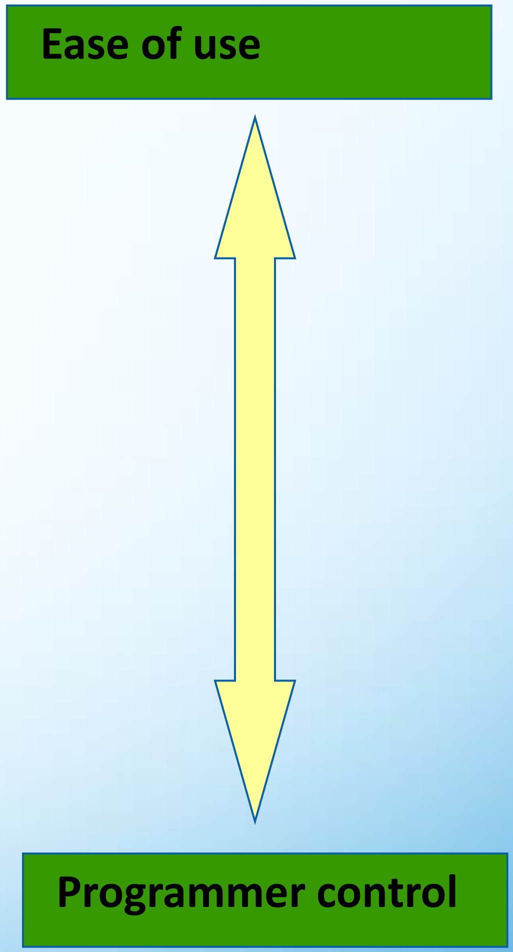
Auto vectorization hints (#pragma ivdep)

User Mandated Vectorization
(SIMD Directive)

SIMD intrinsic class (F32vec4 add)

Vector intrinsic (mm_add_ps())

ASM code (addps)



SIMD Directive Notation

C/C++: #pragma simd [clause [,clause] …]

Fortran: !DIR$ SIMD [clause [,clause] …]

Without any clause, the directive enforces vectorization of the (innermost) loop

Sample:

```
void add_fl(float *a, float *b, float *c, float *d, float *e, int n)
{
  #pragma simd
  for (int i=0; i<n; i++)
    a[i] = a[i] + b[i] + c[i] + d[i] + e[i];
}
```

Without SIMD directive, vectorization will fail since there are too many pointer references to do a run-time check for overlapping (compiler heuristic)

Clauses of SIMD Directive

`vectorlength(n1 [,n2] …)`

- n1, n2, … must be 2,4,8 or 16: The compiler can assume a vectorization for a vector length of n1, n2, … to be save

`private(v1, v2, …)`

- variables private to each iteration; initial value is broadcast to all private instances, and the last value is copied out from the last iteration instance.

`linear(v1:step1, v2:step2, …)`

- for every iteration of original scalar loop, v1 is incremented by step1, … etc. Therefore it is incremented by step1 *(vector length) for the vectorized loop.

`reduction(operator:v1, v2, …)`

- v1 etc are reduction variables for operation "operator"

`[no]assert`

- reaction in case vectorization fails: Print a warning only (noassert, the default) or treat failure as error and stop compilation

Sample: SIMD Directive Vectorlength Clause

```
Void foo(float *a, float *b, float *c, int n)
{
   for (int k=0; k<n; k++)    c[k] = a[k] + b[k];
}
```

Due to the overlapping nature of array accesses from the different call sites, it might not be semantically correct to use restrict keyword or IVDEP directive (there are dependencies between iterations for one call)

But it might be true for all calls, that e.g 4 consecutive iterations can be executed in parallel without violating any dependencies

```
void foo(float *a, float *b, float *c, int n)
{
   #pragma simd vectorlength(4)
   for (int k=0; k<n; k++ )  c[k] = a[k] + b[k];
}
```

Auto-Parallelization

- Compiler automatically translates portions of serial code into equivalent multithreaded code with using these options:
  `/Qparallel, -parallel`

- The auto-parallelizer analyzes the dataflow of loops and generates multithreaded code for those loops which can safely and efficiently be executed in parallel.

- The auto-parallelizer report can provide information about program sections that could be parallelized by the compiler. Compiler switch:
  `/Qpar-report:0|1|2|3`
  `-par-report0|1|2|3`
  ‚0‛ is report disabled, ‚3‛ maximum diagnostics level

Case study: Auto-parallelization

- Compile the program matrix.c/matrix.f90 with /Qparallel /Qpar-report2 with and without additional vectorization/optimizations:

```
icl /Qparallel /Qpar-report2 /Qvec- /o par_novec_matrix.exe
icl /Qparallel /Qpar-report2 /o par_matrix.exe
icl /Qparallel /Qpar-report2 /O3 /Qvec-report2 /Qipo /QxHost /o
   par_vec_matrix.exe
icl /O3 /Qparallel /Qprof-use /O3 /Qipo /QxHost /o
   par_vec_puse_matrix.exe
```

[Examine the parallel report. Is the program well parallelized? What are the possible pitfalls with thread-level parallelization?]

- Run and time the executable

```
O3_ipo_xHost_matrix.exe    >>      0,75 sec.
puse_matrix.exe            >>      0,32 sec.
par_novec_matrix.exe       >>      0,90 sec.
par_matrix.exe             >>      0,35 sec.
par_vec_matrix.exe         >>      0,35 sec.
par_vec_puse_matrix.exe    >>      0,12 sec.
```

[Intel® Core™ i7, 4-Core 64-bit, 1,6GHz]

Auto-Parallelization vs. Auto-Vectorization

Explain difference

- **Auto-vectorization** which is DLP (Data Level Parallelization) uses wide registers (SSE) for operation of multiple instructions.

- **Auto-parallelization** which is TLP (Thread Level Parallelization) uses processor cores (and hardware threads) for parallelization of serial code.

- Possible pitfalls in auto-parallelization: Concurrent access by more threads to the same memory locations (data race!).

Guided Automatic Parallelization (GAP)

- Feature of the C++/Fortran Compiler that offers advice and, when correctly applied, results in auto-vectorization or auto-parallelization of serial code.
  - Compiler switch `/Qguide` in parallel with `/O2` or higher gives advise for auto-vectorization
  - Compiler switch `/Qguide` in parallel with `/Qparallel` or higher gives advise for auto-parallelization
- Compiler runs in advisor mode
  - Compiler does not generate ANY code with `/Qguide, -guide` – vectorized or not.

Intel® Guided Auto Parallelism (GAP)

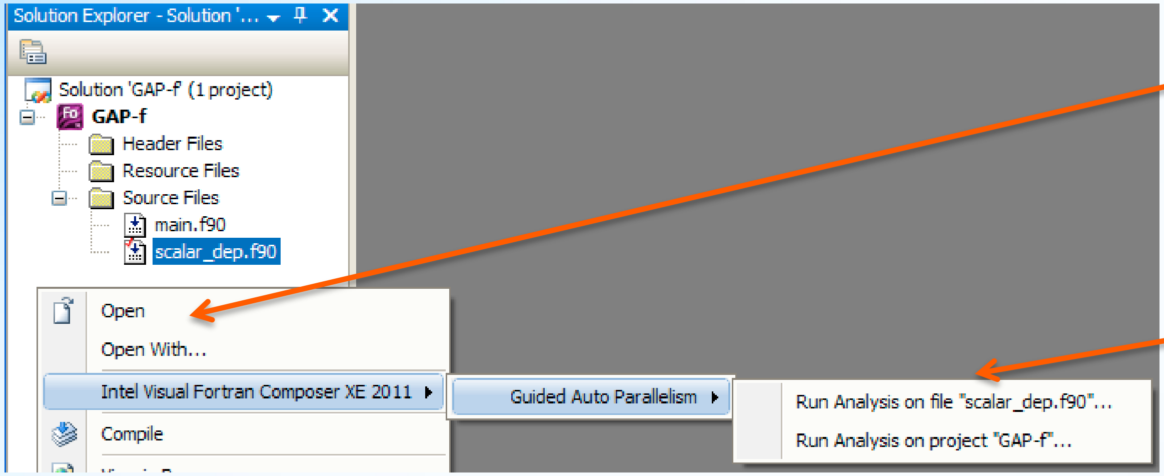
Let the Compiler Tell You What it Needs

- Motivation
  - Effective, simplify way to add parallelism to your applications
  - Use built-in compiler technology to speed parallelism development
- What is GAP?
  - Compiler-based analyzer that provides guidance to developers to change code so it can be compiled to automatically optimize code through vectorization, parallelization, or data transformation
  - Built upon existing auto-vectorization and auto-parallelization technology
- GAP does not
  - Analyze code and find hotspots for threading (see Advisor)
  - Verify threading correctness (use Inspector, Inspector XE)
  - Do any performance/hotspot analysis (use Amplifier, VTune Amplifier XE)

Developer Must Verify Semantics of GAP Recommendations

GAP – How it Works (Windows)

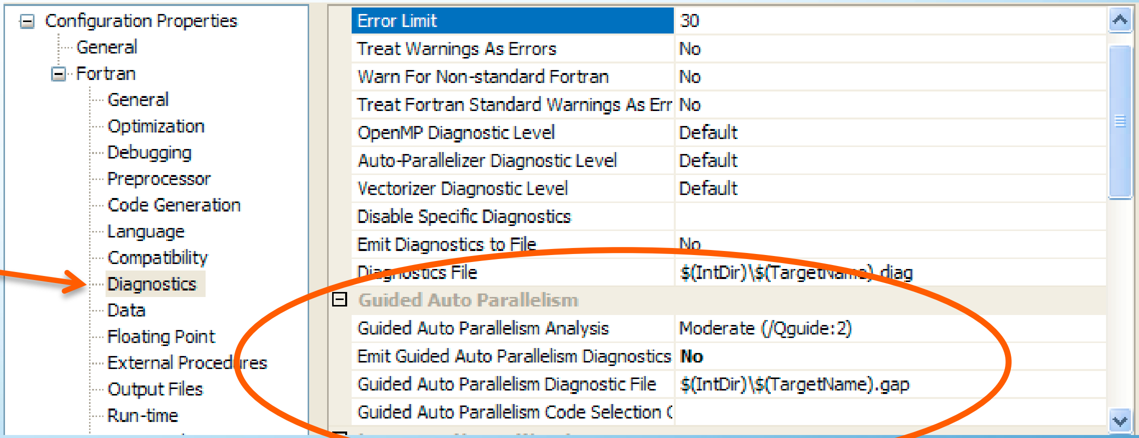
- Windows – right-click or Project Properties
  - GAP analysis appears in Output window


Solution Explorer - Solution '...
Solution 'GAP-f' (1 project)
GAP-f
Header Files
Resource Files
Source Files
main.f90
scalar_dep.f90
Open
Open With...
Intel Visual Fortran Composer XE 2011
Compile
Guided Auto Parallelism
Run Analysis on file "scalar_dep.f90"...
Run Analysis on project "GAP-f"...

Right-click file or project, Fortran Composer pop-up

Choose single-file or whole project

OR you can use the Project Properties, Fortran, Diagnostics property page



Error Limit	30
Treat Warnings As Errors	No
Warn For Non-standard Fortran	No
Treat Fortran Standard Warnings As Err	No
OpenMP Diagnostic Level	Default
Auto-Parallelizer Diagnostic Level	Default
Vectorizer Diagnostic Level	Default
Disable Specific Diagnostics	
Emit Diagnostics to File	No
Diagnostics File	$(IntDir)\$(TargetName).diag
Guided Auto Parallelism	
Guided Auto Parallelism Analysis	Moderate (/Qguide:2)
Emit Guided Auto Parallelism Diagnostics	**No**
Guided Auto Parallelism Diagnostic File	$(IntDir)\$(TargetName).gap
Guided Auto Parallelism Code Selection (	

Vectorization Example [1]

```
void f(int n, float *x, float *y, float *z, float *d1, float *d2)
{
  for (int i = 0; i < n; i++)
     z[i] = x[i] + y[i] – (d1[i]*d2[i]);
}
```

GAP Message:

g.c(6): remark #30536: (LOOP) Add -Qno-alias-args option for better type-based disambiguation analysis by the compiler, if appropriate (the option will apply for the entire compilation). This will improve optimizations such as vectorization for the loop at line 6. [VERIFY] Make sure that the semantics of this option is obeyed for the entire compilation. [ALTERNATIVE] Another way to get the same effect is to add the "restrict" keyword to each pointer-typed formal parameter of the routine "f". This allows optimizations such as vectorization to be applied to the loop at line 6. [VERIFY] Make sure that semantics of the "restrict" pointer qualifier is satisfied: in the routine, all data accessed through the pointer must not be accessed through any other

The compiler guides the user on source-change and on what pragma to insert and on how to determine whether that pragma is correct for this case

Vectorization Example [2]

```
void mul(NetEnv* ne, Vector* rslt
  Vector* den, Vector* flux1,
  Vector* flux2, Vector* num
{
  float *r, *d, *n, *s1, *s2;
  int i;
  r=rslt->data;
  d=den->data;
  n=num->data;
  s1=flux1->data;
  s2=flux2->data;

  for (i = 0; i < ne->len; ++i)
    r[i] = s1[i]*s2[i] +
    n[i]*d[i];
}
```

GAP Messages (simplified):

1. "Use a local variable to store the upper-bound of loop at line 29 (variable:ne->len) if the upper-bound does not change during execution of the loop"
2. "Use "#pragma ivdep" to help vectorize the loop at line 29, if these arrays in the loop do not have cross-iteration dependencies: r, s1, s2, n, d"

-> Upon recompilation, the loop will be vectorized

Parallelization Example

```
#define N 10000
double A[N], B[N];
int bar(int);
void foo(){
  int i;
  for (i=0;i<N;i++)
    A[i] = B[i] * bar(i);
}
```

Compiler invocation:

```
{L}: icc -c funcall.c
       -parallel
       -guide
{W}: icl /c funcall.c
       /Qparallel
       /Qguide
```

GAP Message on Windows:

```
funcall.c(6): remark #30528: (PAR) Add "__declspec(const)" to the
    declaration of routine "bar" in order to parallelize the loop at
    line 12. Alternatively, adding "__declspec(concurrency_safe
    (profitable))" achieves a similar effect.
[VERIFY] Make sure that the routine satisfies the semantics of this
    declaration.
[ALTERNATIVE] Yet another way to help the loop being parallelized is
    to inline the routine with "#pragma forceinline recursive". This
    method does not guarantee parallelization.
```

Data Transformation Example

```
struct S3 {
    int a;
    int b; // hot
    double c[100];
    struct S2 *s2_ptr;
    int d;  int e;
    struct S1 *s1_ptr;
    char *c_p;
    int f; // hot
};
```

```
…
for (ii = 0; ii < N; ii++){
        sp->b = ii;
        sp->f = ii + 1;
        sp++;
}
…
```

```
peel.c(22): remark #30756: (DTRANS) Splitting the structure 'S3' into
    two parts will improve data locality and is highly recommended.
    Frequently accessed fields are 'b, f'; performance may improve by
    putting these fields into one structure and the remaining fields
    into another structure. Alternatively, performance may also improve
    by reordering the fields of the structure. Suggested field order:'b,
    f, s2_ptr, s1_ptr, a, c, d, e, c_p'. [VERIFY] The suggestion is
    based on the field references in current compilation …
```

Compiler Floating Point Model

The Floating Point options allow to control the optimizations on floating-point data. These options can be used to tune the performance, level of accuracy or result consistency.

Accuracy

Produce results that are "close" to the correct value

- –Measured in relative error, possibly ulps (units in the last place)

Reproducibility

Produce consistent results

- –From one run to the next
- –From one set of build options to another
- –From one compiler to another
- –From one platform to another

Performance

Produce the most efficient code possible

- –Default, primary goal of Intel® Compilers

These objectives usually conflict! Wise use of compiler options lets you control the tradeoffs.

Compiler Floating-Point Model

The Floating-Point Compiler Switch

`-fp-model` ***`keyword`*** (Linux*, Mac OS* X)

`/fp:`***`keyword`*** (Windows*)

Lets you choose the FP semantics at a coarse granularity and specify the compiler rules for

- Value safety
- FP expression evaluation
- FPU environment access
- Precise FP exceptions
- FP contractions
- Abrupt underflow (flush to zero)
  - Denormals are set to zero
  - May improve performance, esp. if HW doesn‘t support denormals

Floating-Point Keywords

Controls consistency of floating point results by restricting certain optimizations. Values for *keywords* are

- `fast[=1|2]`; default is `fast=1`
  - Allows „value-unsafe" optimizations (=default)
  - Allows aggressive optimizations at a slight cost in accuracy or consistency.
  - Some additional approximations allowed with `fast=2`
- `precise`
  - Enables only value-safe optimizations on floating point code.
- `source`
  - Implies precise and enables intermediates to be computed in source precision.
  - Source is the recommended form for the majority of situations on processors supporting Intel® 64 and IA-32 platforms when SSE are enabled with /QxSSE2 or higher.

Floating-Point Keywords (2)

- `double`
  - Implies precise and enables intermediates to be computed in double or extended precision.
  - Not avaliable in Intel® Fortran Compilers
- `extended`
  - Rounds intermediate results to 64-bit (extended) precision
  - Enables value safe optimization
- `except`
  - Enables floating point exception semantics
- `strict`
  - Strictest mode of operation, enables both the precise and except options and disables contractions (i.e., precise + strict + disable fma)

The *–fp-model<key>* Switch

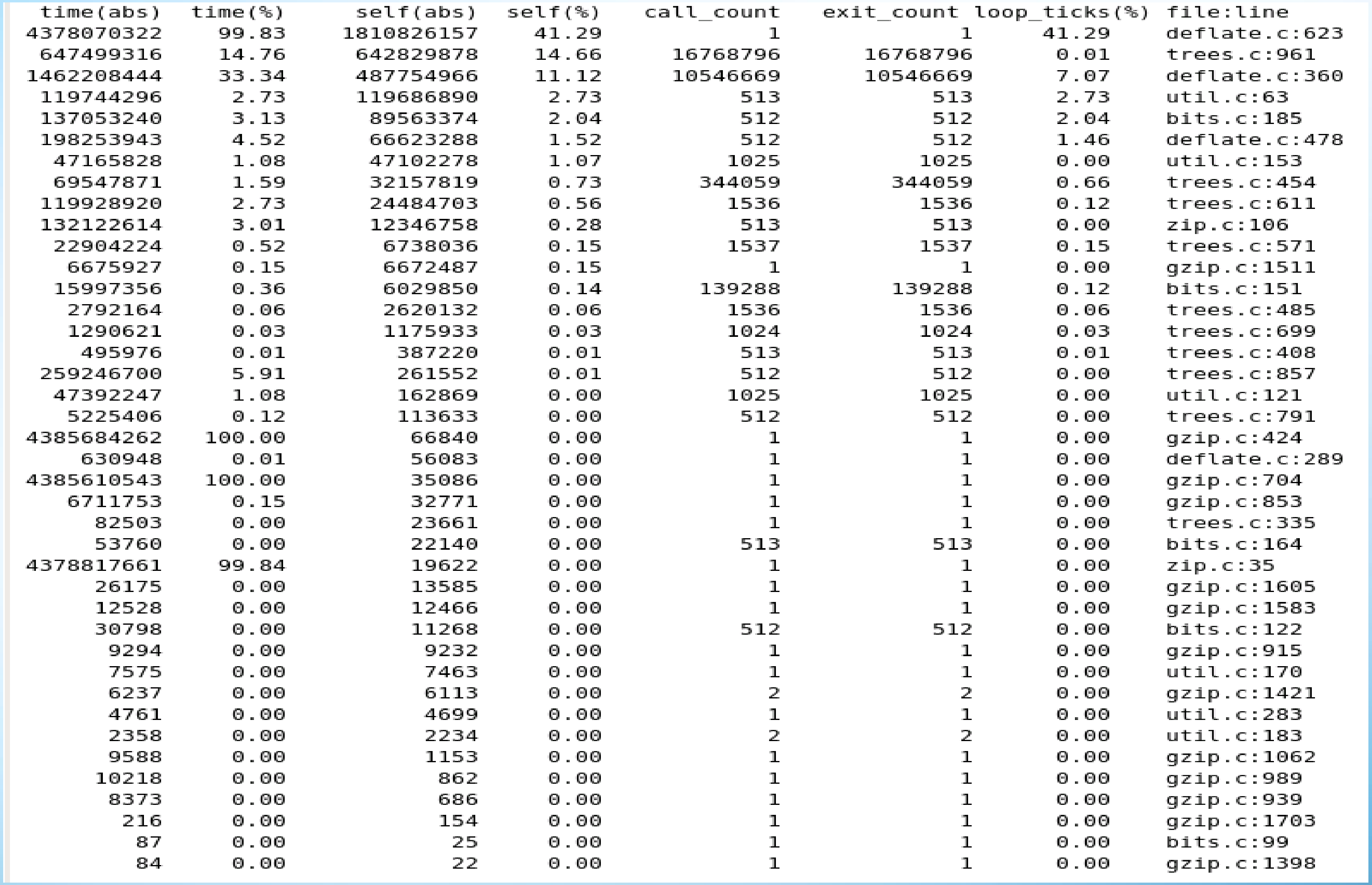
Key	Value Safety	Expression Evaluation	FPU Environ. Access	Precise FP Exceptions	FP contract
precise source double extended	Safe	Varies Source Double Extended	No	No	Yes
strict	Safe	Varies	Yes	Yes	No
fast=1 (default)	Unsafe	Unknown	No	No	Yes
fast=2	Very Unsafe	Unknown	No	No	Yes
except except-	*/** *	* *	* *	Yes No	* *

* These modes are unaffected. –fp-model except[-] only affects the precise FP exceptions mode.
** It is illegal to specify *–fp-model except* in an unsafe value safety mode.

Loop Profiler
Identify Time Consuming Loops/Functions

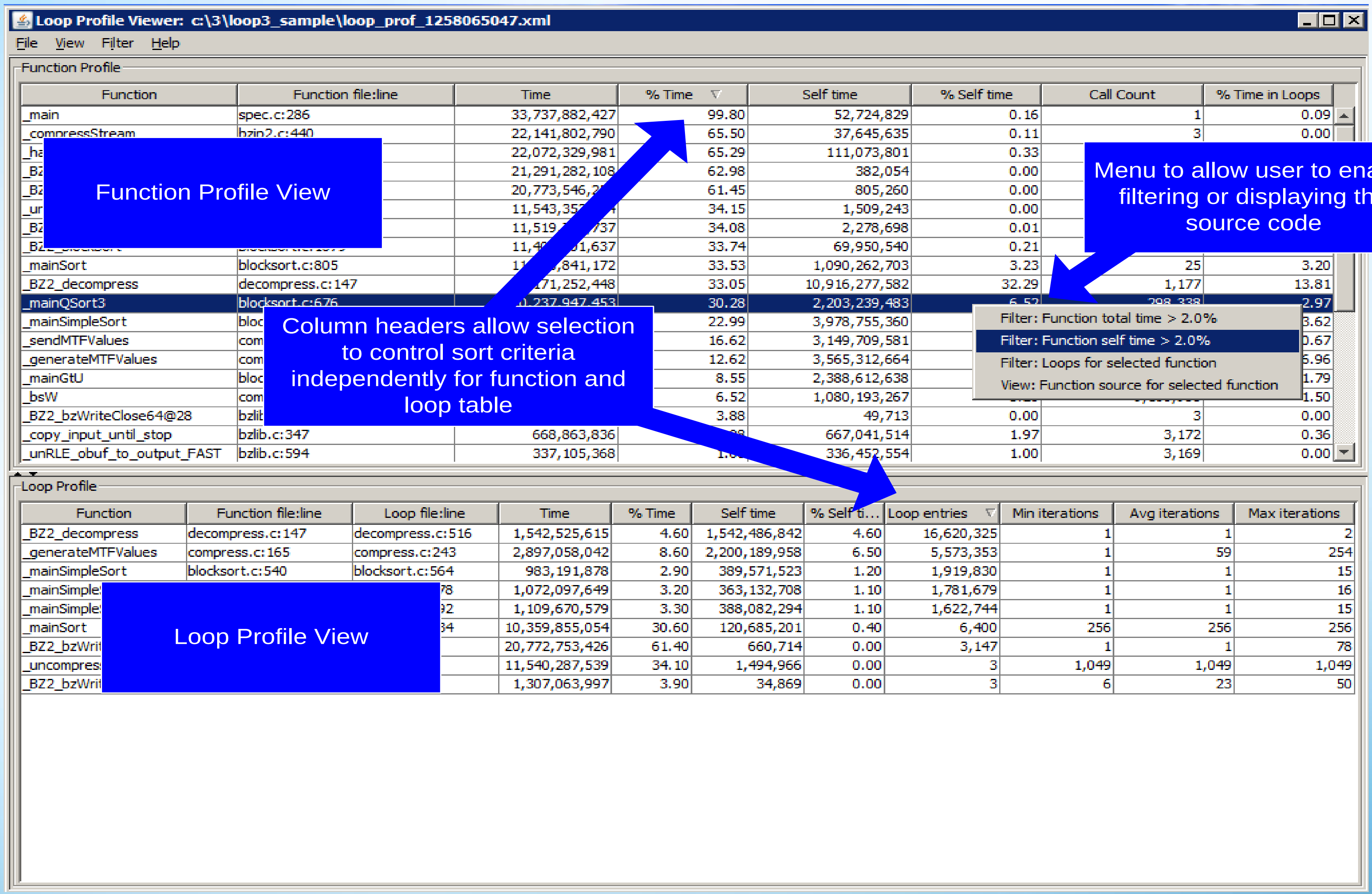
- Compiler switch:
  `/Qprofile-functions, -profile-functions`
  - Insert instrumentation calls on function entry and exit points to collect the cycles spent within the function.
- Compiler switch:
  `/Qprofile-loops=<inner|outer|all>,`
  `-profile-loops= <inner|outer|all>`
  - Insert instrumentation calls for function entry and exit points as well as the instrumentation before and after instrument able loops of the type listed as the option's argument.
- Loop Profiler switches trigger generation of text (.dump) and XML (.xml) output files
  - Invocation of XML viewer on command line:
    `java -jar loopprofviewer.jar <xml datafile>`

Loop Profiler Text Dump (.dump file)



```
  time(abs)  time(%)      self(abs)  self(%)    call_count    exit_count loop_ticks(%) file:line
 4378070322   99.83      1810826157    41.29             1             1     41.29    deflate.c:623
  647499316   14.76       642829878    14.66      16768796      16768796      0.01    trees.c:961
 1462208444   33.34       487754966    11.12      10546669      10546669      7.07    deflate.c:360
  119744296    2.73       119686890     2.73           513           513      2.73    util.c:63
  137053240    3.13        89563374     2.04           512           512      2.04    bits.c:185
  198253943    4.52        66623288     1.52           512           512      1.46    deflate.c:478
   47165828    1.08        47102278     1.07          1025          1025      0.00    util.c:153
   69547871    1.59        32157819     0.73        344059        344059      0.66    trees.c:454
  119928920    2.73        24484703     0.56          1536          1536      0.12    trees.c:611
  132122614    3.01        12346758     0.28           513           513      0.00    zip.c:106
   22904224    0.52         6738036     0.15          1537          1537      0.15    trees.c:571
    6675927    0.15         6672487     0.15             1             1      0.00    gzip.c:1511
   15997356    0.36         6029850     0.14        139288        139288      0.12    bits.c:151
    2792164    0.06         2620132     0.06          1536          1536      0.06    trees.c:485
    1290621    0.03         1175933     0.03          1024          1024      0.03    trees.c:699
     495976    0.01          387220     0.01           513           513      0.01    trees.c:408
  259246700    5.91          261552     0.01           512           512      0.00    trees.c:857
   47392247    1.08          162869     0.00          1025          1025      0.00    util.c:121
    5225406    0.12          113633     0.00           512           512      0.00    trees.c:791
 4385684262  100.00           66840     0.00             1             1      0.00    gzip.c:424
     630948    0.01           56083     0.00             1             1      0.00    deflate.c:289
 4385610543  100.00           35086     0.00             1             1      0.00    gzip.c:704
    6711753    0.15           32771     0.00             1             1      0.00    gzip.c:853
      82503    0.00           23661     0.00             1             1      0.00    trees.c:335
      53760    0.00           22140     0.00           513           513      0.00    bits.c:164
 4378817661   99.84           19622     0.00             1             1      0.00    zip.c:35
      26175    0.00           13585     0.00             1             1      0.00    gzip.c:1605
      12528    0.00           12466     0.00             1             1      0.00    gzip.c:1583
      30798    0.00           11268     0.00           512           512      0.00    bits.c:122
       9294    0.00            9232     0.00             1             1      0.00    gzip.c:915
       7575    0.00            7463     0.00             1             1      0.00    util.c:170
       6237    0.00            6113     0.00             2             2      0.00    gzip.c:1421
       4761    0.00            4699     0.00             1             1      0.00    util.c:283
       2358    0.00            2234     0.00             2             2      0.00    util.c:183
       9588    0.00            1153     0.00             1             1      0.00    gzip.c:1062
      10218    0.00             862     0.00             1             1      0.00    gzip.c:989
       8373    0.00             686     0.00             1             1      0.00    gzip.c:939
        216    0.00             154     0.00             1             1      0.00    gzip.c:1703
         87    0.00              25     0.00             1             1      0.00    bits.c:99
         84    0.00              22     0.00             1             1      0.00    gzip.c:1398
```

Loop Profiler Data Viewer GUI (copy from sl. 46)


Loop Profile Viewer: c:\3\loop3_sample\loop_prof_1258065047.xml
File View Filter Help
Function Profile View
Column headers allow selection to control sort criteria independently for function and loop table
Menu to allow user to enable filtering or displaying the source code
Filter: Function total time > 2.0%
Filter: Function self time > 2.0%
Filter: Loops for selected function
View: Function source for selected function
Loop Profile View

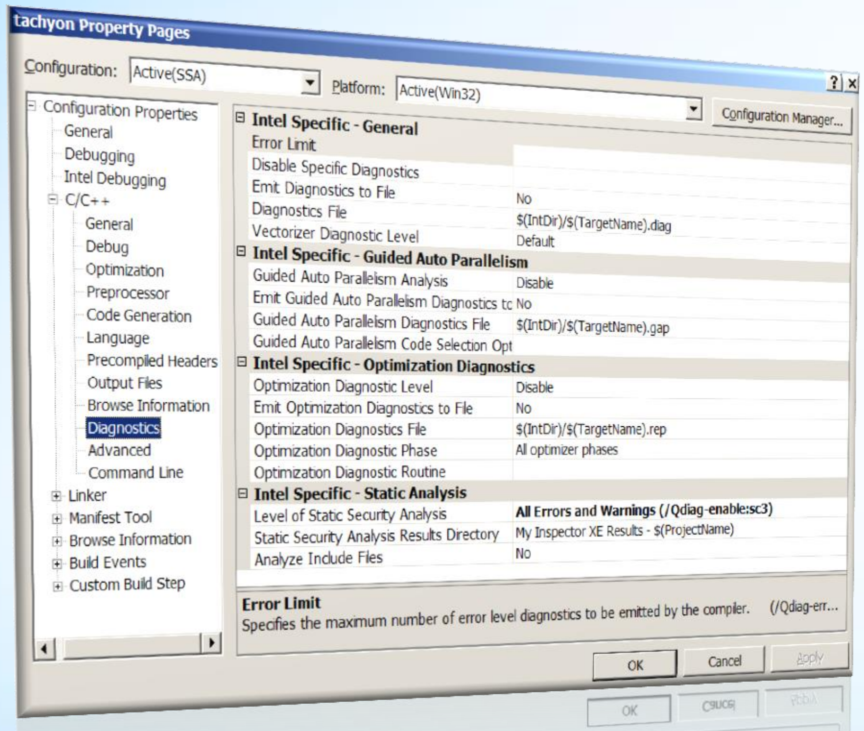
Function Profile

Function	Function file:line	Time	% Time	Self time	% Self time	Call Count	% Time in Loops
_main	spec.c:286	33,737,882,427	99.80	52,724,829	0.16	1	0.09
_compressStream	bzip2.c:440	22,141,802,790	65.50	37,645,635	0.11	3	0.00
_ha		22,072,329,981	65.29	111,073,801	0.33		
_BZ		21,291,282,108	62.98	382,054	0.00		
_BZ		20,773,546,2	61.45	805,260	0.00		
_ur		11,543,35	34.15	1,509,243	0.00		
_BZ		11,519 737	34.08	2,278,698	0.01		
_BZ		11,4 1,637	33.74	69,950,540	0.21		
_mainSort	blocksort.c:805	11 ,841,172	33.53	1,090,262,703	3.23	25	3.20
_BZ2_decompress	decompress.c:147	171,252,448	33.05	10,916,277,582	32.29	1,177	13.81
_mainQSort3	blocksort.c:676	0,237,947,453	30.28	2,203,239,483	6.52	298,338	2.97
_mainSimpleSort	bloc		22.99	3,978,755,360			3.62
_sendMTFValues	com		16.62	3,149,709,581			0.67
_generateMTFValues	com		12.62	3,565,312,664			5.96
_mainGtU	bloc		8.55	2,388,612,638			1.79
_bsW	com		6.52	1,080,193,267			1.50
_BZ2_bzWriteClose64@28	bzlit		3.88	49,713	0.00	3	0.00
_copy_input_until_stop	bzlib.c:347	668,863,836		667,041,514	1.97	3,172	0.36
_unRLE_obuf_to_output_FAST	bzlib.c:594	337,105,368	1.	336,452,554	1.00	3,169	0.00

Loop Profile

Function	Function file:line	Loop file:line	Time	% Time	Self time	% Self ti...	Loop entries	Min iterations	Avg iterations	Max iterations
_BZ2_decompress	decompress.c:147	decompress.c:516	1,542,525,615	4.60	1,542,486,842	4.60	16,620,325	1	1	2
_generateMTFValues	compress.c:165	compress.c:243	2,897,058,042	8.60	2,200,189,958	6.50	5,573,353	1	59	254
_mainSimpleSort	blocksort.c:540	blocksort.c:564	983,191,878	2.90	389,571,523	1.20	1,919,830	1	1	15
_mainSimple		78	1,072,097,649	3.20	363,132,708	1.10	1,781,679	1	1	16
_mainSimple		92	1,109,670,579	3.30	388,082,294	1.10	1,622,744	1	1	15
_mainSort		34	10,359,855,054	30.60	120,685,201	0.40	6,400	256	256	256
_BZ2_bzWrit			20,772,753,426	61.40	660,714	0.00	3,147	1	1	78
_uncompres			11,540,287,539	34.10	1,494,966	0.00	3	1,049	1,049	1,049
_BZ2_bzWrit			1,307,063,997	3.90	34,869	0.00	3	6	23	50

Static Security Analysis – (SSA)



Configuration: Active(SSA) Platform: Active(Win32)

- Configuration Properties
  - General
  - Debugging
  - Intel Debugging
  - C/C++
    - General
    - Debug
    - Optimization
    - Preprocessor
    - Code Generation
    - Language
    - Precompiled Headers
    - Output Files
    - Browse Information
    - Diagnostics
    - Advanced
    - Command Line
  - Linker
  - Manifest Tool
  - Browse Information
  - Build Events
  - Custom Build Step

Intel Specific - General	
Error Limit	
Disable Specific Diagnostics	
Emit Diagnostics to File	No
Diagnostics File	$(IntDir)/$(TargetName).diag
Vectorizer Diagnostic Level	Default
Intel Specific - Guided Auto Parallelism	
Guided Auto Parallelism Analysis	Disable
Emit Guided Auto Parallelism Diagnostics tc	No
Guided Auto Parallelism Diagnostics File	$(IntDir)/$(TargetName).gap
Guided Auto Parallelism Code Selection Opt	
Intel Specific - Optimization Diagnostics	
Optimization Diagnostic Level	Disable
Emit Optimization Diagnostics to File	No
Optimization Diagnostics File	$(IntDir)/$(TargetName).rep
Optimization Diagnostic Phase	All optimizer phases
Optimization Diagnostic Routine	
Intel Specific - Static Analysis	
Level of Static Security Analysis	**All Errors and Warnings (/Qdiag-enable:sc3)**
Static Security Analysis Results Directory	My Inspector XE Results - $(ProjectName)
Analyze Include Files	No

Error Limit
Specifies the maximum number of error level diagnostics to be emitted by the compiler. (/Qdiag-err...

Enhanced Application Security
Catch defects early in lifecycle

- Detects over 250 different kinds of errors and security risks, such as:
  - Buffer overruns and uninitialized variables
  - Heap corruption and bad pointer usage
  - Unchecked use of input data
  - Error prone usage of libraries and language features
  - Arithmetic overflow and divide by zero
- True global analysis crossing subroutine and file boundaries
- Easy set up and usability, delivers immediate results
- Intuitive GUI & Command Line interface for Windows* and Linux
- Displays problems and associated source code
- Tracks state associated with issues, even as source evolves and line numbers change
- Provides filters and sorting to organize results and reduce clutter

C, C++, Fortran for Windows* and Linux support

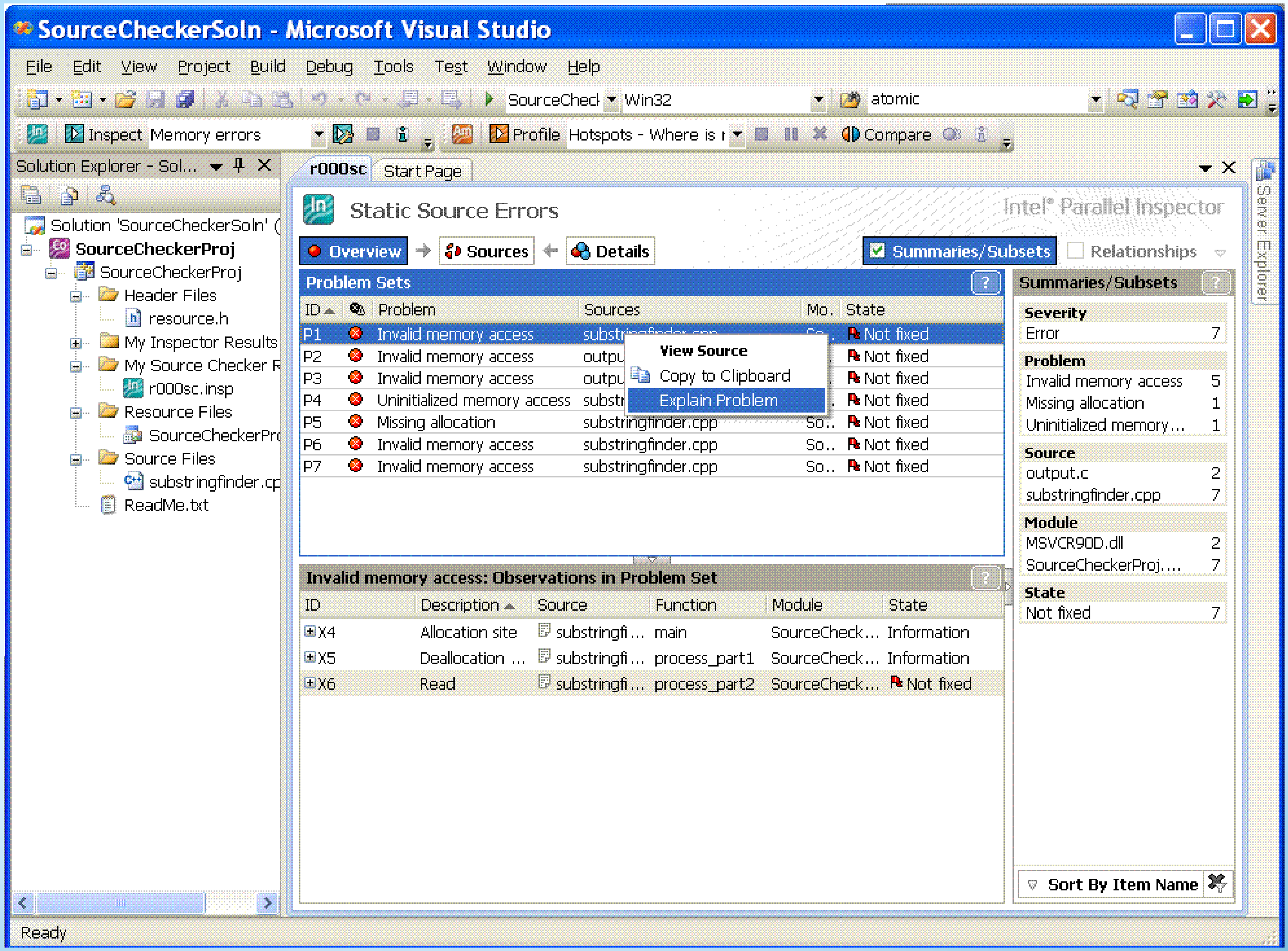
Static Security Analysis (SSA)

- SSA uses the compiler technology to generate analysis information
  - Compiler switch:

    ```
    /Qdiag-enable:sc{[1|2|3]}
    -diag-enable sc{[1|2|3]}
    ```
- Viewing SSA analysis data requires the **Intel® Inspector XE** to manage the analysis
  - Inspector automatically started after compilation if installed on the same computer
  - SSA results can be viewed on different computer where the Inspector is available
    - Data collection using compiler front end
    - Data filtering
    - Display of results
    - Interactive source vie

NOTE: Using SSA requires a Parallel Studio XE license!

SSA Results - Evaluation with Intel Inspector XE



Problem Sets

ID	Problem	Sources	Mo.	State
P1	Invalid memory access	substringfinder.cpp	So..	Not fixed
P2	Invalid memory access	outpu		Not fixed
P3	Invalid memory access	outpu		Not fixed
P4	Uninitialized memory access	substr		Not fixed
P5	Missing allocation	substringfinder.cpp	So..	Not fixed
P6	Invalid memory access	substringfinder.cpp	So..	Not fixed
P7	Invalid memory access	substringfinder.cpp	So..	Not fixed

Invalid memory access: Observations in Problem Set

ID	Description	Source	Function	Module	State
X4	Allocation site	substringfi...	main	SourceCheck...	Information
X5	Deallocation ...	substringfi...	process_part1	SourceCheck...	Information
X6	Read	substringfi...	process_part2	SourceCheck...	Not fixed

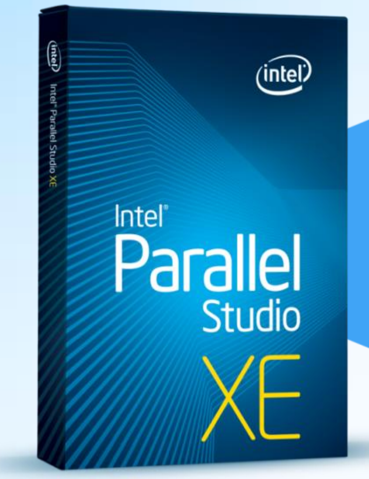
Summaries/Subsets

Severity	
Error	7
Problem	
Invalid memory access	5
Missing allocation	1
Uninitialized memory...	1
Source	
output.c	2
substringfinder.cpp	7
Module	
MSVCR90D.dll	2
SourceCheckerProj....	7
State	
Not fixed	7

Optimization Notice

Pointer Checker

Requires a Studio XE License!



- A key feature of Intel® Parallel Studio XE 2013
- Catches out-of-bounds memory accesses through pointers
  - ➢ Identifies and reports before memory corruption occurs!
  - ➢ Buffer overruns /overflows
  - ➢ Dangling Pointers
    - o memory accesses through freed pointers
- Designed for use during application debugging and testing
- Security benefits from catching vulnerabilities prior to product release.

- Enabled via compile time switches.
- User API allows control over what happens when a violation is detected
- Implemented mostly in a runtime library which is automatically linked in by the compiler
- No change to structure layout or ABIs

Key Benefit: Enable Incrementally
Pointer Checker can be enabled on a single file, group of files or all files. Pointer Checker enabled code and non-enabled code can coexist!

Pointer Checker - *Why?*

C/C++ pointers have well defined semantics for memory range access:

```
// Pointer Bounds
p = malloc(size);
// Lower Bound(p) is (char *)p
// Upper Bound(p) is lower_bound(p) + size - 1
```

Buffer overflow/overrun anomaly:

- Violation of memory safety
- Data corruption
- Erratic program behavior
- Breach of system security
- Basis of many software vulnerabilities

```
char *buf = malloc(5);
for (int i=0; i<=5;i++)
{
        buf[i] = 'A' + i;
}
```

Pointer Checker does bounds checking!

Implementation Details

- Each pointer variable has bounds associated with it.
  - Bounds for p = malloc(size) are: p and p+size-1
  - Bounds for &v are: &v and &v+sizeof(v)-1
  - Bounds for &a[i] are: the bounds of &a. A pointer to an array element is allowed to traverse the entire array.
  - Bounds of &a.b are: &a+offset(b) and &a+offset(b) + sizeof(b)-1
- The pointer is allowed to go out of bounds, only memory accesses are checked to make sure the address is in bounds.
- Bounds for pointer variables in memory are kept in memory in a separate location mapped via the address of the pointer.
- Bounds for pointers in registers are in registers.

Enabling Pointer Checker

Getting started is easy…

Meet requirements:

Supported Languages	Supported Architecture	Supported Platforms	Supported Processor
C, C++	IA-32, Intel® 64	Linux*, Windows*	Intel® Pentium® 4 processor or later, or compatible processor

Compile and build your application with:

```
-check-pointers= [none | write | rw]       (Linux* OS)
/Qcheck-pointers:[none | write | rw]       (Windows* OS)
```

- Pointer Checker is off by default
- Checks all indirect accesses through pointers and accesses to arrays

One compiler switch enables Pointer Checker!

Enabling Pointer Checker [cont]

- Compile with Pointer Checker Enabling switch for “r” or“rw”.
- Execute; Check for out-of-bounds violations (OOB) and if OOB is true or false positive.

Sample Program

```
%cat main.c
1 #include<stdio.h>
2 #include<malloc.h>
3 #include "chkp.h"
4
5 int main () {
6 #ifdef REPORT
7
__chkp_report_control(__CHKP_REPORT_TRACE_LOG,
0);
8 #endif
9  char *buf = malloc(4);
10   int i;
11    for (i=0; i<=4; i++) {
12       printf(" %c",buf[i]);
13    }
14    for (i=0; i<=4; i++) {
15       buf[i] =  'A' + i;
16       printf(" %c",buf[i]);
17    }
18    printf ("\n");
19    return 0;
20 }
```

Pointer Checker Enabling – Check Bounds

- **Compile without Pointer Checker enabling switch:**

```
 % icc main.c -g;./a.out
 A B C D E
```

- **Compile with Pointer Checker enabling option:**

```
% icc main.c -DREPORT -check-pointers=write -
rdynamic -g;./a.out
CHKP: Bounds check error
   lb: 0xe67010
  ub: 0xe67013
  addr: 0xe67014
  end: 0xe67014
size: 1
```

Traceback is:

```
./a.out(main+0x21f) [0x402de7] in file main1.c line
15
 /lib64/libc.so.6(__libc_start_main+0xf4)
[0x35e8a1d994] in file unknown line 0
…


A B C D E
CHKP Total number of bounds violations: 1
%
```

Pointer Checker – *Detailed Control*

Model	Description	
Header File	Defines intrinsics and reporting functions (chkp.h)	
Compiler Options	-check-pointers (/Qcheck-pointers)	Enables pointer checker and adds associated libraries
	-check-pointers-dangling (/Q-check-pointers-dangling)	Enables checking for dangling pointer references
	-check-pointers-undimensioned (Qcheck-pointers-undimensioned)	Enables the checking of bounds for arrays without dimensions
Intrinsics	void * __chkp_lower_bound(void **)	Returns the lower bound associated with the pointer
	void * __chkp_upper_bound(void **)	Returns the upper bound associated with the pointer
	void * __chkp_kill_bounds(void *p)	Removes the bounds information to allow the pointer in the argument to access all memory.
	void * __chkp_make_bounds(void *p, size_t size)	Creates new bounds information within the allocated memory address for the pointer in the argument
Reporting API (Function/Enumeration)	void __chkp_report_control(__chkp_report_option_t option, __chkp_callback_t callback)	Determines how errors are reported
	__chkp_report_option_t {Enumerations: __CHKP_REPORT_LOG, __CHKP_REPORT_TRACEBACK, __CHKP_REPORT_CALLBACK, __CHKP_REPORT_BPT, __CHKP_REPORT_TERM}	Controls how out-of-bounds error are reported. Enumerations in header file
RTL Functions	Provides checking on C run-time library functions that manipulate memory through pointers	

Software & Services Group
Developer Products Division

Optimization Notice

Pointer Checker – *Samples for Control*

Check Arrays (with Bounds)/ Undimensioned Arrays

- **Check arrays without dimensions:**

```
-[no-]check-pointers-undimensioned
   (Linux* OS)
/Qcheck-pointers-undimensioned[-]
   (Windows* OS)
```

Check Runtime Library (RTL) functions / Intrinsics

```
Wrapper libraries:
     libchkpwrap.a>  [Linux*]
     libchkpwrap.lib [Windows*]

Intrinsics:
Write own wrappers for RTL functions
Work with Enabled and Non-Enabled code
Check and create correct bounds
```

Check for Dangling Pointers

- **Compiler uses a wrapper for free() and delete operator.**

<u>Option:</u>

```
-check-pointers-dangling=[none |
heap | stack | all] (Linux* OS)
/Qcheck-pointers-dangling:[none |
heap | stack | all] (Windows* OS)
```

Intrinsics - Example

```
extern void *wrap_malloc(size_t bytes) {
   // code
   ….
  (void *) p = my_realloc(p, old_size +
  100);
  p = __chkp_kill_bounds(p);
  p = (void*)__chkp_make_bounds(p,
  old_size + 100);
  return p;
}
```

Performance Overhead

- Runtime cost is relatively high, about 2x to 5x execution time effect.
- Code size increase from 20% to a very large increase (>100% plus), depending on the application.
- Pointer Checker is seen as a debug feature
- Deployed applications are expected to have Pointer Checker disabled
- Security benefits from catching vulnerabilities prior to product release is the trade-off.

Coding Security Support from Intel

Feature	Required Tools
Switches of Intel® Compiler to control diagnostics: Remarks, Warnings and Errors	Compiler
Code Coverage and Test Prioritization	Compiler
Static Verifier (Static Source Code Checking)	Compiler, Inspector XE – license of IPS-XE required
Switches of Intel® Compilers for run time checking (like variables not initialized, interface in FORTRAN, …)	Compiler
Intel® Inspector XE memory checking	Inspector XE
Intel® Inspector XE thread checking	Inspector XE
Pointer Checking	Compiler, license of IPS-XE required
Data race detection of Intel® Debugger and Intel-extended GDB	Compiler, Debugger

Is Pointer Checking Redundant then ?

1. Investment and intrusiveness is different for all test methods
   - Inspector XE memory checking can extend run time by a factor far above 50 !
2. Pointer Checking can discover issues not found by other methods

```
void s(char *buf)
for (int i=0; i<=5;i++)
{
   buf[i] = 'A' + i;
}

int main()
{
  char buf[5];
  char *p=buf;
  s(p);
}
```

- Static Verifier will detect this only in case both modules are compiled in one step
- Inspector XE memory checking will miss the fault since memory is not from heap

Intel® Fortran Composer XE

Coarray Fortran (CAF)

- Simple extension to Fortran to make Fortran into a robust and efficient parallel programming language
- Single-Process-Multiple-Data programming model (SPMD).
  - Single program is replicated a fixed number of times
  - Each program instance has it's own set of data objects – called an "IMAGE"
  - Each image executes asynchronously and normal Fortran rules apply
  - Extensions to normal Fortran array syntax to allow images to reference data in other image(s)
- Part of the Fortran 2008 standard

Compilation

- `ifort –coarray          !Linux*`
- `ifort /Qcoarray         !Windows*`

  along with other options. Enables compiling for CAF. By default, executable will use as many cores (real and hyperthreaded) as are available.

  ```
  ifort –coarray -coarray-num-procs=x
  ifort /Qcoarray /Qcoarray-num-procs=x
  ```

  along with other options. Sets number of images to “x”. This option will soon change to –coarray-num-images=x, in our next update hopefully. See the RELEASE NOTES on each beta update this summer for the latest breaking news on option and env name changes.
- No Visual Studio property settings with initial beta products. Manually add to VS “Command Line” “Additional Options”

Running (linux)

- Simple hello world:

```
program hello_image
    write(*,*) "Hello from image ", this_image(), &
       "out of ", num_images()," total images"
end program hello_image
```

```
ifort –coarray –o hello_image hello_image.f90
./hello_image
 Hello from image           1 out of          4  total images
 Hello from image           4 out of          4  total images
 Hello from image           2 out of          4  total images
 Hello from image           3 out of          4  total images
```

Controlling the Number of Images, Command Line

- CAF is a SPMD model. "Images" similar to MPI "Processes"
- Environment variable can set number of images
- Environment variable overrides –coarray-num-procs compiler option

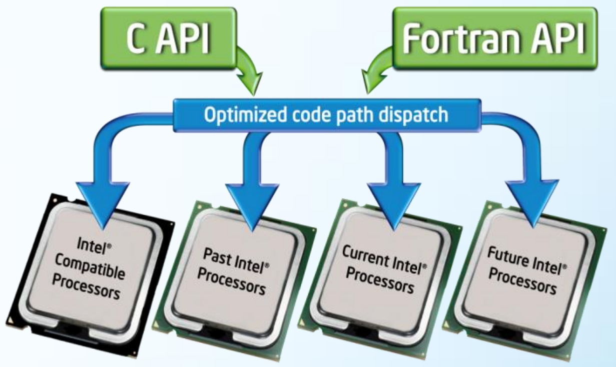
```
Linux host>  export FOR_IMAGES_NUM=2
./hello_image
Window host> set FOR_IMAGES_NUM=2
hello_image.exe
 Hello from image 1 out of 2  total images
 Hello from image 2 out of 2  total images
```

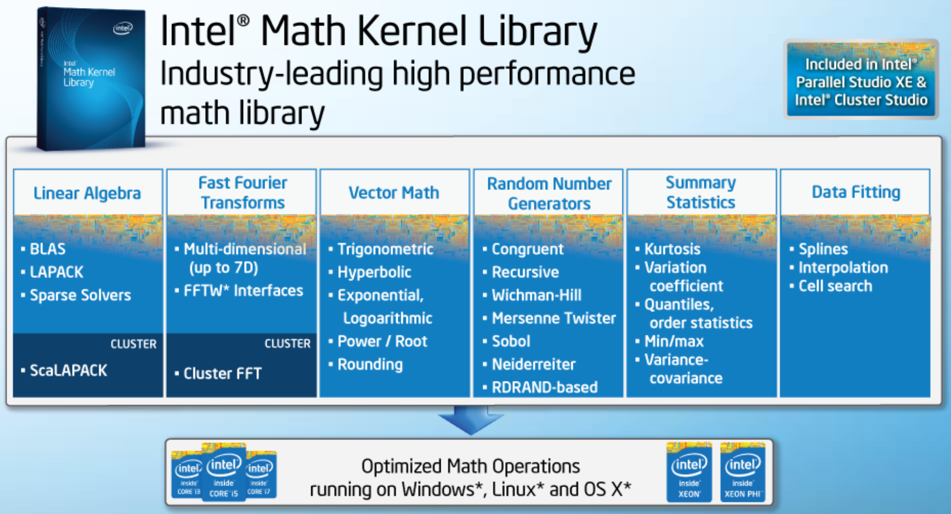
Further Reading

- Coarrays in the next Fortran Standard
  - ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1824.pdf
- The New Features of Fortran 2008
  - ftp://ftp.nag.co.uk/sc22wg5/N1801-N1850/N1828.pdf
- Fortran 2008 Standard (current draft)
  - http://j3-fortran.org/doc/standing/links/007.pdf

Intel® Math Kernel Library
Flagship math processing library

- Multi-core ready with excellent scaling
- Highly optimized, extensively threaded math routines for science, engineering and financial applications for maximum performance
- Automatic runtime processor detection ensures great performance on whatever processor your application is running on.
- Support for C and Fortran
- Optimizations for latest Intel processors including 4th-gen Core processors


C API
Fortran API
Optimized code path dispatch
Intel® Compatible Processors
Past Intel® Processors
Current Intel® Processors
Future Intel® Processors


Intel Math Kernel Library
Included in Intel® Parallel Studio XE & Intel® Cluster Studio

Intel® Math Kernel Library
Industry-leading high performance math library

Linear Algebra	Fast Fourier Transforms	Vector Math	Random Number Generators	Summary Statistics	Data Fitting
• BLAS • LAPACK • Sparse Solvers	• Multi-dimensional (up to 7D) • FFTW* Interfaces	• Trigonometric • Hyperbolic • Exponential, Logoarithmic • Power / Root • Rounding	• Congruent • Recursive • Wichman-Hill • Mersenne Twister • Sobol • Neiderreiter • RDRAND-based	• Kurtosis • Variation coefficient • Quantiles, order statistics • Min/max • Variance-covariance	• Splines • Interpolation • Cell search
CLUSTER • ScaLAPACK	CLUSTER • Cluster FFT				

Optimized Math Operations running on Windows*, Linux* and OS X*

Optimization Notice

Application Areas which could use MKL

- **Energy** - Reservoir simulation, Seismic, Electromagnetics, etc.
- **Finance** - Options pricing, Mortgage pricing, financial portfolio management etc.
- **Manufacturing** **-** CAD, FEA etc.
- **Applied mathematics**
  - Linear programming, Quadratic programming, Boundary value problems, Nonlinear parameter estimation, Homotopy calculations, Curve and surface fitting, Numerical integration, Fixed-point methods, Partial and ordinary differential equations, Statistics, Optimal control and system theory
- **Physics** & **Computer science**
  - Spectroscopy, Fluid dynamics, Optics, Geophysics, seismology, and hydrology, Electromagnetism, Neural network training, Computer vision, Motion estimation and robotics
- **Chemistry**
  - Physical chemistry, Chemical engineering, Study of transition states, Chemical kinetics, Molecular modeling, Crystallography, Mass transfer, Speciation
- **Engineering**
  - Structural engineering, Transportation analysis, Energy distribution networks, Radar applications, Modeling and mechanical design, Circuit design
- **Biology and medicine**
  - Magnetic resonance applications, Rheology, Pharmacokinetics, Computer-aided diagnostics, Optical tomography
- **Economics and sociology**
  - Random utility models, Game theory and international negotiations, Financial portfolio management

Intel® MKL Contents

- **BLAS**
  - Basic vector-vector/matrix-vector/matrix-matrix computation routines.
- **Sparse BLAS**
  - BLAS for sparse vectors/matrices
- **LAPACK (Linear algebra package)**
  - Solvers and eigensolvers. Many hundreds of routines total!
  - C interface to LAPACK
- **ScaLAPACK**
  - Computational, driver and auxiliary routines for distributed-memory architectures
- **DFTs (General FFTs)**
  - Mixed radix, multi-dimensional transforms
- **Cluster DFT**
  - For Distributed Memory systems
- **Sparse Solvers (PARDISO, DSS and ISS)**
  - Direct and Iterative sparse solvers for symmetric, structurally symmetric or non-symmetric, positive definite, indefinite or Hermitian sparse linear system of equations
  - Out-Of-Core (OOC) version for huge problem sizes

Intel® MKL Contents

- **VML (Vector Math Library)**
  - Set of vectorized transcendental functions, most of libm functions, but faster
- **VSL (Vector Statistical Library)**
  - Set of vectorized random number generators
  - SSL (Summary Statistical Library) : Computationally intensive core/building blocks for statistical analysis
- **PDEs (Partial Differential Equations)**
  - Trigonometric transform and Poisson solvers.
- **Optimization Solvers**
  - Solvers for nonlinear least square problems with/without boundary condition
- **Support Functions**

References

- Intel® MKL product Information
  - http://software.intel.com/en-us/intel-mkl/
- User Discussion Forum
  - http://software.intel.com/en-us/forums/intel-math-kernel-library/
- What are the new software tools?
  - http://whatif.intel.com

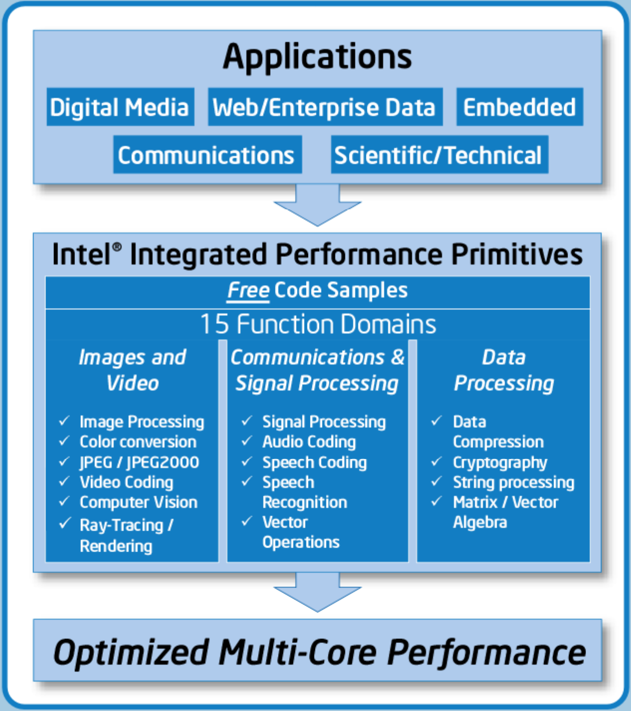
Intel® Integrated Performance Primitives (IPP)

Multicore Power for Multimedia and Data Processing

- Features
  - Rapid Application Development
  - Cross-platform Compatibility & Code Re-Use
  - Highly optimized functions from 15 Domains
    - Images and Video
    - Communications and Signal Processing
    - Data Processing
  - Performance optimizations for latest 4th generation Intel Core processors and Atom processors

"I recently compared Intel IPP to the open source library zlib. I observed a ***considerable performance improvement*** (at least 10-15%) using IPP compared to zlib on the latest Intel processors. I've concluded that we should migrate our open source part of the project to utilize IPP."

Vadim Kavaleroy
Investment Analyst
Susquehanna International Group



Optimization Notice

Intel® Integrated Performance Primitives

Applications
Digital Media | Web/Enterprise Data | Embedded Communications | Scientific/Technical

High level APIs and Codecs Interfaces and Code Samples

Intel® Integrated Performance Primitives 16 Function Domains

Multimedia

- Image Processing
- Color Conversion
- JPEG/JPEG2000
- Video Coding
- Computer Vision
- Realistic Rendering

Signal Processing

- Signal Processing
- Audio Coding
- Speech Coding
- Speech Recognition
- Vector Operations

Data Processing

- Data Compression
- Data Integrity
- Cryptography
- String Processing
- Matrix Operations

Cross-platform C/C++ API for Code Re-use

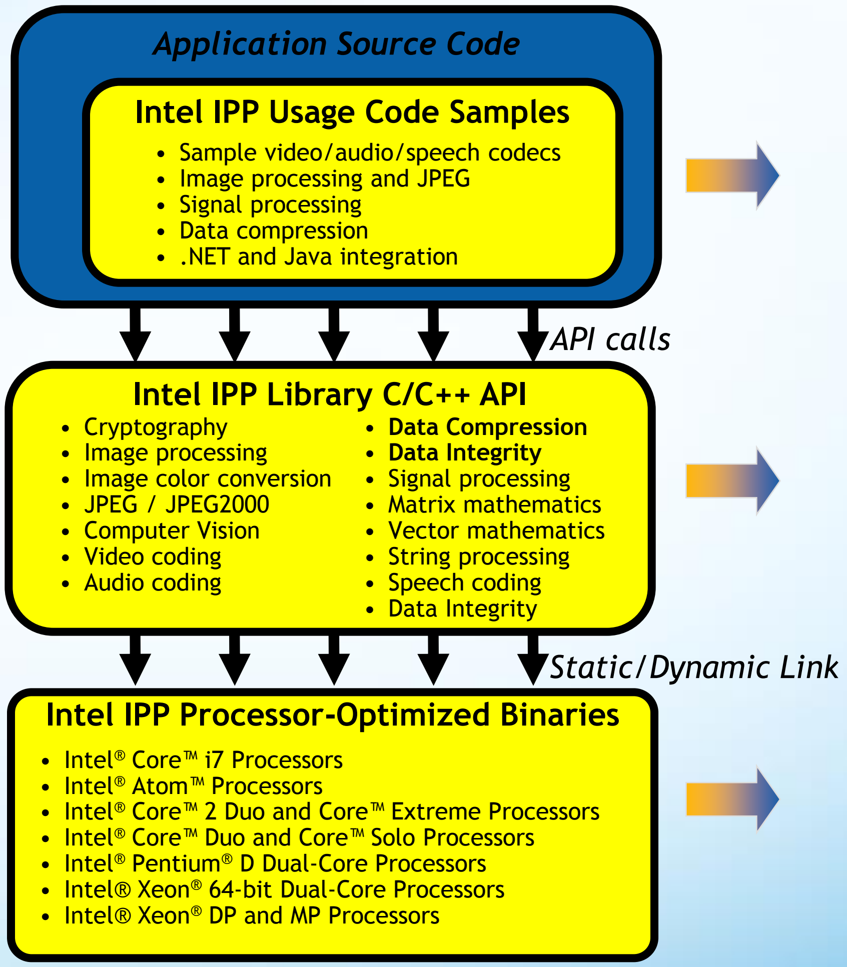
Optimized 32-bit and 64-bit Multicore Performance

Intel® Integrated Performance Primitives

Free Code Samples

Cross-platform API

Processor-Optimized Implementation



Application Source Code

Intel IPP Usage Code Samples

- Sample video/audio/speech codecs
- Image processing and JPEG
- Signal processing
- Data compression
- .NET and Java integration

API calls

Intel IPP Library C/C++ API

- Cryptography
- Image processing
- Image color conversion
- JPEG / JPEG2000
- Computer Vision
- Video coding
- Audio coding
- **Data Compression**
- **Data Integrity**
- Signal processing
- Matrix mathematics
- Vector mathematics
- String processing
- Speech coding
- Data Integrity

Static/Dynamic Link

Intel IPP Processor-Optimized Binaries

- Intel® Core™ i7 Processors
- Intel® Atom™ Processors
- Intel® Core™ 2 Duo and Core™ Extreme Processors
- Intel® Core™ Duo and Core™ Solo Processors
- Intel® Pentium® D Dual-Core Processors
- Intel® Xeon® 64-bit Dual-Core Processors
- Intel® Xeon® DP and MP Processors

Rapid Application Development

Compatibility and Code Re-Use

Outstanding Performance

Optimization Notice

Intel® Integrated Performance Primitives
Functions and Samples

Domains	Functions	Samples
1. Image Processing	* Geometry transformations, such as resize/rotate * Linear and non-linear filtering operation on an image for edge detection, blurring, noise removal and etc for filter effect. * Linear transforms for 2D FFTs, DFTs, DCT. * image statistics and analysis	* Tiled Image Processing / 2D Wavelet Transform /C++ Image Processing Classes/Image Processing functions Demo
2. Computer Vision	* Background differencing, Feature Detection (Corner Detection, Canny Edge detection), Distance Transforms, Image Gradients, Flood fill, Motion analysis and Object Tracking, Pyramids, Pattern recognition, Camera Calibration	* Face Detection
3. Color conversion	* Converting image/video color space formats: RGB, HSV, YUV, YCbCr * Up/Down sampling * Brightness and contrast adjustments	
4. JPEG Coding	* High-level JPEG and JPEG2000 compression and decompression functions * JPEG/JPEG2000 support functions: DCT, Wavelet transforms, color conversion, downsampling	• UIC-Unified Image Codec/ Integration with the Independent JPEG Group (IJG) library
5. Video Coding	* VC-1, H.264, MPEG-2, MPEG-4, H.261, H.263 and DV codec support functions	* Simple Media Player/ Video Encoder / h.264 decoding
6. Audio Coding	* Echo cancellation and audio transcoding, BlockFiltering, Spectral Data prequantization.	* Audio Codec Console application
7. Realistic Rendering	* Acceleration Structures, Ray-Scene Intersection and Ray Tracing * Surface properties, shader support, tone mapping	* Ray Tracing

Intel® Integrated Performance Primitives
Functions and Samples

Domains	Functions	Samples
8. Speech Coding	* Adaptive/Fixed Codebook functions, Autocorrelation, Convolution, Levinson-Durbin recursion, Linear Prediction Analysis & Quantization, Echo Cancellation, Companding	* G.168, G.167, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.723.1, G.726, G.728, G.729, RT-Audio, GSM AMR, GSM FR
9. Data Integrity	▪ Error-Correcting Codes ▪ Reed-Solomon	
10. Data Compression	* Entropy-coding compression: Huffman, VLC * Dictionary-based compression: LZSS, LZ77 * Burrows-Wheeler Transform, MoveToFront, RLE, Generalized Interval Transformation * Compatible feature support for zlib and bzip2	* zlib, bzip2, gzip-compatible /General data compression examples
11. Cryptography	* Big-Number Arithmetic / Rijndael, DES, TDES, SHA1, MD5, RSA, DSA, Montgomery, prime number generation and pseudo-random number generation (PRNG) functions	* Intel IPP crypto usage in Open SSL*
12. String Processing	* Compare, Insert, change case, Trim, Find, Regexp, Hash	* “ippgrep” – regular expression matching

Intel® Integrated Performance Primitives
Functions and Samples

Domains	Functions	Samples
13. Signal Processing	* Transforms: DCT, DFT, MDCT, Wavelet (both Haar and user-defined filter banks), Hilbert * Convolution, Cross-Correlation, Auto-Correlation, Conjugate * Filtering: IIR/FIR/Median filtering, Single/Multi-Rate FIR LMS filters * Other: Windowing, Jaehne/Tone/Traingle signal generation, Thresholding	* Signal Processing Function Demo
14. Vector Math	* Logical, Shift, Conversion, Power, Root, Exponential, Logarithmic, Trigonometric, Hyperbolic, Erf, Erfc	
15. Matrix Math	* Addition, Multiplication, Decomposition, Eigenvalues, Cross-product, transposition	
Other Common Functions	* CPUTypes, Thread Number Control, Memory Allocation	* Linkages/Different language support

Intel IPP is suitable for a very wide range of applications

Video broadcasting, Video/Voice Conferencing

Consumer Multimedia

Medical Imaging, Document Imaging

Computer Vision /Object Tracking / Machine Learning

Databases and Enterprise Data Management

Information Security

Embedded Applications

Mathematical and Scientific

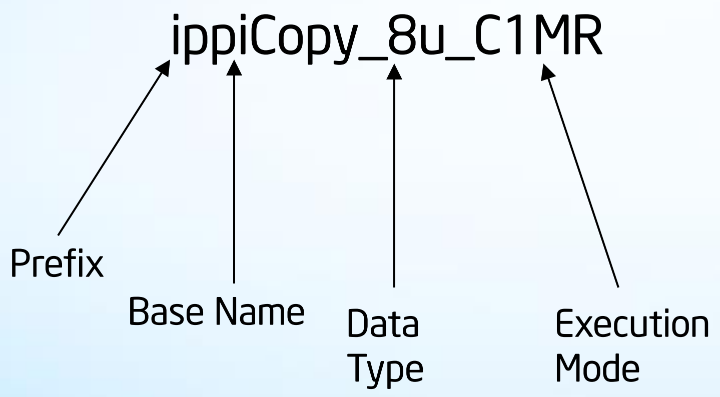
Optimization Notice

Intel® Integrated Performance Primitives

Function naming convention and usage

Function names

- ✓ are easy to understand
- ✓ directly indicate the purpose of the function via distinct elements
- ✓ each element has a fixed number of pre-defined values



Name Elements	Description	Examples
Prefix	Indicates the functional data type in 1D , 2D and Matrix	ipps, ippi, ippm
Base Name	Abbreviation for the core operation	Add, FFTFwd, LuDecomp
Data Type	Describes bit depth and sign	8u, 32f, 64f
Execution mode	Indicates data layout and scaling	ISfs, C1R, P

Each function performs a particular operation on a known type of data in a specific mode

References

- Intel® IPP product Information
  - http://software.intel.com/en-us/intel-ipp/
- User Discussion Forum
  - http://software.intel.com/en-us/forums/intel-integrated-performance-primitives
- What are the new software tools?
  - http://whatif.intel.com

Intel® Composer XE 3
Debugging Tools Overview

- **For Linux*/Mac OS*:**
  Intel® Debugger 13.0 (IDB)
  - deprecated, but still required for full Fortran debugging

  Enhanced GDB* Debugger
  - GNU gdb 7.5 + branch trace and data race detection

Intel® Composer XE 2011

Debugging Tools Overview (cont'd)

Intel® Debugger (IDB)

Linux*

- Intel® Parallel Studio XE
- Intel® C++ Studio XE
- Intel® Composer XE
- Intel® C++ Composer XE
- Intel® Fortran Composer XE

- Intel® Cluster Toolkit Compiler Edition

Mac OS*

- Intel® Composer XE
- Intel® C++ Composer XE
- Intel® Fortran Composer XE

Intel® Parallel Debugger Extension

Windows*

- Intel® Parallel Studio XE
- Intel® C++ Studio XE
- Intel® Composer XE
- Intel® C++ Composer XE
- Intel® Visual Fortran Composer XE

Intel® Debugger (IDB)
Overview

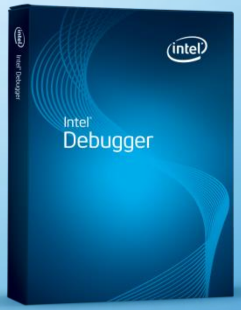
Linux*

- GUI + Command Line

Mac OS*

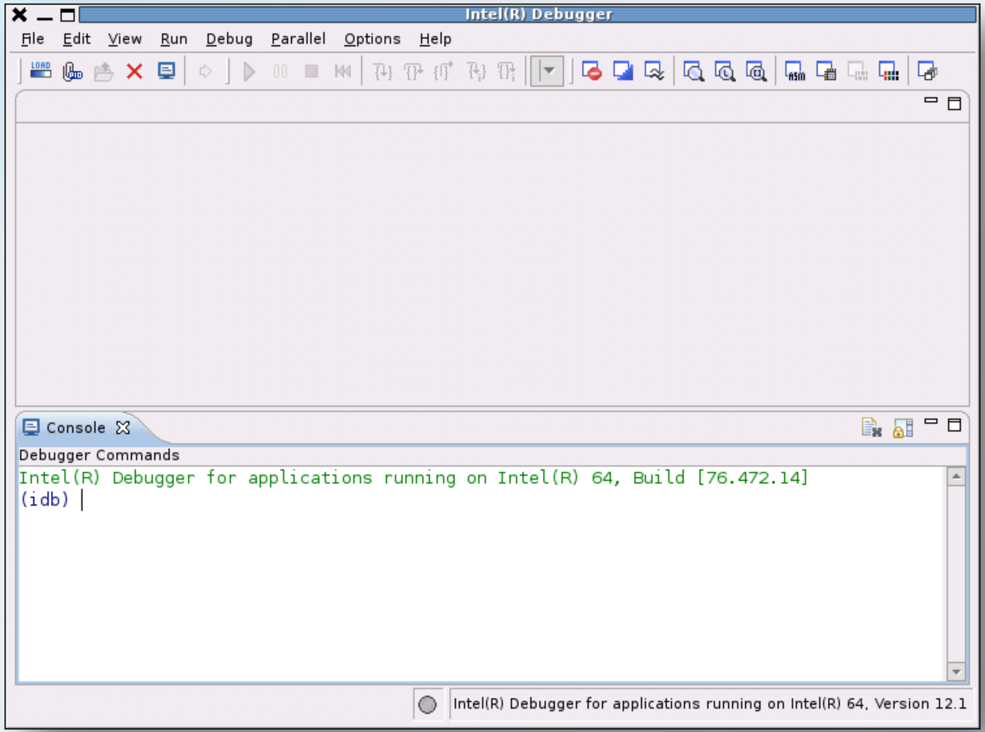
- Command Line only

- Support for TBB, Cilk Plus, OpenMP & Native Threads
- GDB alike Command Line Syntax (also in GUI if available)
- Languages: C/C++ & FORTRAN
- Start GUI via `idb`, Command Line via `idbc`



Intel® Debugger (IDB)

GUI



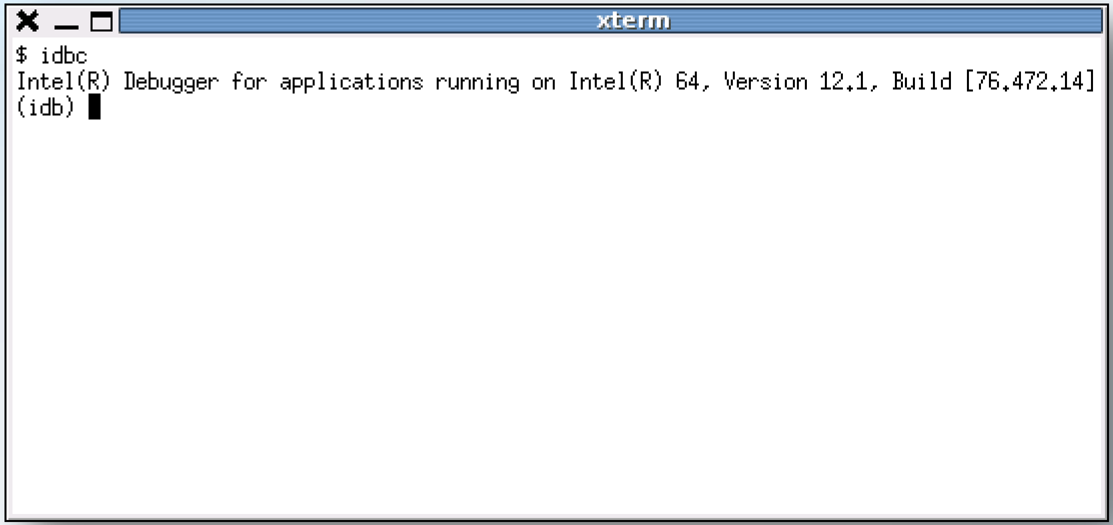
```
Intel(R) Debugger for applications running on Intel(R) 64, Build [76.472.14]
(idb)
```

Intel(R) Debugger for applications running on Intel(R) 64, Version 12.1

- Eclipse based GUI with similar Look&Feel
- No project/solution to set up – use debugger on-the-fly
- Remembers windows and positions across debugging sessions

→ Intuitive and always-ready to debug

Intel® Debugger (IDB)
Command Line

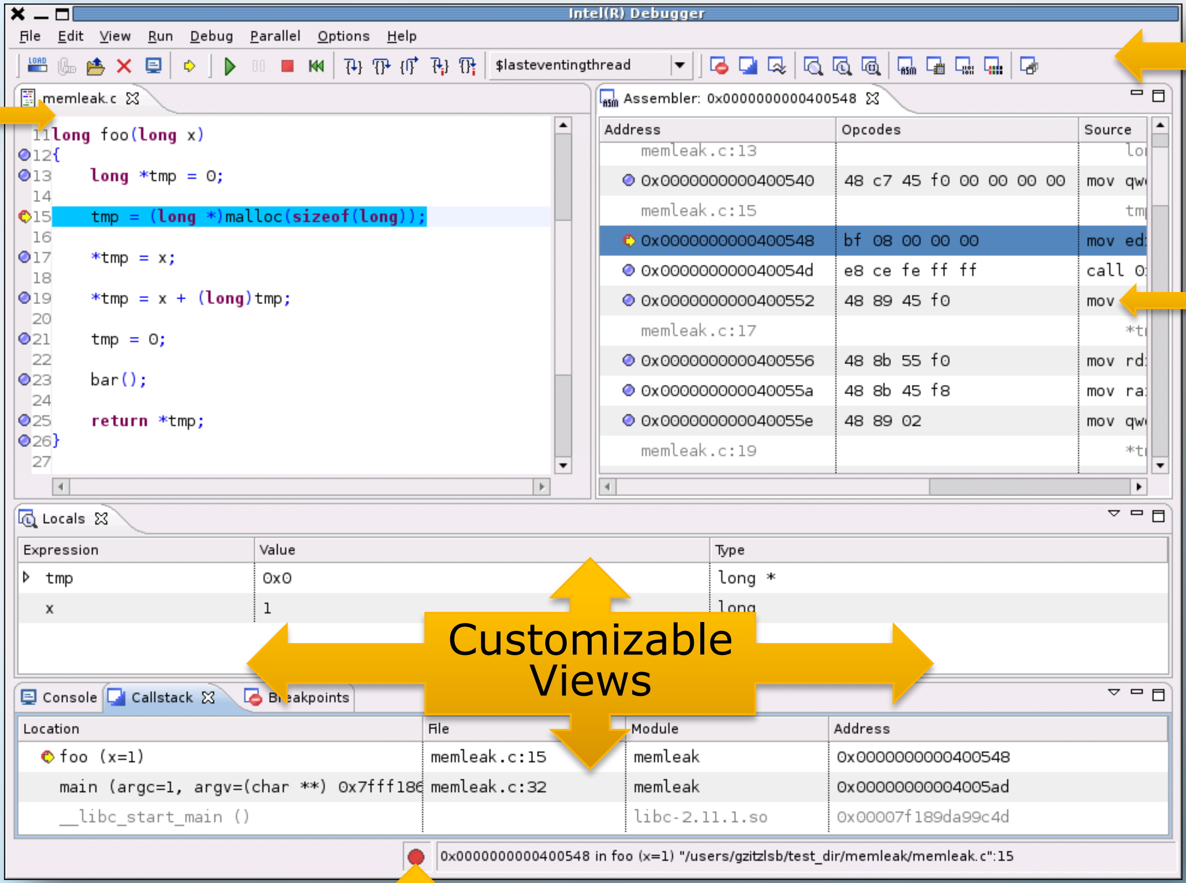


```
$ idbc
Intel(R) Debugger for applications running on Intel(R) 64, Version 12.1, Build [76.472.14]
(idb) 
```

- GDB alike syntax with similar Look&Feel
- Useful for debugging where no GUI is available
- Allows execution of scripts (e.g. regression testing)

Low system requirements & familiar syntax

Intel® Debugger (IDB)
Information at a Glance



```
long foo(long x)
{
    long *tmp = 0;

    tmp = (long *)malloc(sizeof(long));

    *tmp = x;

    *tmp = x + (long)tmp;

    tmp = 0;

    bar();

    return *tmp;
}
```

Address	Opcodes	Source
memleak.c:13		lo
0x0000000000400540	48 c7 45 f0 00 00 00 00	mov qw
memleak.c:15		tm
0x0000000000400548	bf 08 00 00 00	mov ed
0x000000000040054d	e8 ce fe ff ff	call 0
0x0000000000400552	48 89 45 f0	mov
memleak.c:17		*t
0x0000000000400556	48 8b 55 f0	mov rd
0x000000000040055a	48 8b 45 f8	mov ra
0x000000000040055e	48 89 02	mov qw
memleak.c:19		*t

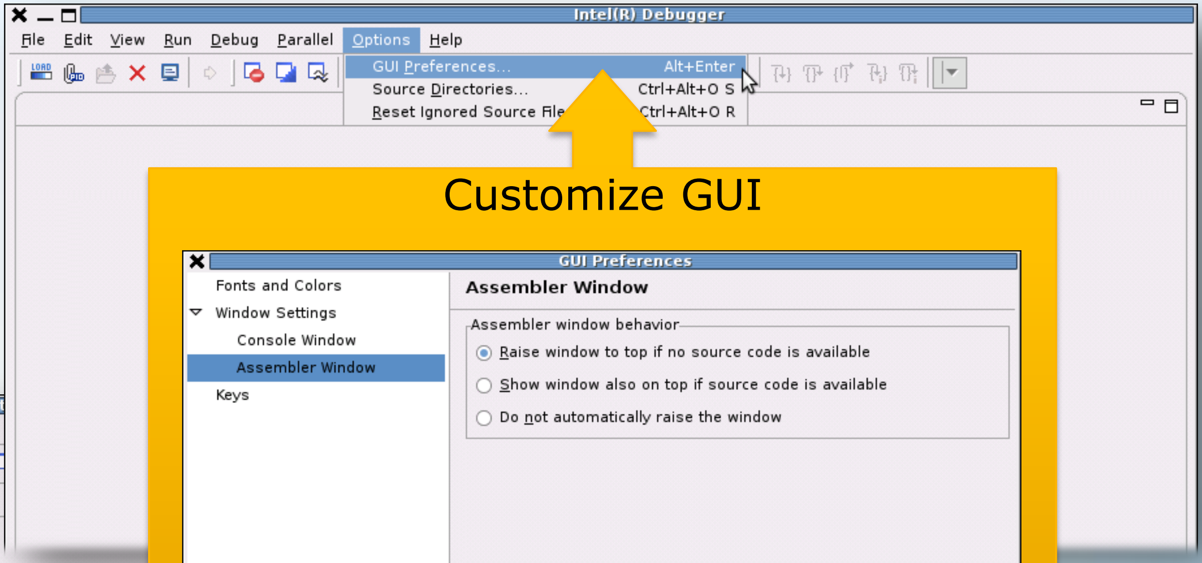
Expression	Value	Type
tmp	0x0	long *
x	1	long

Location	File	Module	Address
foo (x=1)	memleak.c:15	memleak	0x0000000000400548
main (argc=1, argv=(char **) 0x7fff186	memleak.c:32	memleak	0x00000000004005ad
__libc_start_main ()		libc-2.11.1.so	0x00007f189da99c4d

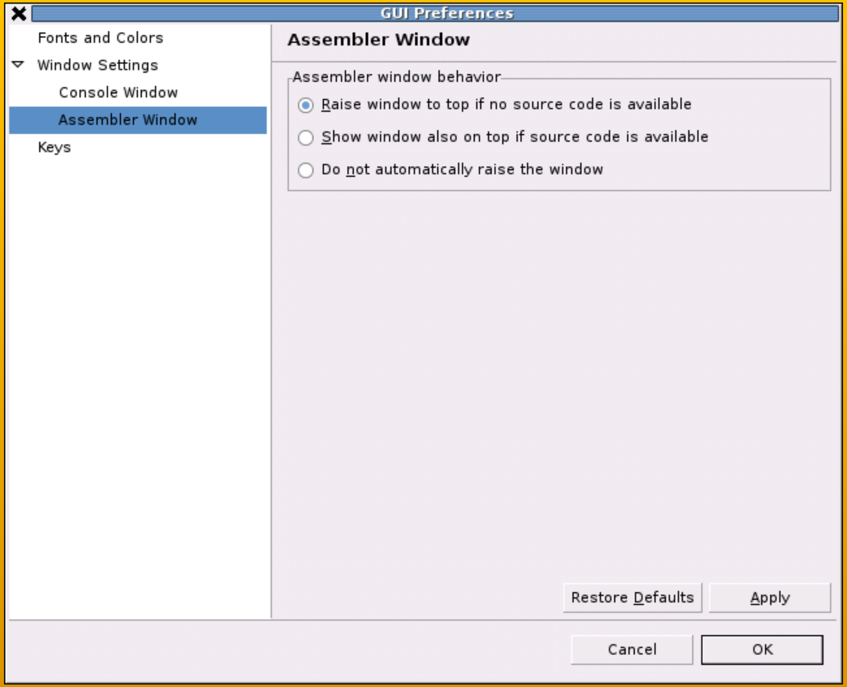
0x0000000000400548 in foo (x=1) "/users/gzitzlsb/test_dir/memleak/memleak.c":15

Customizable Hotkeys

Intel® Debugger (IDB)
Customize I


Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
GUI Preferences... Alt+Enter
Source Directories... Ctrl+Alt+O S
Reset Ignored Source File Ctrl+Alt+O R

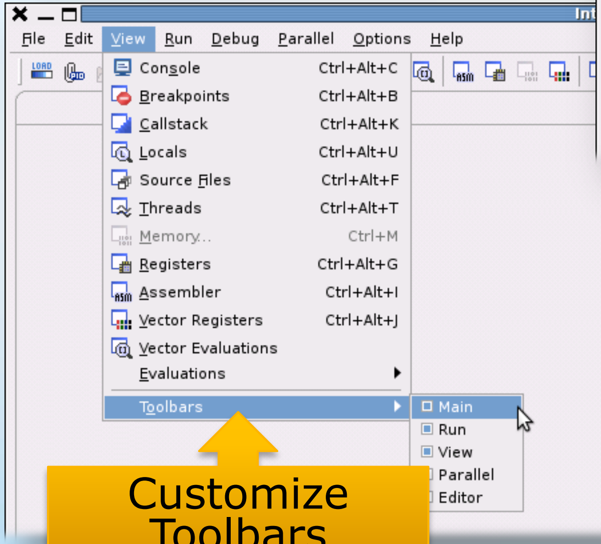
Customize GUI



Assembler Window

Assembler window behavior

- [✓ Raise window to top if no source code is available]
- Show window also on top if source code is available
- Do not automatically raise the window

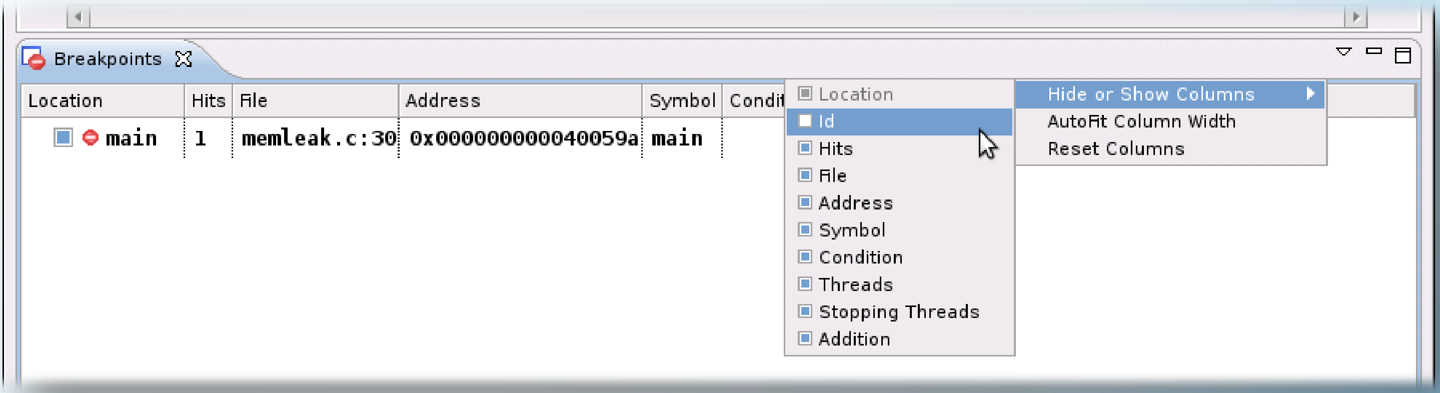


Customize Toolbars

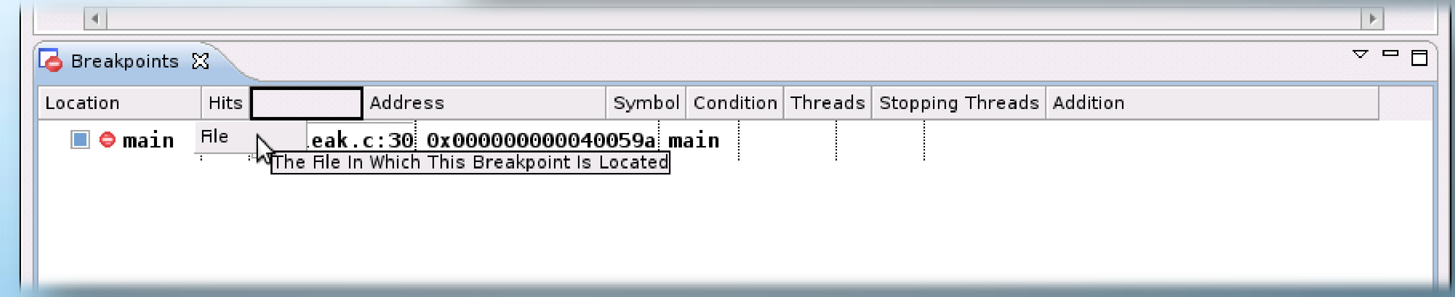
Optimization Notice

Intel® Debugger (IDB)
Customize II

Windows showing Tables can be customized:
•Hide or Show Column Entries
•Change Column Width
•Rearrange Entries



Location	Hits	File	Address	Symbol
main	1	memleak.c:30	0x000000000040059a	main

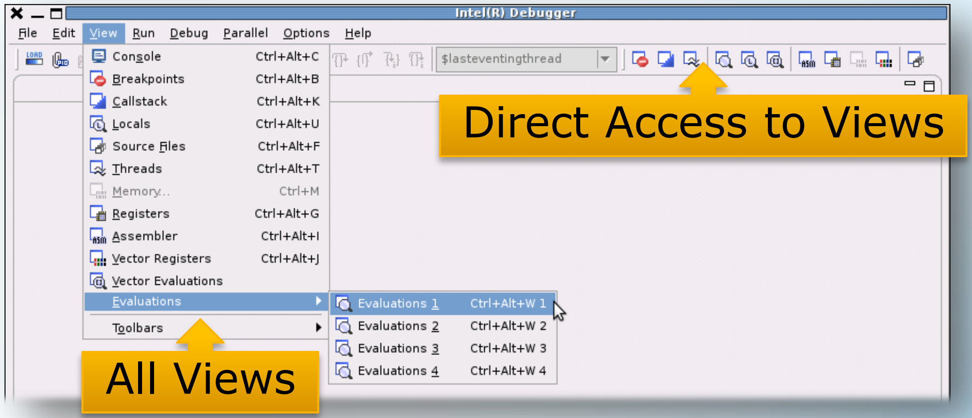


Location	Hits		Address	Symbol	Condition	Threads	Stopping Threads	Addition
main		.eak.c:30	0x000000000040059a	main				

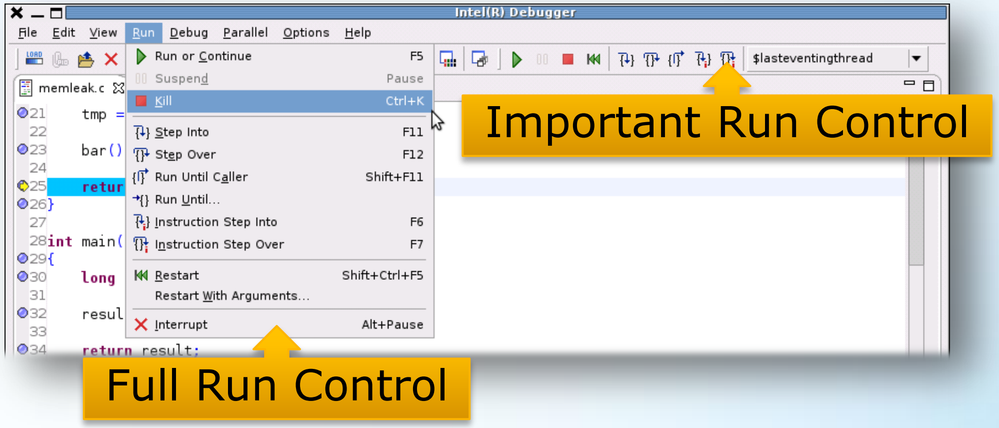
Optimization Notice

Intel® Debugger (IDB)
Views

•**Console**:
Like Command Line; GDB Syntax
•**Threads**:
List of all Threads
•**Callstack**:
Show Backtrace
•**(Vector) Registers**:
CPU Registers
•**Memory**:
Manually inspect Memory
•**Assembler**:
Instruction level debugging
•**Breakpoints**:
Events to Stop (Code, Data, Synchronization)
•**Locals, (Vector) Evaluations**:
Show Variables; automatic or selected
•**Source Files**:
List of all Sources with Debug Information

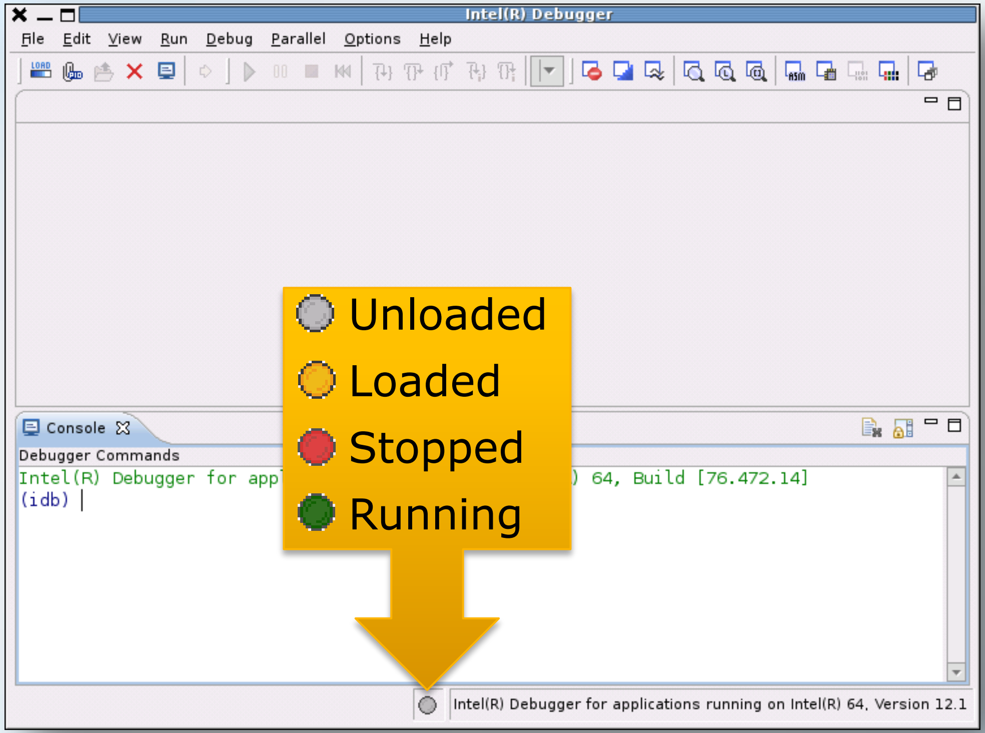


Intel® Debugger (IDB)
Run Control



- **Run or Continue**:
  Run or Continue Execution
- **Suspend**:
  Pause Execution
- **Kill**:
  Terminate Application
- **Interrupt**:
  Abort Requests to IDB (e.g. complex evaluations)
- **(Instruction) Step Into/Over**:
  Next Statement/Instruction following or not following calls/jumps
- **Run Until (Caller)**:
  Continue until Caller or until Location is reached
- **Restart (with Arguments)**:
  Terminate application & run again (providing different arguments)
- **Current Thread Set** (default: $lasteventingthread):
  Set of Threads that are affected by Run Control

Intel® Debugger (IDB)
Run Control Indicator



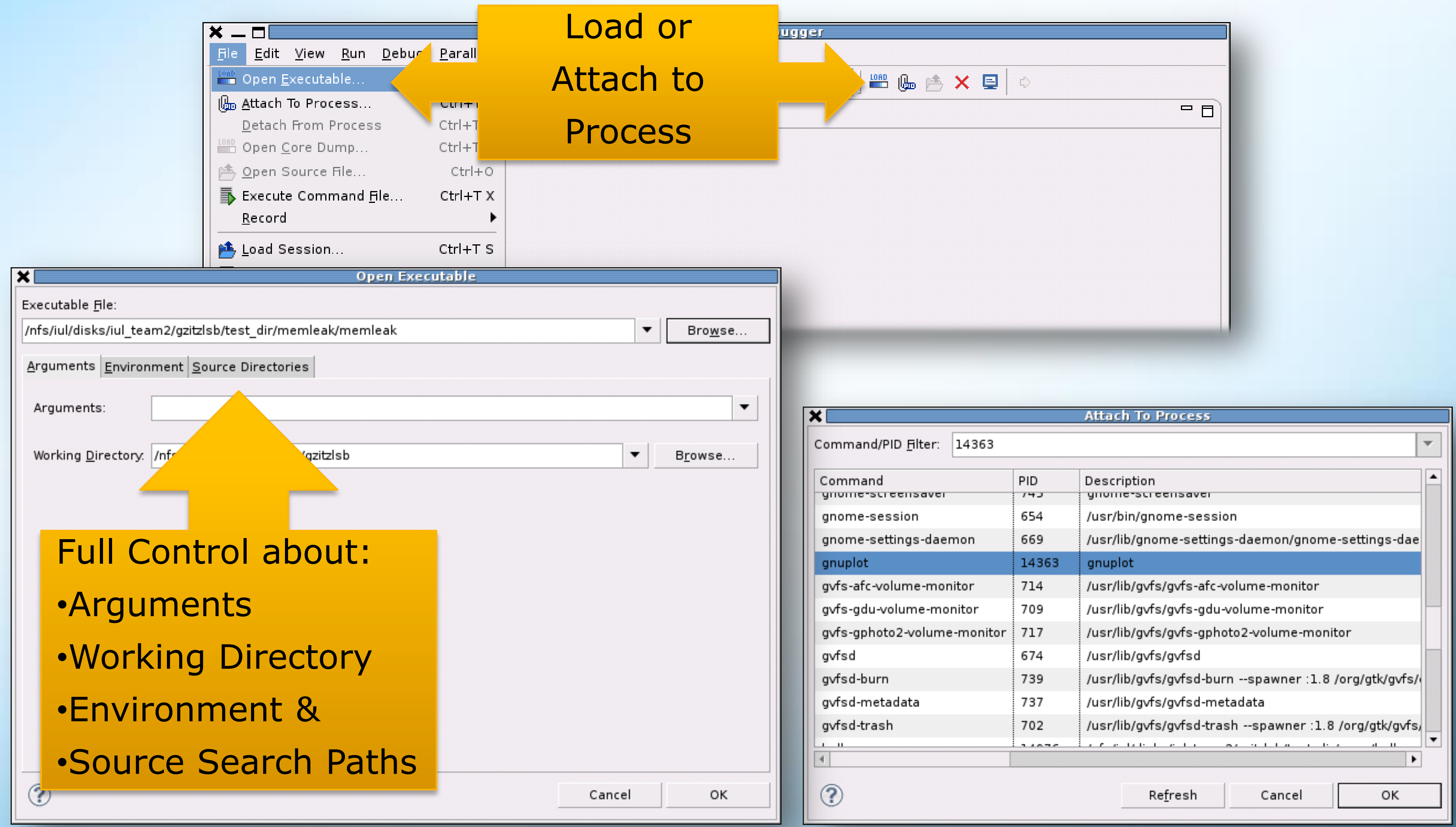
```
Intel(R) Debugger for app...) 64, Build [76.472.14]
(idb) |
```

Intel(R) Debugger for applications running on Intel(R) 64, Version 12.1

- The Run Control Indicator shows the current Application State
- This Indicator is live Information from the Debugger

➡ In some cases (e.g. Inferior Calls) the Debugger might execute Code from the Application even if stopped!

Intel® Debugger (IDB)
Start a Debugging Session Easily



Open Executable

Executable File: /nfs/iul/disks/iul_team2/gzitzlsb/test_dir/memleak/memleak

Arguments | Environment | Source Directories

Arguments:

Working Directory: /nf… /gzitzlsb

Browse... Cancel OK

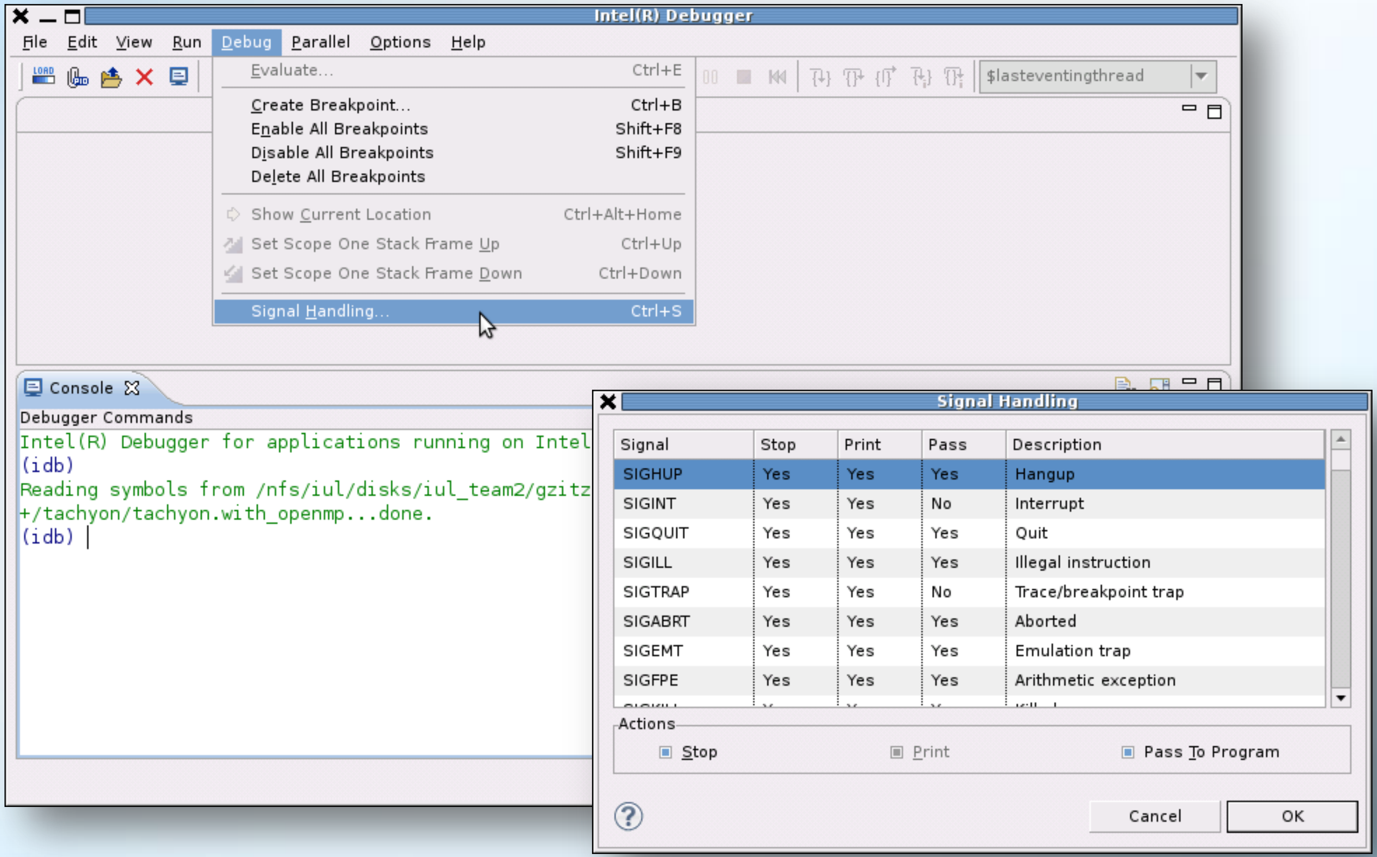
Attach To Process

Command/PID Filter: 14363

Command	PID	Description
gnome-screensaver	743	gnome-screensaver
gnome-session	654	/usr/bin/gnome-session
gnome-settings-daemon	669	/usr/lib/gnome-settings-daemon/gnome-settings-dae
gnuplot	14363	gnuplot
gvfs-afc-volume-monitor	714	/usr/lib/gvfs/gvfs-afc-volume-monitor
gvfs-gdu-volume-monitor	709	/usr/lib/gvfs/gvfs-gdu-volume-monitor
gvfs-gphoto2-volume-monitor	717	/usr/lib/gvfs/gvfs-gphoto2-volume-monitor
gvfsd	674	/usr/lib/gvfs/gvfsd
gvfsd-burn	739	/usr/lib/gvfs/gvfsd-burn --spawner :1.8 /org/gtk/gvfs/
gvfsd-metadata	737	/usr/lib/gvfs/gvfsd-metadata
gvfsd-trash	702	/usr/lib/gvfs/gvfsd-trash --spawner :1.8 /org/gtk/gvfs/

Refresh Cancel OK

Intel® Debugger (IDB)
Signals


Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
Evaluate... Ctrl+E
Create Breakpoint... Ctrl+B
Enable All Breakpoints Shift+F8
Disable All Breakpoints Shift+F9
Delete All Breakpoints
Show Current Location Ctrl+Alt+Home
Set Scope One Stack Frame Up Ctrl+Up
Set Scope One Stack Frame Down Ctrl+Down
Signal Handling... Ctrl+S
$lasteventingthread
Console
Debugger Commands
Signal Handling
Actions
Stop
Print
Pass To Program
Cancel
OK

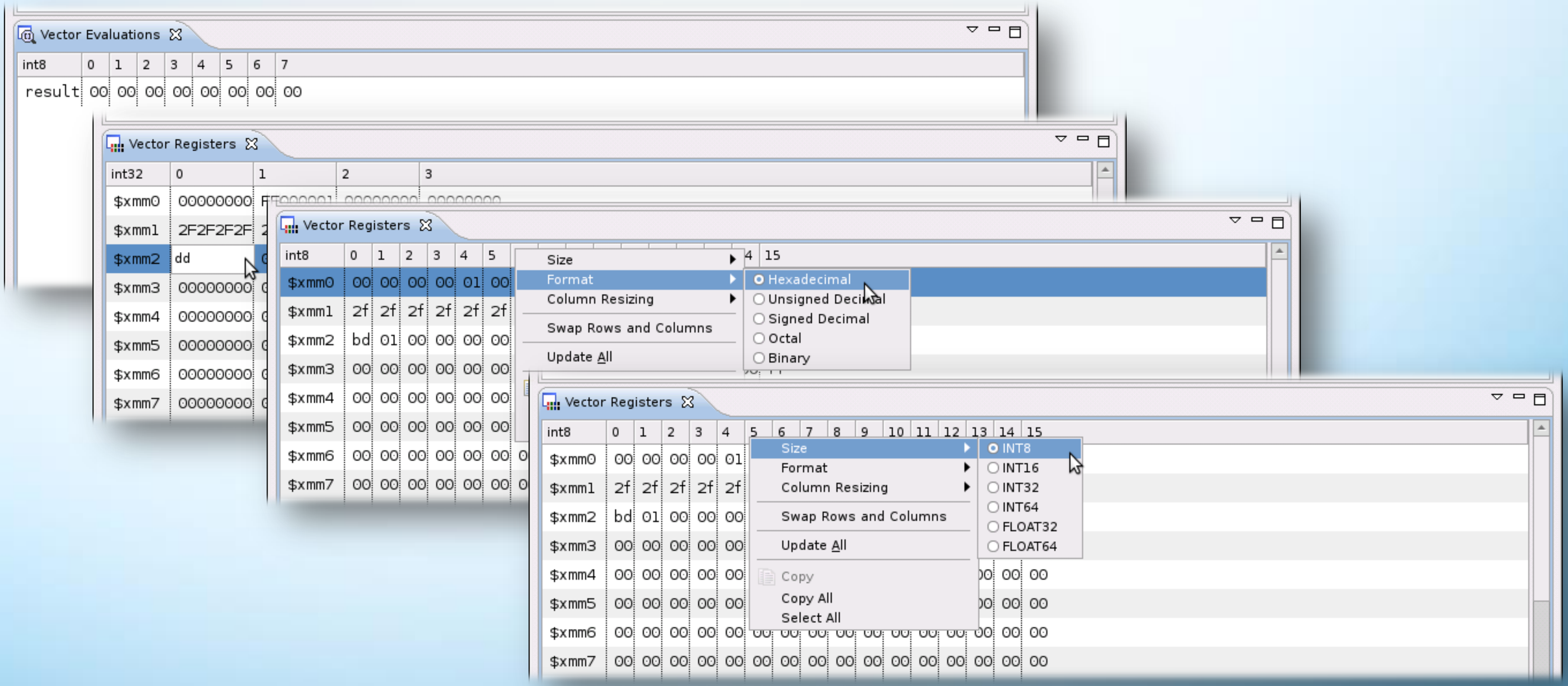
```
Intel(R) Debugger for applications running on Intel
(idb)
Reading symbols from /nfs/iul/disks/iul_team2/gzitz
+/tachyon/tachyon.with_openmp...done.
(idb) |
```

Signal	Stop	Print	Pass	Description
SIGHUP	Yes	Yes	Yes	Hangup
SIGINT	Yes	Yes	No	Interrupt
SIGQUIT	Yes	Yes	Yes	Quit
SIGILL	Yes	Yes	Yes	Illegal instruction
SIGTRAP	Yes	Yes	No	Trace/breakpoint trap
SIGABRT	Yes	Yes	Yes	Aborted
SIGEMT	Yes	Yes	Yes	Emulation trap
SIGFPE	Yes	Yes	Yes	Arithmetic exception

- Flexible and Easy Control of System Signals

Intel® Debugger (IDB)
Features: Vectors

- •Vectors (SIMD) are composed from multiple elements of same Type
- •IDB offers flexible Visualization and Access within **Vector Evaluations & Registers** Views



Vector Evaluations

int8	0	1	2	3	4	5	6	7
result	00	00	00	00	00	00	00	00

Vector Registers

int32	0
$xmm0	00000000
$xmm1	2F2F2F2F
$xmm2	dd
$xmm3	00000000
$xmm4	00000000
$xmm5	00000000
$xmm6	00000000
$xmm7	00000000

Vector Registers (Format: [✓ Hexadecimal])

int8	0	1	2	3	4	5
$xmm0	00	00	00	00	01	00
$xmm1	2f	2f	2f	2f	2f	2f
$xmm2	bd	01	00	00	00	00
$xmm3	00	00	00	00	00	00
$xmm4	00	00	00	00	00	00
$xmm5	00	00	00	00	00	00
$xmm6	00	00	00	00	00	00
$xmm7	00	00	00	00	00	00

Vector Registers (Size: [✓ INT8])

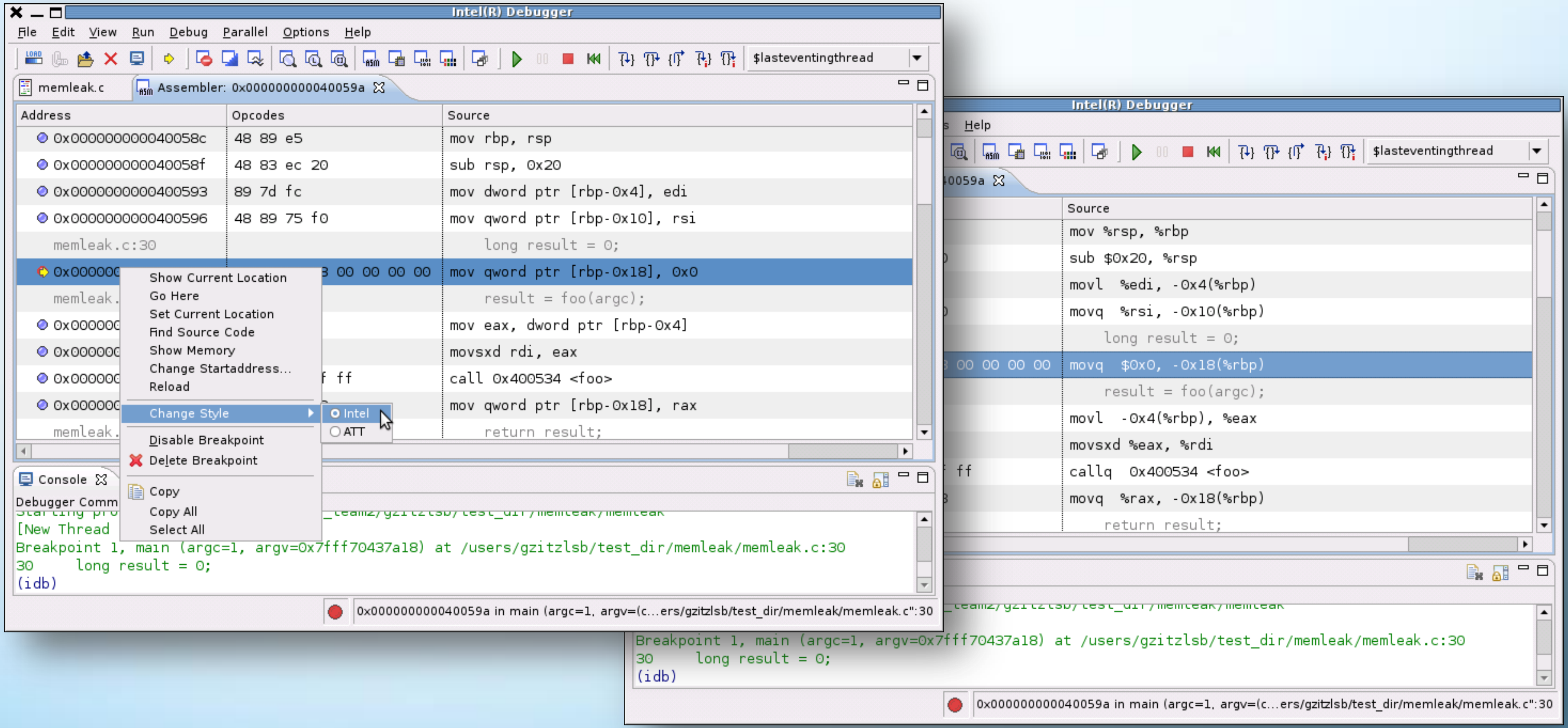
int8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$xmm0	00	00	00	00	01											
$xmm1	2f	2f	2f	2f	2f											
$xmm2	bd	01	00	00	00											
$xmm3	00	00	00	00	00											
$xmm4	00	00	00	00	00									00	00	00
$xmm5	00	00	00	00	00									00	00	00
$xmm6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$xmm7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Software & Services Group
Developer Products Division

Optimization Notice

Intel® Debugger (IDB)
Features: Assembly View

- IDB supports Intel (default) and AT&T style Assembly
- Assembly is shown when no Source is known (can be customized)



Address	Opcodes	Source
0x000000000040058c	48 89 e5	mov rbp, rsp
0x000000000040058f	48 83 ec 20	sub rsp, 0x20
0x0000000000400593	89 7d fc	mov dword ptr [rbp-0x4], edi
0x0000000000400596	48 89 75 f0	mov qword ptr [rbp-0x10], rsi
memleak.c:30		long result = 0;
0x000000	3 00 00 00 00	mov qword ptr [rbp-0x18], 0x0
memleak.		result = foo(argc);
0x000000		mov eax, dword ptr [rbp-0x4]
0x000000		movsxd rdi, eax
0x000000	f ff	call 0x400534 <foo>
0x000000		mov qword ptr [rbp-0x18], rax
memleak.		return result;

```
[New Thread
Breakpoint 1, main (argc=1, argv=0x7fff70437a18) at /users/gzitzlsb/test_dir/memleak/memleak.c:30
30      long result = 0;
(idb)
```

0x000000000040059a in main (argc=1, argv=(c...ers/gzitzlsb/test_dir/memleak/memleak.c":30

```
mov %rsp, %rbp
sub $0x20, %rsp
movl  %edi, -0x4(%rbp)
movq  %rsi, -0x10(%rbp)
    long result = 0;
movq  $0x0, -0x18(%rbp)
    result = foo(argc);
movl  -0x4(%rbp), %eax
movsxd %eax, %rdi
callq  0x400534 <foo>
movq  %rax, -0x18(%rbp)
    return result;
```

```
Breakpoint 1, main (argc=1, argv=0x7fff70437a18) at /users/gzitzlsb/test_dir/memleak/memleak.c:30
30      long result = 0;
(idb)
```

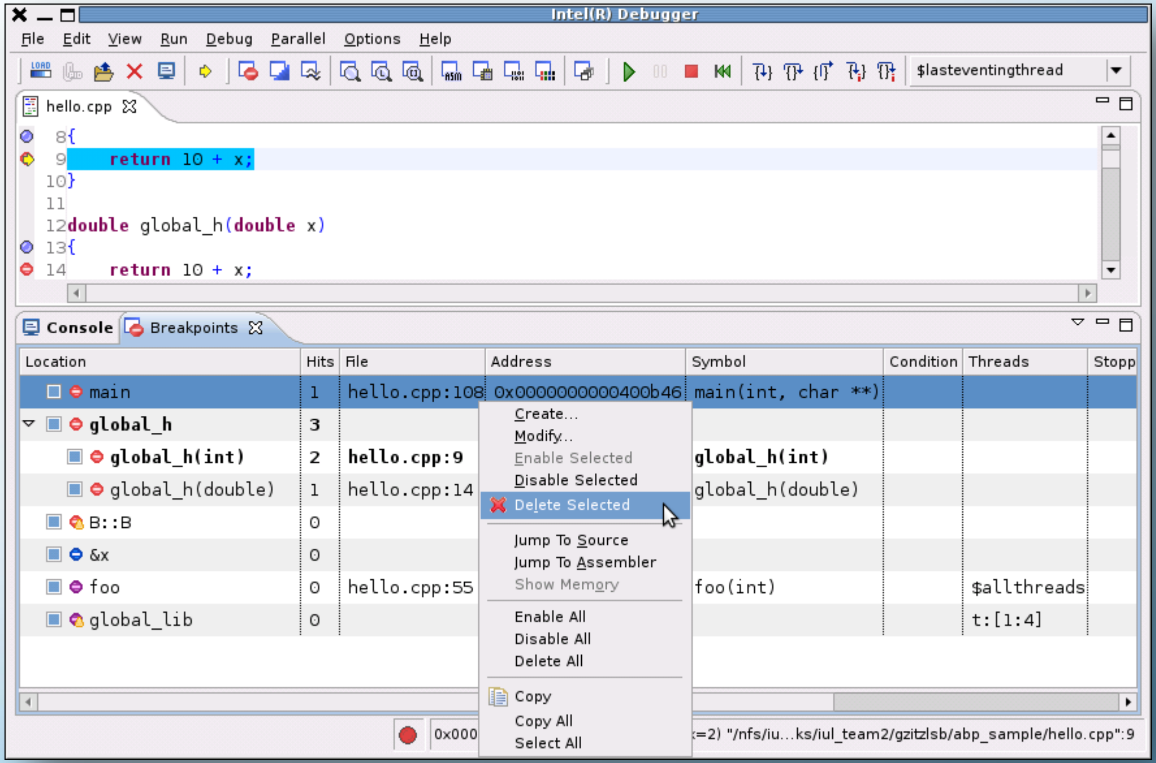
0x000000000040059a in main (argc=1, argv=(c...ers/gzitzlsb/test_dir/memleak/memleak.c":30

Intel® Debugger (IDB)
Features: Breakpoints I

- IDB supports different kinds of Breakpoints:
  - Code Breakpoints
  - Thread Synchronization Breakpoints
  - Data Breakpoints

- Dynamic multiplicity*:
  - 1:1:
    one address
  - 1:n (n > 1):
    multiple addresses
  - 1:0:
    no address (yet)

*: Not for Data Breakpoints



```
 8{
 9    return 10 + x;
10}
11
12double global_h(double x)
13{
14    return 10 + x;
```

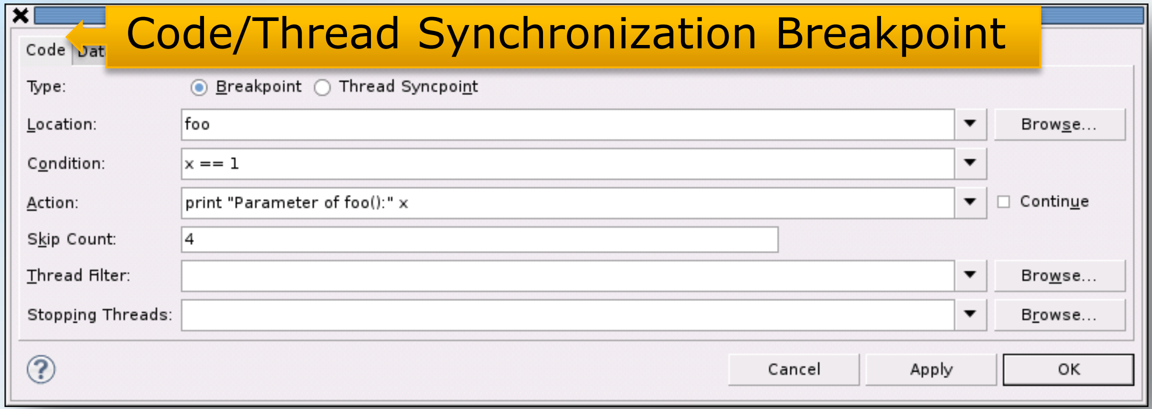
Location	Hits	File	Address	Symbol	Condition	Threads	Stopp
main	1	hello.cpp:108	0x0000000000400b46	main(int, char **)			
global_h	3						
global_h(int)	2	hello.cpp:9		global_h(int)			
global_h(double)	1	hello.cpp:14		global_h(double)			
B::B	0						
&x	0						
foo	0	hello.cpp:55		foo(int)		$allthreads	
global_lib	0					t:[1:4]	

0x000 … =2) "/nfs/iu…ks/iul_team2/gzitzlsb/abp_sample/hello.cpp":9

Optimization Notice

Intel® Debugger (IDB)
Features: Breakpoints II

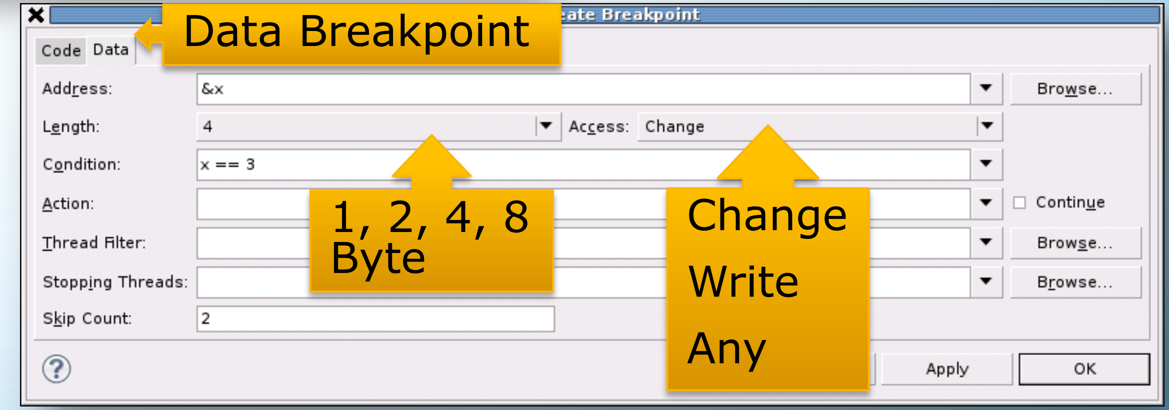
•Create or Modify Breakpoints:



Code | Dat

Type:	[✓ Breakpoint] Thread Syncpoint	
Location:	foo	Browse...
Condition:	x == 1	
Action:	print "Parameter of foo():" x	Continue
Skip Count:	4	
Thread Filter:		Browse...
Stopping Threads:		Browse...

•Flexible Attributes:
- •Condition
- •Skip Count
- •Action
- •more later…

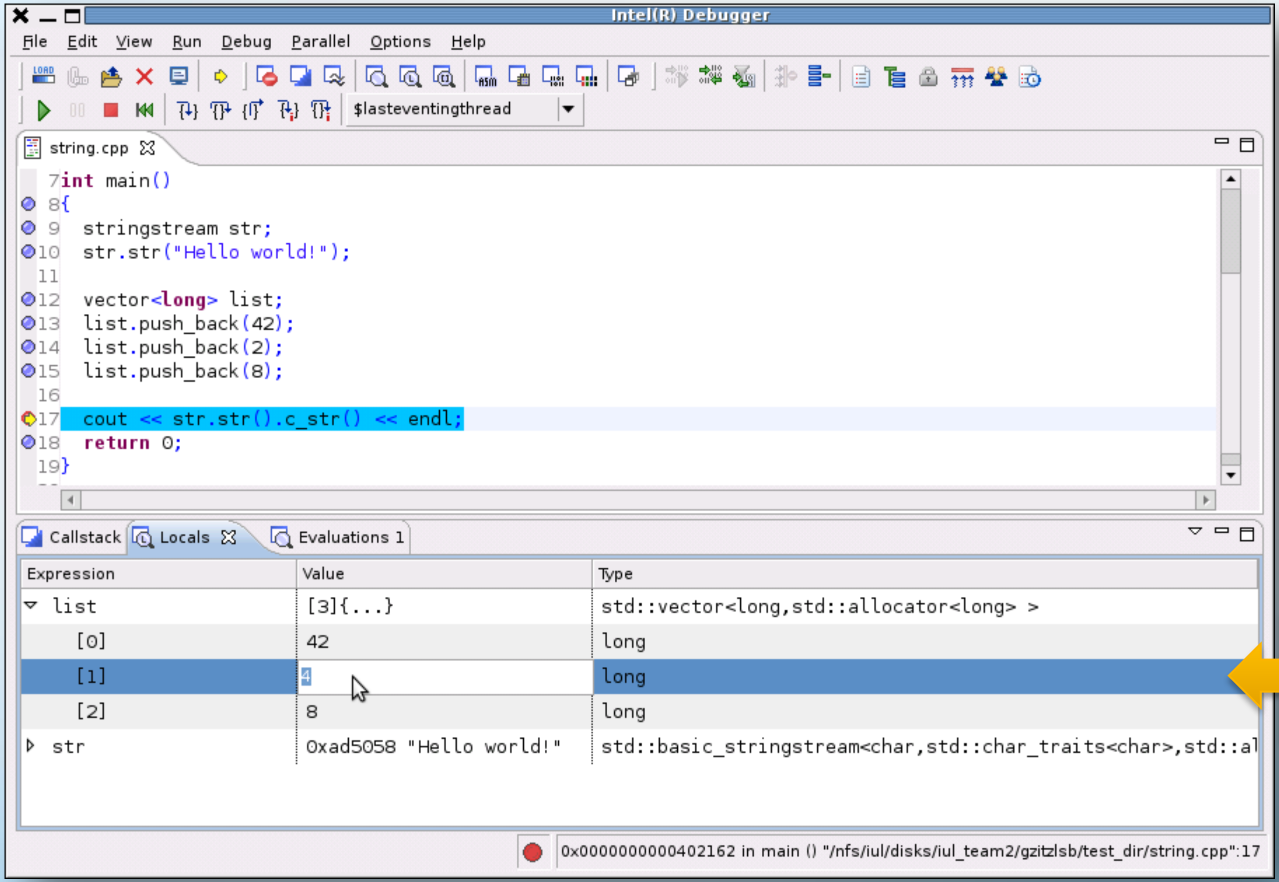


Address:	&x	Browse...
Length:	4	Access: Change
Condition:	x == 3	
Action:		Continue
Thread Filter:		Browse...
Stopping Threads:		Browse...
Skip Count:	2	

•**Browse** buttons aid in selecting proper values:
- •Symbol Selector
- •Thread Set Selector

Intel® Debugger (IDB)
Features: C++ Template Support

•"Pretty print" C++ Templates in **Locals, (Vector) Evaluations** Views and also edit most of them:


Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
$lasteventingthread
string.cpp
Callstack Locals Evaluations 1

```
int main()
{
  stringstream str;
  str.str("Hello world!");

  vector<long> list;
  list.push_back(42);
  list.push_back(2);
  list.push_back(8);

  cout << str.str().c_str() << endl;
  return 0;
}
```

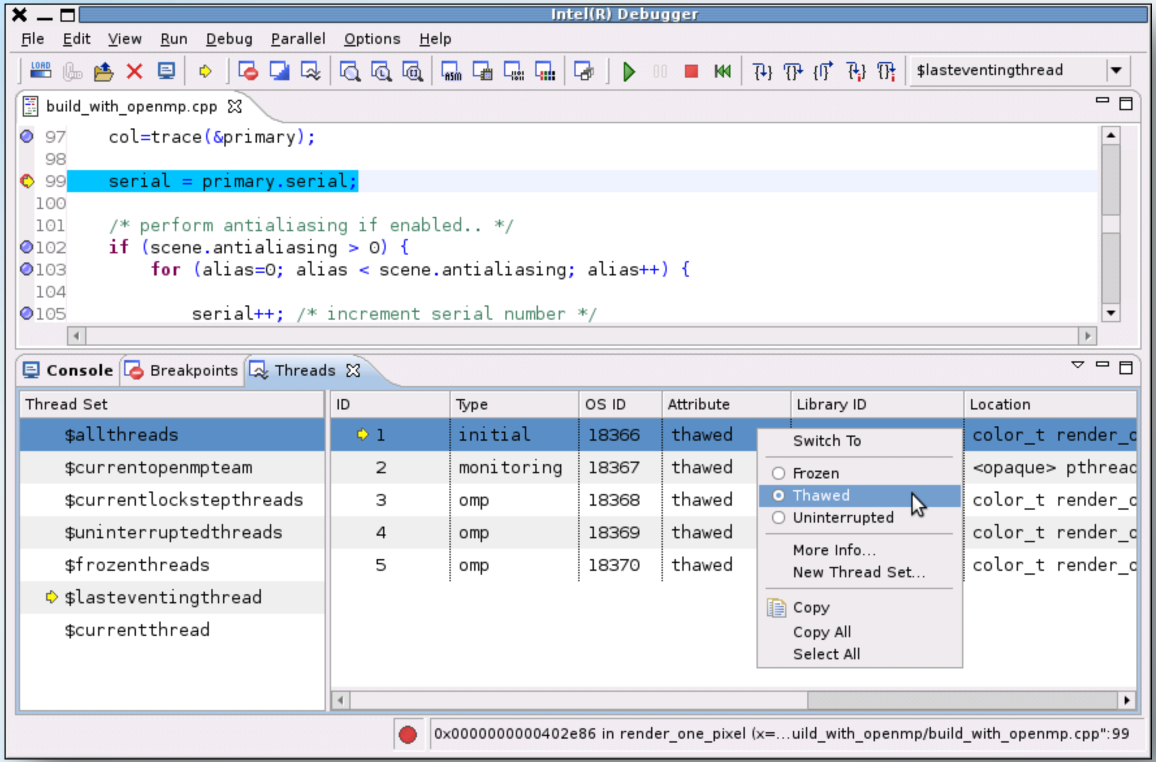
Expression	Value	Type
list	[3]{...}	std::vector<long,std::allocator<long> >
[0]	42	long
[1]	4	long
[2]	8	long
str	0xad5058 "Hello world!"	std::basic_stringstream<char,std::char_traits<char>,std::al

```
0x0000000000402162 in main () "/nfs/iul/disks/iul_team2/gzitzlsb/test_dir/string.cpp":17
```

"Pretty Print" & Edit

Optimization Notice

Intel® Debugger (IDB)
Features: Multithreading Support I



```
    col=trace(&primary);

    serial = primary.serial;

    /* perform antialiasing if enabled.. */
    if (scene.antialiasing > 0) {
        for (alias=0; alias < scene.antialiasing; alias++) {

            serial++; /* increment serial number */
```

Thread Set
$allthreads
$currentopenmpteam
$currentlockstepthreads
$uninterruptedthreads
$frozenthreads
$lasteventingthread
$currentthread

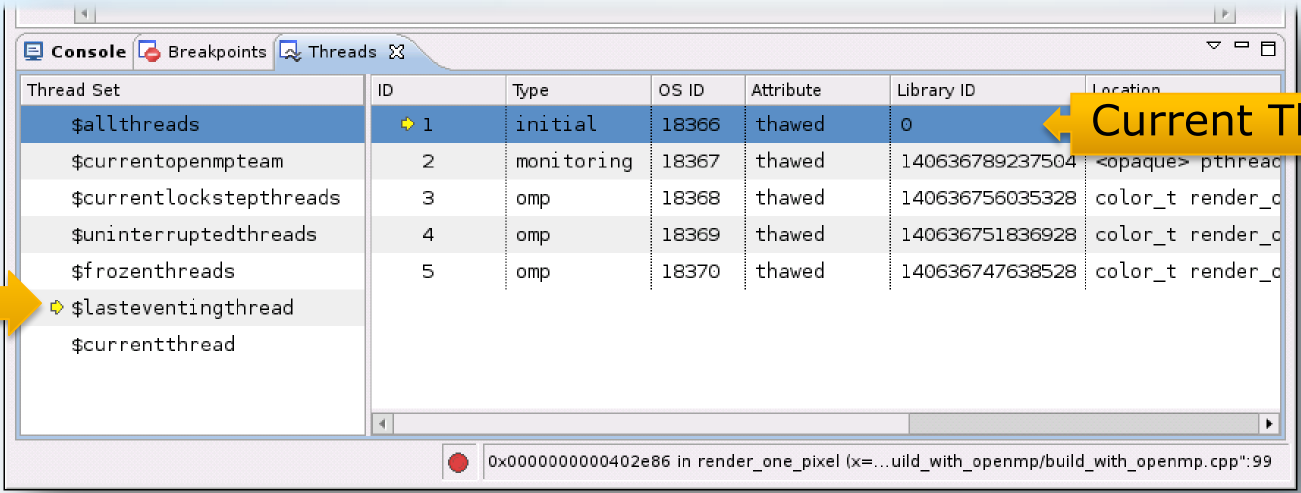
ID	Type	OS ID	Attribute	Library ID	Location
1	initial	18366	thawed		color_t render_c
2	monitoring	18367	thawed		<opaque> pthreac
3	omp	18368	thawed		color_t render_c
4	omp	18369	thawed		color_t render_c
5	omp	18370	thawed		color_t render_c

```
0x0000000000402e86 in render_one_pixel (x=...uild_with_openmp/build_with_openmp.cpp":99
```

- Thread Types:
  - **Native (POSIX)**
  - **OpenMP**
  - **TBB**
  - **Cilk**

- Stop event (e.g. Breakpoint) interrupts all threads, except for **Uninterrupted** Threads
- Only Threads that are **Thawed** can continue execution
- **Frozen** Threads remain interrupted at all time

Intel® Debugger (IDB)
Features: Multithreading Support II



Thread Set
$allthreads
$currentopenmpteam
$currentlocksteptthreads
$uninterruptedthreads
$frozenthreads
$lasteventingthread
$currentthread

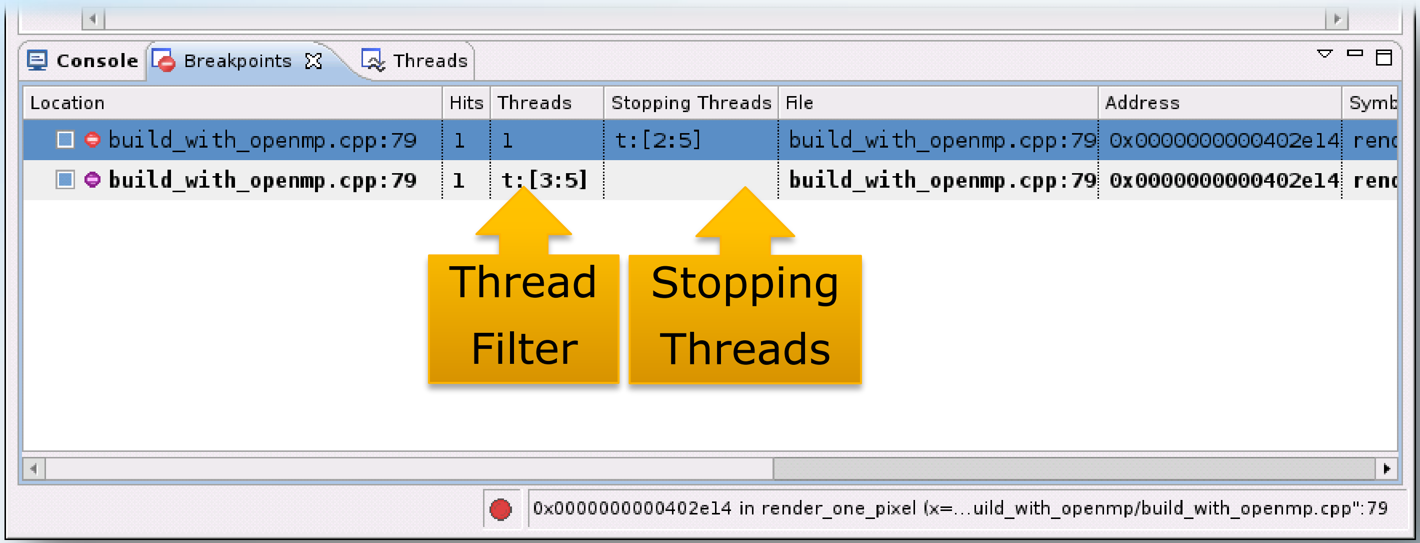
ID	Type	OS ID	Attribute	Library ID	Location
1	initial	18366	thawed	0	
2	monitoring	18367	thawed	140636789237504	<opaque> pthreac
3	omp	18368	thawed	140636756035328	color_t render_c
4	omp	18369	thawed	140636751836928	color_t render_c
5	omp	18370	thawed	140636747638528	color_t render_c

```
0x0000000000402e86 in render_one_pixel (x=...uild_with_openmp/build_with_openmp.cpp":99
```

•The **Current Thread Set** is the set of Threads affected by Run Control
•**Current Thread** defines what Data Views show, if related to Thread Context (e.g. Registers of a Current Thread)

Intel® Debugger (IDB)

Features: Multithreading Support III



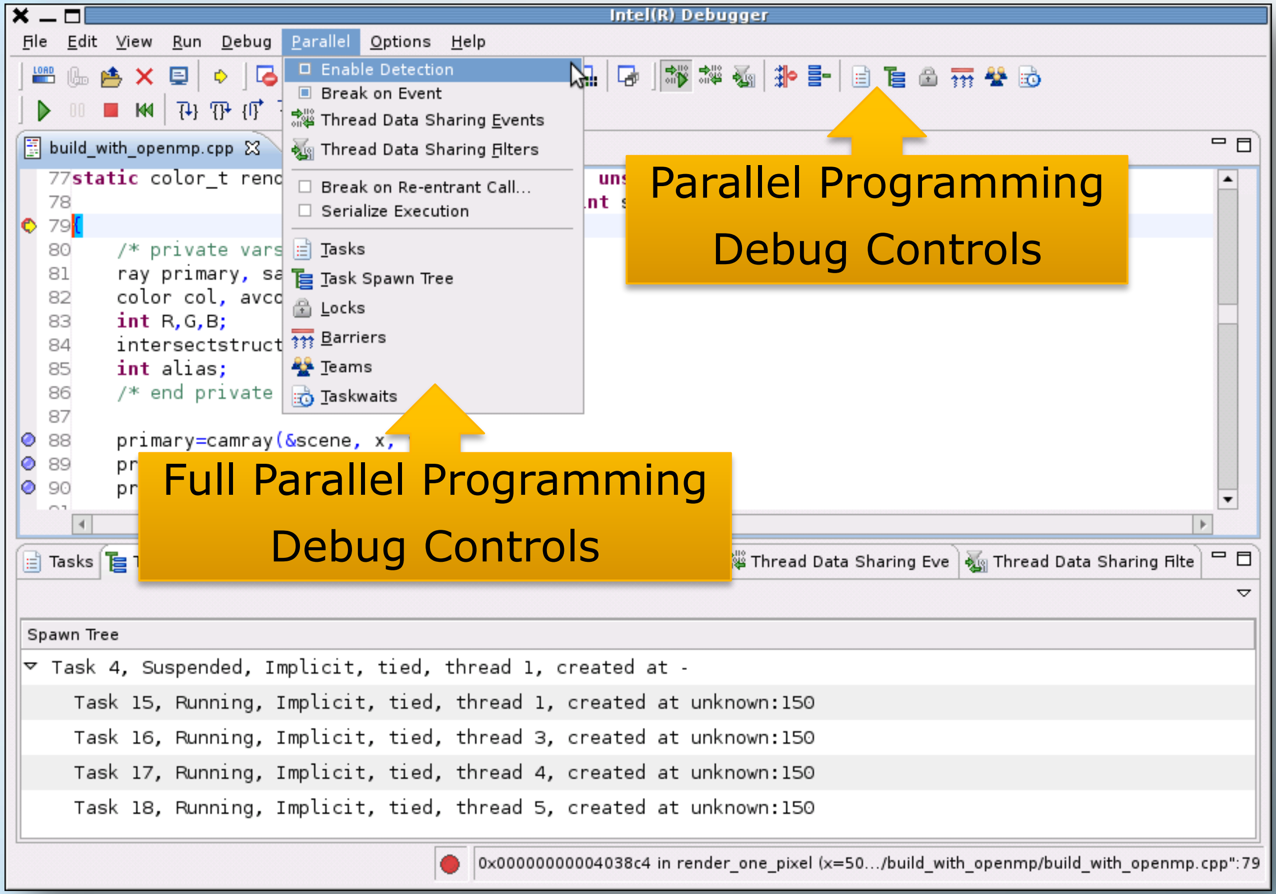
Location	Hits	Threads	Stopping Threads	File	Address	Symb
build_with_openmp.cpp:79	1	1	t:[2:5]	build_with_openmp.cpp:79	0x0000000000402e14	rend
build_with_openmp.cpp:79	1	t:[3:5]		build_with_openmp.cpp:79	0x0000000000402e14	rend

0x0000000000402e14 in render_one_pixel (x=...uild_with_openmp/build_with_openmp.cpp":79

- Breakpoints assist in Multithread Debugging:
  - **Thread Filter** of Code & Data Breakpoints
  - **Thread Filter** of Thread Synchronization Breakpoints create a Barrier – specified Threads of Set stop at Barrier and only continue if all Threads reached the Barrier
  - **Stopping Threads** will stop the specified Threads

Intel® Debugger (IDB)

Features: Support for Parallel Programming I



```
static color_t renc
{
    /* private vars
    ray primary, sa
    color col, avcc
    int R,G,B;
    intersectstruct
    int alias;
    /* end private

    primary=camray(&scene, x,
```

```
Spawn Tree
▽ Task 4, Suspended, Implicit, tied, thread 1, created at -
    Task 15, Running, Implicit, tied, thread 1, created at unknown:150
    Task 16, Running, Implicit, tied, thread 3, created at unknown:150
    Task 17, Running, Implicit, tied, thread 4, created at unknown:150
    Task 18, Running, Implicit, tied, thread 5, created at unknown:150
```

```
0x00000000004038c4 in render_one_pixel (x=50.../build_with_openmp/build_with_openmp.cpp":79
```

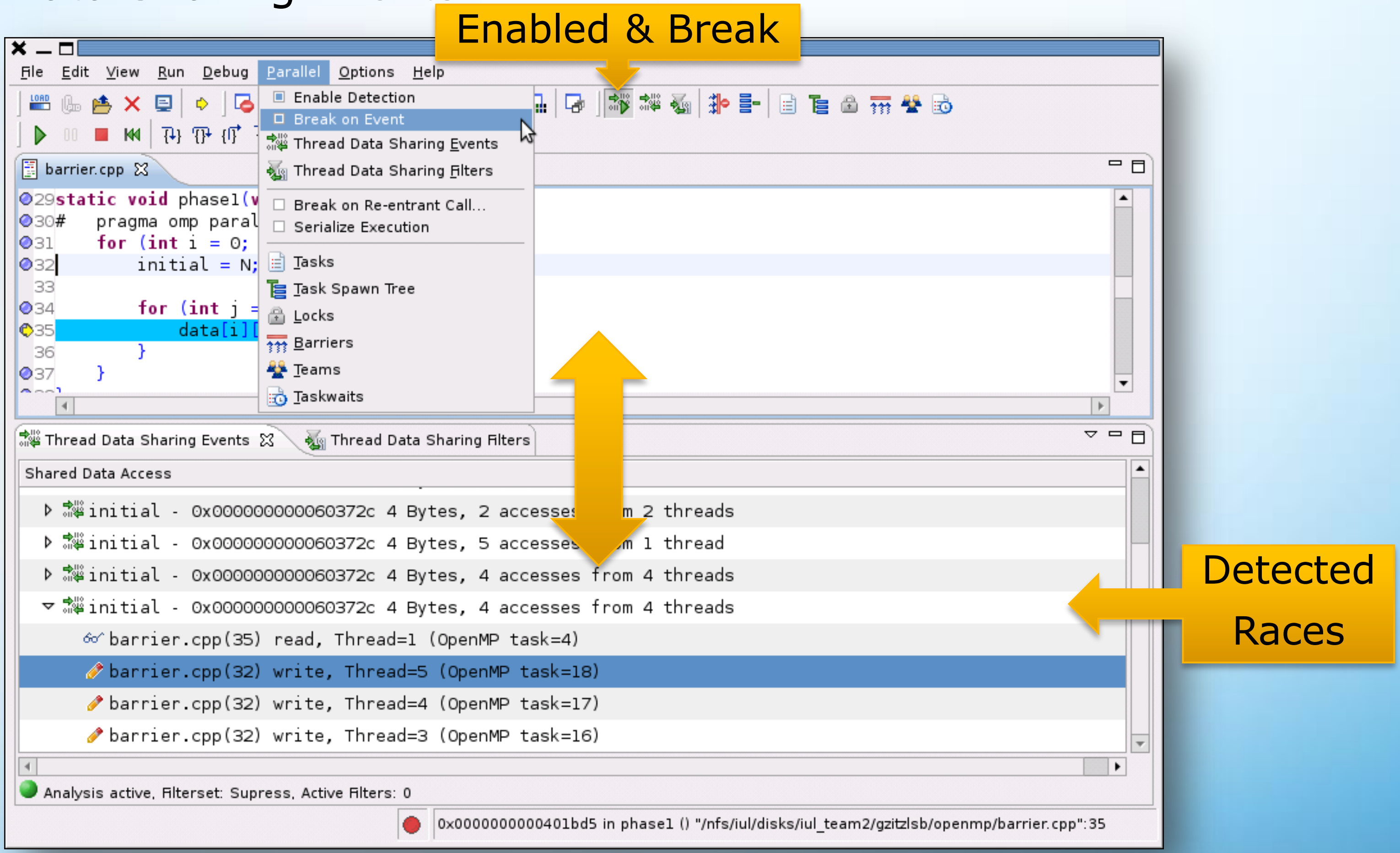
Intel® Debugger (IDB)

Features: Support for Parallel Programming I

- To use Parallel Programming Requirements enable **Compiler Option**: `-debug parallel`
- For Thread Data Sharing Detection the Application Code will be instrumented:
    - Calls to library routines added for memory accesses.
    - These library routines will determine if read/write accesses to the same memory location are done by different threads (without synchronizations) during execution. Such are Race Conditions.
    - Works for Native, TBB, Cilk Plus & OpenMP Threads

Intel® Debugger (IDB)
Features: Support for Parallel Programming II

- Thread Data Sharing Events:



Enabled & Break
Detected Races
File Edit View Run Debug Parallel Options Help
Enable Detection
Break on Event
Thread Data Sharing Events
Thread Data Sharing Filters
Break on Re-entrant Call...
Serialize Execution
Tasks
Task Spawn Tree
Locks
Barriers
Teams
Taskwaits
barrier.cpp
Thread Data Sharing Events
Thread Data Sharing Filters

```
29static void phase1(v
30#    pragma omp paral
31     for (int i = 0;
32         initial = N;
33
34         for (int j =
35             data[i][
36         }
37     }
```

```
Shared Data Access
▷ initial - 0x000000000060372c 4 Bytes, 2 accesses ... m 2 threads
▷ initial - 0x000000000060372c 4 Bytes, 5 accesses ... m 1 thread
▷ initial - 0x000000000060372c 4 Bytes, 4 accesses from 4 threads
▽ initial - 0x000000000060372c 4 Bytes, 4 accesses from 4 threads
      barrier.cpp(35) read, Thread=1 (OpenMP task=4)
      barrier.cpp(32) write, Thread=5 (OpenMP task=18)
      barrier.cpp(32) write, Thread=4 (OpenMP task=17)
      barrier.cpp(32) write, Thread=3 (OpenMP task=16)
```

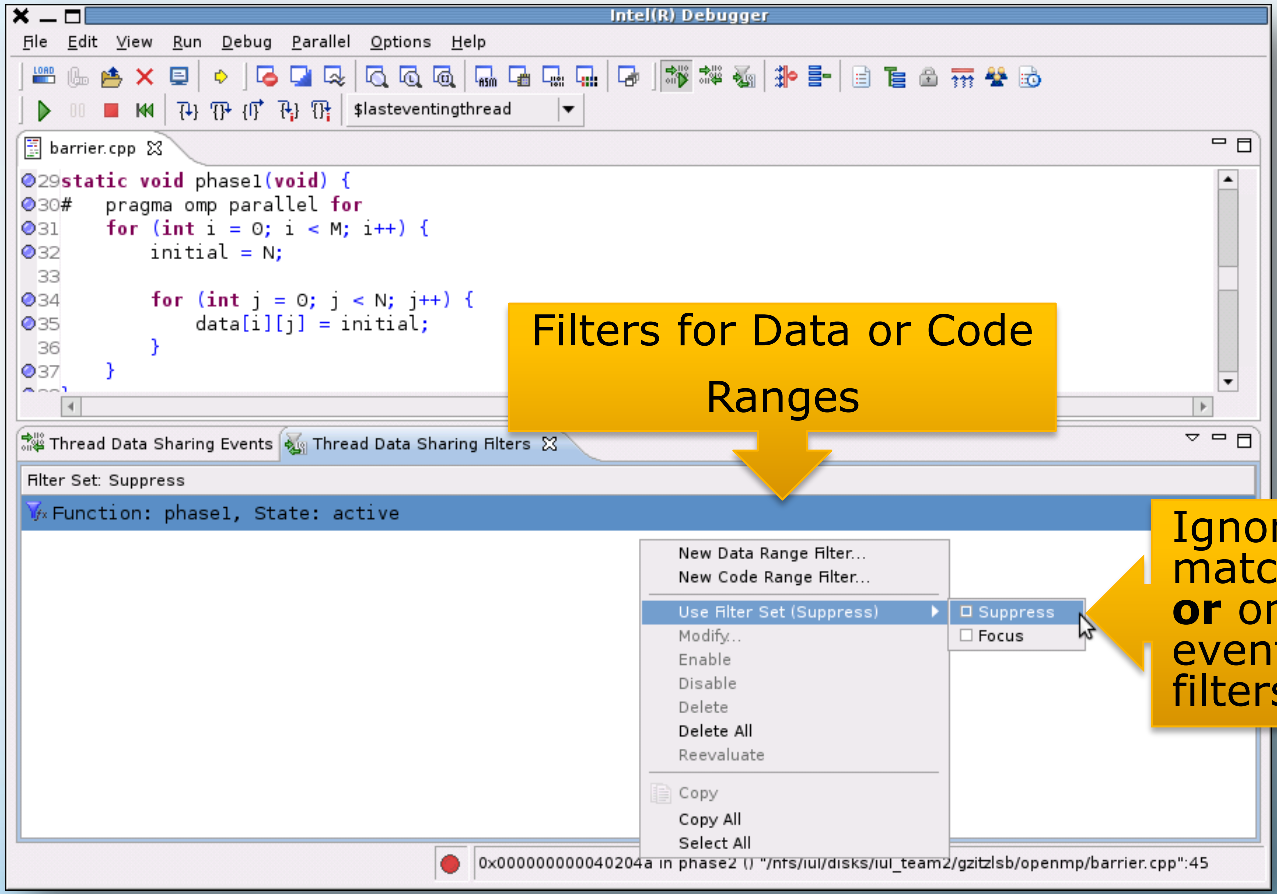
Analysis active, Filterset: Supress, Active Filters: 0

0x0000000000401bd5 in phase1 () "/nfs/iul/disks/iul_team2/gzitzlsb/openmp/barrier.cpp":35

Optimization Notice

Intel® Debugger (IDB)
Features: Support for Parallel Programming III

•Thread Data Sharing Filters:



```
static void phase1(void) {
#   pragma omp parallel for
    for (int i = 0; i < M; i++) {
        initial = N;

        for (int j = 0; j < N; j++) {
            data[i][j] = initial;
        }
    }
```

Filter Set: Suppress

Function: phase1, State: active

0x000000000040204a in phase2 () "/nfs/iul/disks/iul_team2/gzitzlsb/openmp/barrier.cpp":45

Filters for Data or Code Ranges

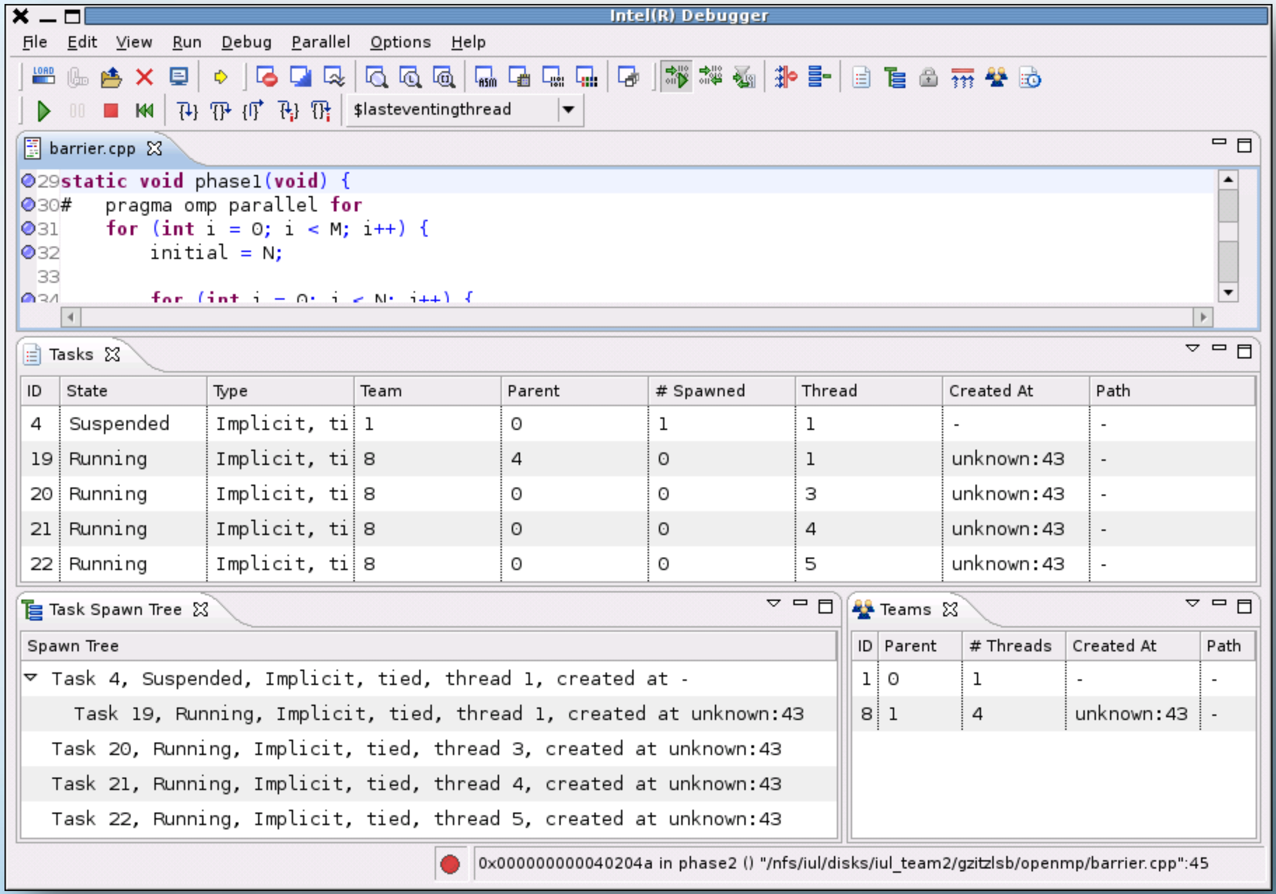
Ignore events matching filters **or** only show events matching filters (focus)

Software & Services Group
Developer Products Division

Optimization Notice

Intel® Debugger (IDB)
Features: Support for Parallel Programming IV

- Tasks, Task Spawn Tree & Teams Views:



```
static void phase1(void) {
#   pragma omp parallel for
    for (int i = 0; i < M; i++) {
        initial = N;

        for (int i = 0; i < N; i++) {
```

ID	State	Type	Team	Parent	# Spawned	Thread	Created At	Path
4	Suspended	Implicit, ti	1	0	1	1	-	-
19	Running	Implicit, ti	8	4	0	1	unknown:43	-
20	Running	Implicit, ti	8	0	0	3	unknown:43	-
21	Running	Implicit, ti	8	0	0	4	unknown:43	-
22	Running	Implicit, ti	8	0	0	5	unknown:43	-

```
Spawn Tree
▽ Task 4, Suspended, Implicit, tied, thread 1, created at -
     Task 19, Running, Implicit, tied, thread 1, created at unknown:43
  Task 20, Running, Implicit, tied, thread 3, created at unknown:43
  Task 21, Running, Implicit, tied, thread 4, created at unknown:43
  Task 22, Running, Implicit, tied, thread 5, created at unknown:43
```

ID	Parent	# Threads	Created At	Path
1	0	1	-	-
8	1	4	unknown:43	-

0x000000000040204a in phase2 () "/nfs/iul/disks/iul_team2/gzitzlsb/openmp/barrier.cpp":45

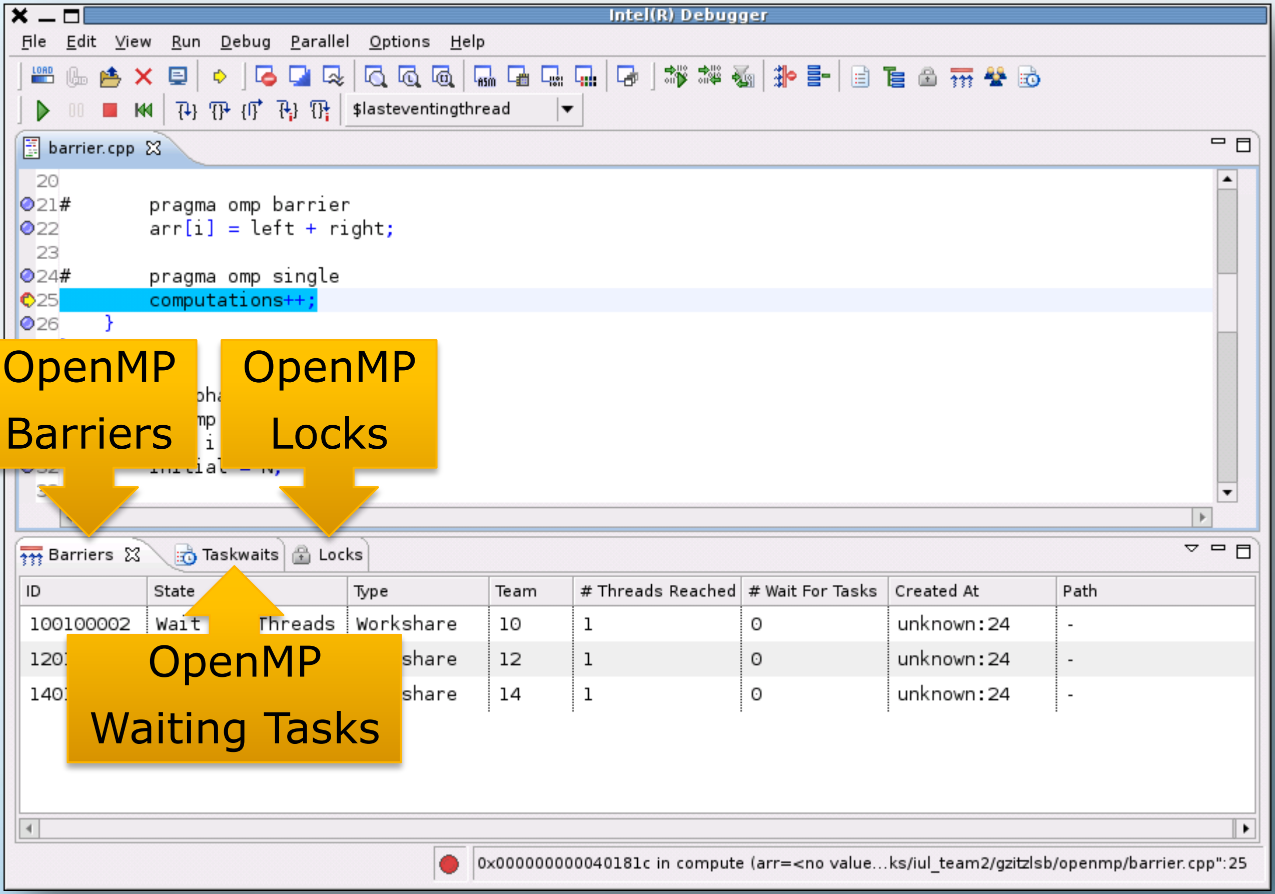
Software & Services Group
Developer Products Division

Optimization Notice

Intel® Debugger (IDB)

Features: Support for Parallel Programming V

- Barriers, Taskwaits & Locks Views:



```
20
21#        pragma omp barrier
22         arr[i] = left + right;
23
24#        pragma omp single
25         computations++;
26     }
```

ID	State	Type	Team	# Threads Reached	# Wait For Tasks	Created At	Path
100100002	Wait … Threads	Workshare	10	1	0	unknown:24	-
120		…share	12	1	0	unknown:24	-
140		…share	14	1	0	unknown:24	-

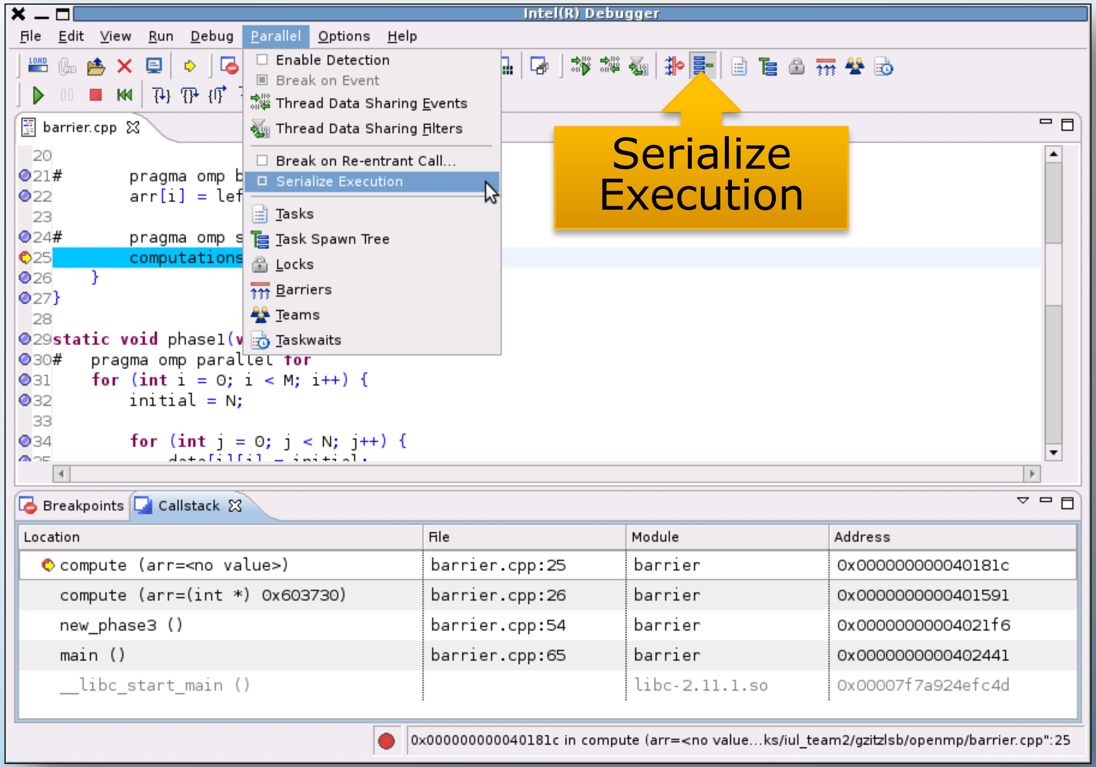
0x000000000040181c in compute (arr=<no value...ks/iul_team2/gzitzlsb/openmp/barrier.cpp":25

Optimization Notice

Intel® Debugger (IDB)

Features: Support for Parallel Programming VI

- Serialize Execution:
  - Execute **OpenMP** & **Cilk Plus** Code in one Thread
  - Distinguish Common Errors & Parallelization Errors (one click)



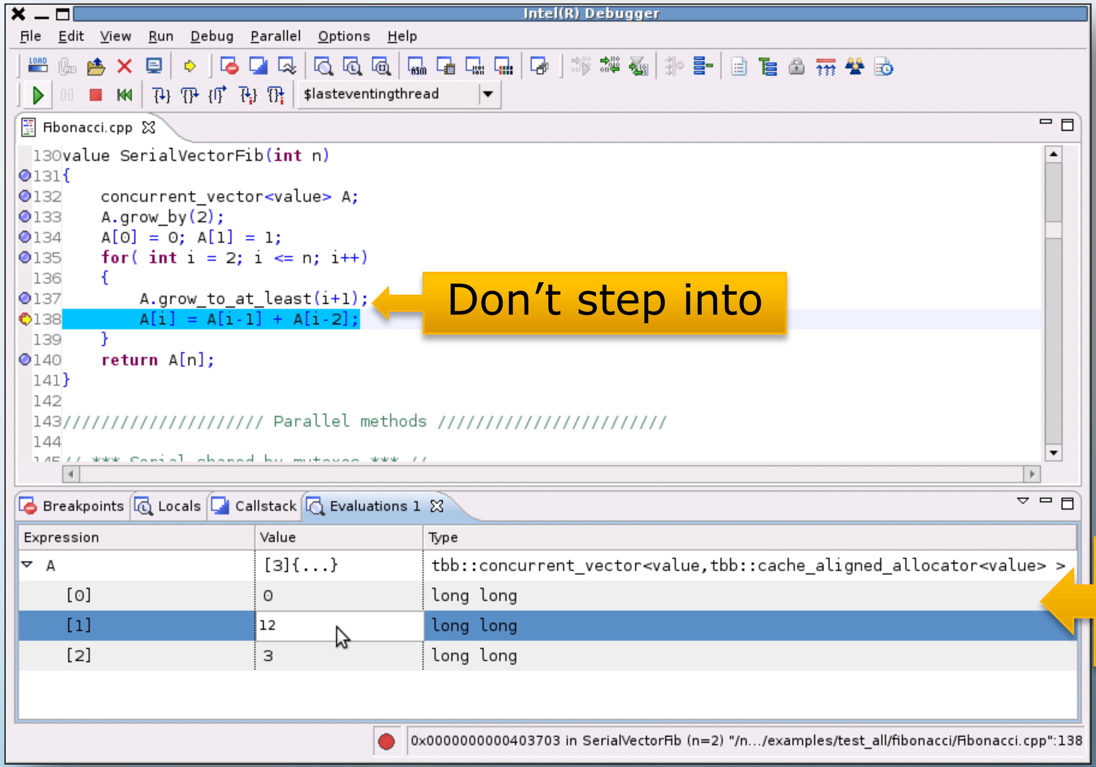
```
#        pragma omp b
         arr[i] = lef

#        pragma omp s
         computations
     }
}

static void phase1(v
#    pragma omp parallel for
     for (int i = 0; i < M; i++) {
         initial = N;

         for (int j = 0; j < N; j++) {
             data[i][j] = initial;
```

Location	File	Module	Address
compute (arr=<no value>)	barrier.cpp:25	barrier	0x000000000040181c
compute (arr=(int *) 0x603730)	barrier.cpp:26	barrier	0x0000000000401591
new_phase3 ()	barrier.cpp:54	barrier	0x00000000004021f6
main ()	barrier.cpp:65	barrier	0x0000000000402441
__libc_start_main ()		libc-2.11.1.so	0x00007f7a924efc4d

0x000000000040181c in compute (arr=<no value...ks/iul_team2/gzitzlsb/openmp/barrier.cpp":25

Intel® Debugger (IDB)
Features: Threading Building Blocks (TBB) Support

- TBB Containers can be "pretty printed" and even partly modified in **Locals, (Vector) Evaluations** Views
- **Step into** does not follow TBB internal Code (set `$usessteppingrules=0` to turn off default)


Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
$lasteventingthread
Fibonacci.cpp
Breakpoints Locals Callstack Evaluations 1
Don't step into
"Pretty Print" & Edit

```
value SerialVectorFib(int n)
{
    concurrent_vector<value> A;
    A.grow_by(2);
    A[0] = 0; A[1] = 1;
    for( int i = 2; i <= n; i++)
    {
        A.grow_to_at_least(i+1);
        A[i] = A[i-1] + A[i-2];
    }
    return A[n];
}

//////////////////// Parallel methods ///////////////////////
```

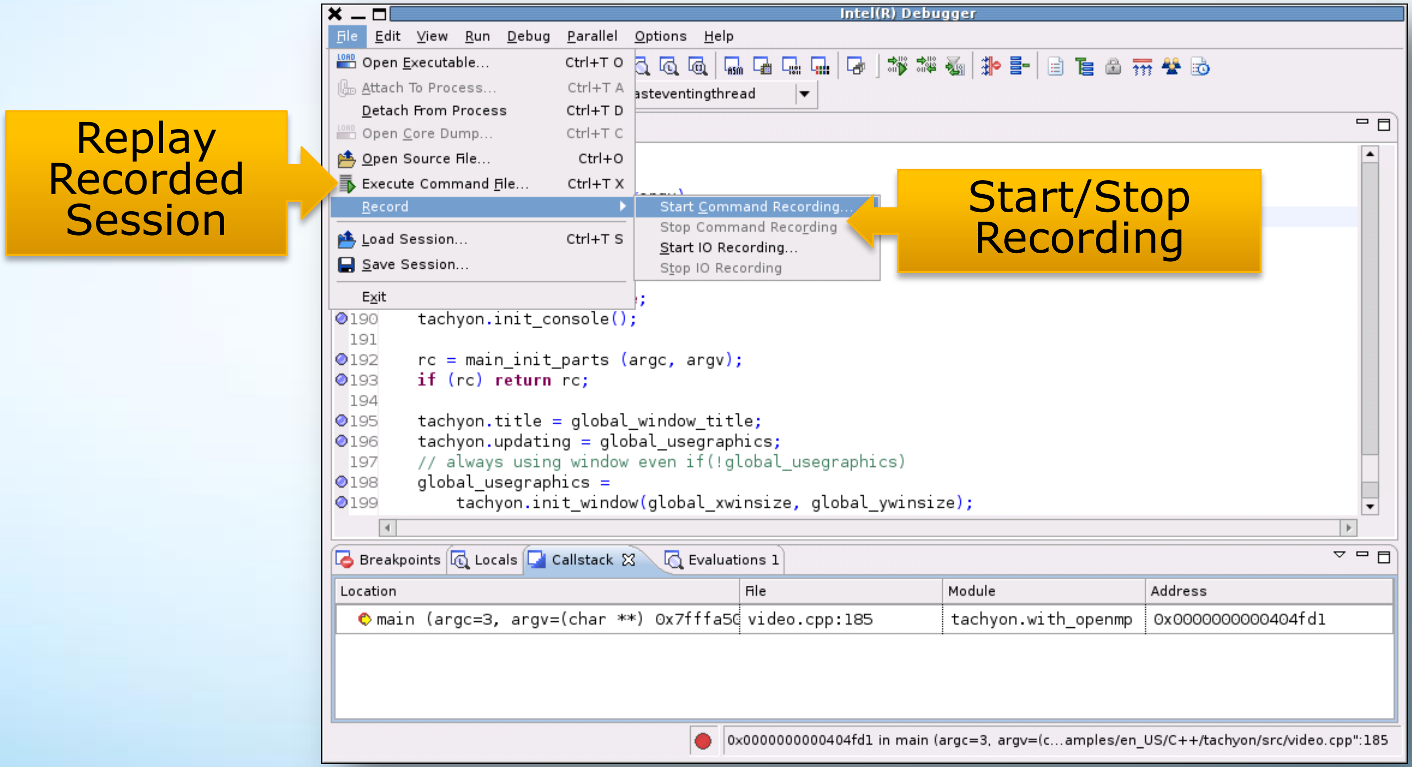
Expression	Value	Type
A	[3]{...}	tbb::concurrent_vector<value,tbb::cache_aligned_allocator<value> >
[0]	0	long long
[1]	12	long long
[2]	3	long long

0x0000000000403703 in SerialVectorFib (n=2) "/n.../examples/test_all/fibonacci/Fibonacci.cpp":138

Optimization Notice

Intel® Debugger (IDB)
Features: Record & Execute Command File

- **Record** a session – all I/O or just commands/actions during a session
- Replay via **Execute Command File**



Replay Recorded Session
Start/Stop Recording
Intel(R) Debugger
File Edit View Run Debug Parallel Options Help
Open Executable... Ctrl+T O
Attach To Process... Ctrl+T A
Detach From Process Ctrl+T D
Open Core Dump... Ctrl+T C
Open Source File... Ctrl+O
Execute Command File... Ctrl+T X
Record
Load Session... Ctrl+T S
Save Session...
Exit
Start Command Recording...
Stop Command Recording
Start IO Recording...
Stop IO Recording
asteventingthread
Breakpoints
Locals
Callstack
Evaluations 1

```
    tachyon.init_console();

    rc = main_init_parts (argc, argv);
    if (rc) return rc;

    tachyon.title = global_window_title;
    tachyon.updating = global_usegraphics;
    // always using window even if(!global_usegraphics)
    global_usegraphics =
        tachyon.init_window(global_xwinsize, global_ywinsize);
```

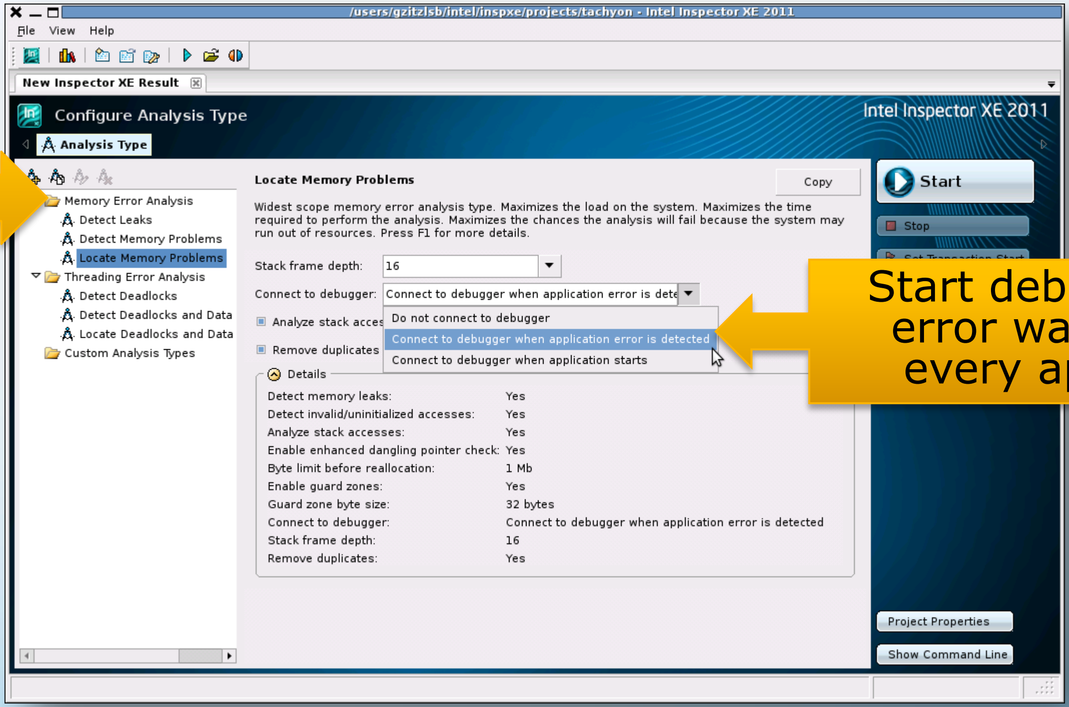
Location	File	Module	Address
main (argc=3, argv=(char **) 0x7fffa5(	video.cpp:185	tachyon.with_openmp	0x0000000000404fd1

```
0x0000000000404fd1 in main (argc=3, argv=(c...amples/en_US/C++/tachyon/src/video.cpp":185
```

Intel® Debugger (IDB)
IDB + Inspector XE (memory analysis)

- IDB can be used with Inspector XE as debugger during memory analysis.
- For Inspector XE configure IDB as debugger via setting environment variable **$INSPXE_DEBUGGER** (if not set GDB is the default)

IDB supports all 3 types of memory analysis



Locate Memory Problems

Widest scope memory error analysis type. Maximizes the load on the system. Maximizes the time required to perform the analysis. Maximizes the chances the analysis will fail because the system may run out of resources. Press F1 for more details.

Stack frame depth: 16

Connect to debugger: Connect to debugger when application error is dete

- Do not connect to debugger
- Connect to debugger when application error is detected
- Connect to debugger when application starts

Analyze stack acces

Remove duplicates

Details

Detect memory leaks:	Yes
Detect invalid/uninitialized accesses:	Yes
Analyze stack accesses:	Yes
Enable enhanced dangling pointer check:	Yes
Byte limit before reallocation:	1 Mb
Enable guard zones:	Yes
Guard zone byte size:	32 bytes
Connect to debugger:	Connect to debugger when application error is detected
Stack frame depth:	16
Remove duplicates:	Yes

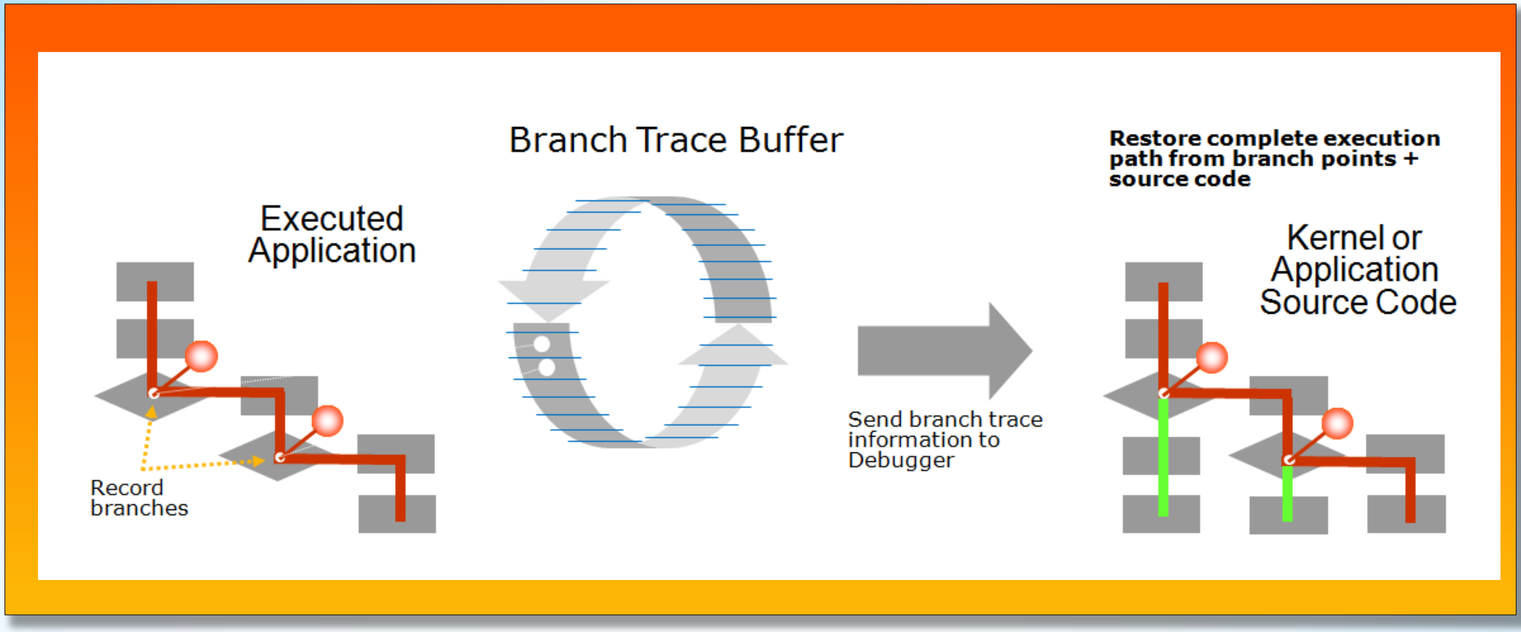
Start debugger when first error was found or with every application start

Optimization Notice

Intel GDB* Enhancements for Linux* and OS X*

- Intel® Debugger (Intel® IDB) is deprecated:
  - No changes since Composer XE 2013 (13.0)
  - May be removed in a future release
- New debugger solution is based on GNU Project Debugger (GDB)* and includes additional improvements & features:
  - Improved Fortran support
  - GNU GDB 7.5 based
  - Python* 2.6 or greater required
- Linux only:
  - BTrace: Trace back branches taken before a crash
  - PDBX: Parallel Debug Extensions for data race detection
  - Pointer Checker: Assist in finding pointer issues
  - Intel® Transactional Synchronization Extensions

Btrace – Based on Branch Trace Store


Branch Trace Buffer
Executed Application
Record branches
Send branch trace information to Debugger
Restore complete execution path from branch points + source code
Kernel or Application Source Code

- Intel® Atom™ Processor and Intel® Core™ Family processors use *Branch Trace Store* (BTS) to keep track of branch (“return”) targets for calls
- BTRACE uses this buffer to display stack trace “even in difficult cases” where program execution stack is corrupted completely

Btrace: Example

Program crashed and back trace not available

```
(gdb) btrace enable auto
(gdb) run
Starting program: ../gdb/trace/examples/function_pointer/stack64
Program received signal SIGSEGV, Segmentation fault. 0x0000002a in ?? ()
(gdb) bt

#0  0x000000000000002a in ?? ()
#1  0x0000000000000017 in ?? ()
#2  0x000000000040050e in fun_B (arg=0x4005be) at src/stack.c:32
#3  0x0000000000000000 in ?? ()

Look at the branch trace.
List of blocks starts from the most recent block (ending at the current pc) and
continues towards older blocks such that control flows from block n+1 to block n.

(gdb) btrace list 1-7
1   in ?? ()
2   in fun_B () at src/stack.c:36-37
3   in fun_B () at src/stack.c:32-34
4   in main () at src/stack.c:57
5   in fun_A () at src/stack.c:22-25
6   in fun_A () at src/stack.c:18-20
7   in main () at src/stack.c:51-56
```

PDBX: Data Race Detection

- **Sample**: Assume two global variables
  a=1
  b=2
- And two threads T1, T2 executing
  T1: x = a + b
  T2: b = 42
- Value of x depends on execution order:
  If T1 runs before T2 → x = 3
  If T2 runs before T1 → x = 43
- Execution order is not guaranteed unless synchronization methods are used.

```
(gdb) pdbx enable
(gdb) continue
 data race detected
 1: write answer, 4 bytes from foo.c:36
 3: read answer, 4 bytes from foo.c:40
Breakpoint -11, 0x401515 in foo () at foo.c:36
(gdb)
```

PDBX: Requirements /Notes

- Intel® Compiler 12.1 or later
- Compilation by -debug parallel option
- Core, Atom, or Xeon Phi processor
- Pthreads and /or OpenMP
  - Works too for Intel® Cilk™ Plus and Intel® TBB tasking but currently many *false positives*
- Python 2.6+ (host)
- Rather intrusive execution mode
  - Both for time and memory consumption
  - But mode can be enabled/disabled any time during debug session
- Many filters available to fine tune PDBX debugging

Optimization Notice

Backup

Packages Overview

- The Intel® Composer XE is part of the respective Intel® Parallel Studio XE series and being installed within one unique setup
  - Please refer to the Introduction Module for the Intel® Parallel Studio XE setup
- The Intel® Composer XE is also available separately and can be installed independtly from Parallel Studio XE.
  - for example when dedicated Composer updates are available

Intel® Composer XE Standalone Packages

- Installation Fileset Combinations
  - Six filesets with C/C++ and Fortran for Windows and Linux
  - C/C++ and Fortran packages sync'd on same platform are always in sync and have the same package versions and features (except of language specific deviations)
  - Optional IPP packages additionally to the compiler packages due to licensing reasons
- Visual Studio* Shell and Libraries included in the Intel® Visual Fortran Composer XE
  - Packages with VS Shell included don't require any Visual Studio installation on the system
  - This is the default package available from Intel's Registration Center product download.

Composer XE Standalone Packages (2)

- Fileset Format
  - [l|m|w]_[c|f]compxe_2013_sp1.<updatenr>.<buildnr>.[exe|tgz|dmg]
    - *w* (Windows), *l* (Linux)
    - *c* (C/C++ package), *f* (Fortran package)
    - *exe* (Windows self-extracting executable), *tgz* (Linux tarball), *dmg* (OS X package file)
- Examples:
  - w_ccompxe_2013_sp1.0.103.exe
    - Self-extracting Windows .exe file with C++ IA32 and Intel64 installers
  - l_fcompxe_2013_sp1.0.103.tgz
    - Linux archive with Fortran IA32 and Intel64 insntallers

Composer XE Optional Standalone Packages

- Optional Packages
  - Due to licensing or packaging reasons some packages may be offered as additional standalone installations
  - IPP Crypto and Fixed Size Funtion packages for C/C++ Compiler
    - l|m|w_ccompxe_[gen|crypto]_ipp_7.x.x.xxx.exe|tgz|dmg
  - Fortran packages without Visual Studio Shell and Libraries (requires an existing Visual Studio installation on the development machine)
    - w_fcompxe_novsshell_2013_sp1.x.xxx.exe

License File

- A FLEXlm* license file is installed by the setup program at
  - <common files>\intel\licenses (Windows)
  - </opt/intel/licenses> or <$HOME/intel/licenses> (Linux)
- Web users: a serial number is sent by email for web users
- CD users: a serial number is provided on the CD
  - Register to get one year free support and software updates
- Install-time license checking

Product Activation

- 3 methods
  - Using license file (.lic)
    - Licese file is sent from Intel Registration Center upon serial number registration
  - Remote activation
    - Requires Internet access
    - Product S/N is registered at the Intel Registration Center and license file is sent via email
  - Evaluation installation
    - Deos not require registration and license file

License Types

- Single user
  - Product is licensed to be installed and used by a single user
- Concurrent users
  - Product may be installed and used on as many systems as desired, but no more than N users may compile concurrently
  - See FLEXlm user guide for more details
- Special licenses
  - Evaluation licenses: single-user license that expires (typically in 30 days)
  - CD license: single-user license, support service available upon registration
  - Beta license: expires after a certain time period and is used by beta compiler only

Installation - Windows Setup

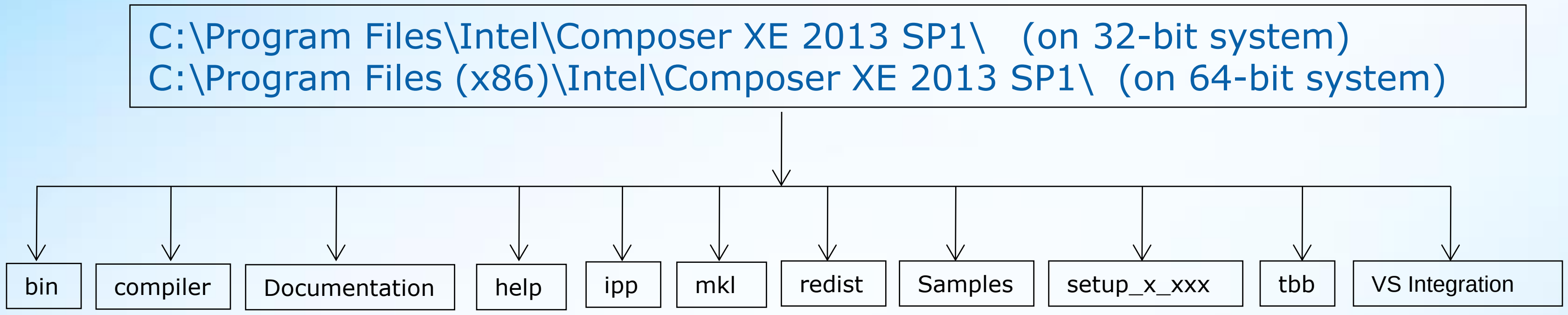
- GUI-driven and self-explaining installer
  - Self-extracting .exe file automatically starts the installer
  - Setup can be re-voked again to perform a repair/modify/remove operation of the toolsuite via
    - Windows Control Panel/Add or Remove Programs -> Intel Composer XE 2013 SP1 for Windows*
    - <installidr>\setup_x_xxx\Setup.exe, e.g. C:\Program Files (x86)\Intel\Composer XE 2013 SP1\setup_1_139\Setup.exe
    - C:\Program Files (x86)\Intel\Download\<package>\Setup.exe

Installation - Linux Setup

- Command line-driven installer/uninstaller
  - Extract file to temp dir and run installer, e.g.:

```
$ cd
$ tar -xzvf l_ccompxe_2013_sp1.1.106.tgz
$ cd l_ccompxe_2013_sp1.1.106
$ ./install  //choose root , sudo root or user installation
.....
$ <installdir> bin/uninstall.sh      //root or user uninstall, for example:
$ sudo /opt/intel/l_ccompxe_2013_sp1.1.106/bin/uninstall.sh
```

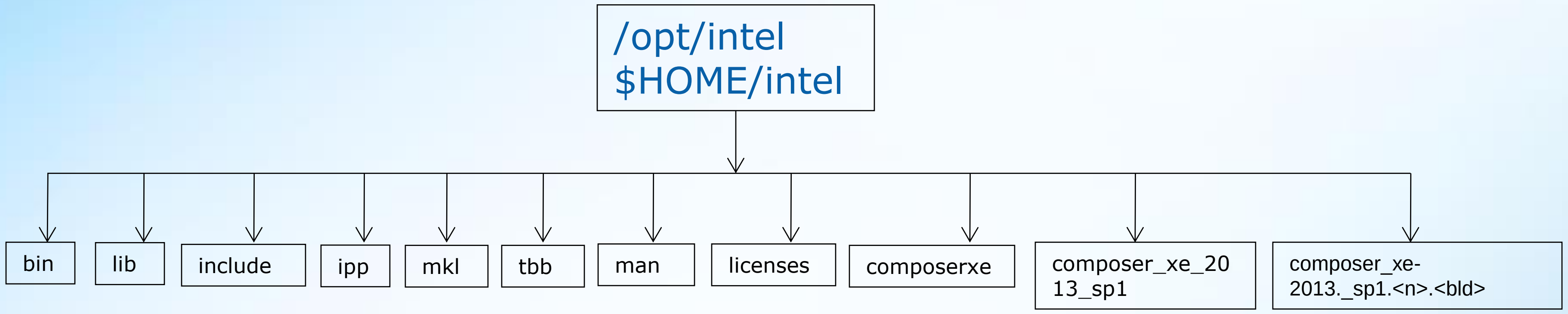
Directory Structure - Windows


C:\Program Files\Intel\Composer XE 2013 SP1\ (on 32-bit system)
C:\Program Files (x86)\Intel\Composer XE 2013 SP1\ (on 64-bit system)
bin
compiler
Documentation
help
ipp
mkl
redist
Samples
setup_x_xxx
tbb
VS Integration

If not specified otherwise during the installation, the C/C++ Compiler and Fortran Compiler are beiing installed under the same directory structure

- **bin -** start scripts, C/C++ and/or Fortran compiler executables and DLLs (including source checker)
- **compiler:** Intel specific includes and libraries
- **Documentation**: Complete documentation of compiler, debugger and libraries
- **help:** Registration files for on-line help.
- **ipp:** IPP library files and demos
- **mkl:** MKL library files with examples, tests and benchmarks
- **redist:** Redistributable libraries, can be packed with application if required on runtimes the update number **Samples:** Visual Studio projects with compiler sample source code for C/C++ and/or Fortran and IPP
- **setup_0:** Uninstall, repair or modify can be started with Setup.exe from here
- **tbb:** TBB library files with examples
- **VS Integration:** Plug-in files for Compiler integration into Visual Studio

Directory Structure – Linux/OS X



- Based on Linux standard guidelines
- **bin** contains compiler source scripts iccvars.(c)sh/ifortvars.(c)sh and symbolic links to executables that can be invoked by user
- The **lib**, **include**, **ipp**, **mkl**, and **tbb** directories are symbolic links to directories with the same name in composerxe and contain the performance library headers and libraries
- **man** contains man pages for executable commands. It will be a symbolic link to the man structure under composerxe.
- **licenses** is the default licenses files (.lic) directory
- **composerxe** is a symbolic link pointing to the composer_xe_2013_sp1 directory.
- **composer_xe_2013_sp1** is a physical directory containing links to header files and libraries that are part of the latest Composer XE product configuration.
- **composer_xe_2013_sp1.<n>.<bld>**, for example composer_xe_2013_sp1.1.106 is the directory containing the files from a Composer release. <n> is the update number (starting at 0 for RTM) and <bld> is the build number.
- If /opt is a network drive, symbolic links are created if possible. They may or may not be visible on systems on which the product was not installed.

Intel® C++ and Fortran Compiler XE 13.x
Standards Conformance

- C (`icc`):
  - ISO/IEC 9899:1990 standard (`-std=c89`)
  - C90 plus GNU extensions (`-std=gnu89` [default])
  - ISO/IEC9899:1999 standard (`-std=c99` or `-std=c9x`)
  - C99 plus GNU extensions (`-std=gnu99`)
- C++ (`icpc`):
  - ISO/IEC 14882:1998 standard plus GNU* extensions (`-std=gnu++98` [default])
    Note:
    Almost same as 2003 standard (ISO/IEC 14882:2003)
  - ISO/IEC 14882:2011 standard (`-std=c++0x`) or including GNU* extensions (`-std=gnu++0x`)
- On Windows* (`/Qstd=`), you can only specify values `c99` and `c++0x`!

Intel® C++ and Fortran Compiler XE
Standards Conformance – cont'd

- Fortran (`ifort`):
  - Fortran 90, FORTRAN 77 and FORTRAN IV (FORTRAN 66)
  - Fortran 95 (ISO/IEC 1539:1997) [baseline]
  - Fortran 2003 (ISO/IEC 1539:2004)
  - Fortran 2008 (ISO/IEC 1539:2010)

- More details here:
  http://software.intel.com/en-us/articles/intel-fortran-compiler-support-for-fortran-language-standards/

Intel® C++ and Fortran Compiler XE 13.x
Standards Conformance – C99

- Full support:
  - Restricted pointers (`restrict` keyword).
  - Variable-length Arrays
  - Flexible array members
  - Complex number support (`_Complex` keyword)
  - Hexadecimal floating-point constants
  - Compound literals
  - Designated initializers
  - Mixed declarations and code
  - Macros with a variable number of arguments
  - Inline functions (`inline` keyword)
  - Boolean type (`_Bool` keyword)
  - `long double` is 64 bit, not 128 bit (only Linux*)
- More here: http://software.intel.com/en-us/articles/c99-support-in-intel-c-compiler

Intel® C++ and Fortran Compiler XE 13.x
Standards Conformance – C++0x

- **Most important** already in Intel® C++ Composer XE 2011!
- New in Intel® C++ Composer XE 2013:
  - Additional type traits
  - Uniform initialization
  - Generalized constant expressions (partial support)
  - `noexcept`
  - Range based for loops
  - Conversions of lambdas to function pointers
  - Implicit move constructors and move assignment operators
  - Support for C++11 features in gcc 4.6 and 4.7 headers
- See full list here:
  http://software.intel.com/en-us/articles/c0x-features-supported-by-intel-c-compiler/

Note:
Features marked as "partial" are not complete but at least implemented to the extent of the typical default compiler (GNU GCC* or Microsoft Visual Studio*)!

Intel® C++ and Fortran Compiler XE 13.x
Standards Conformance – Fortran 2003

- Almost complete!

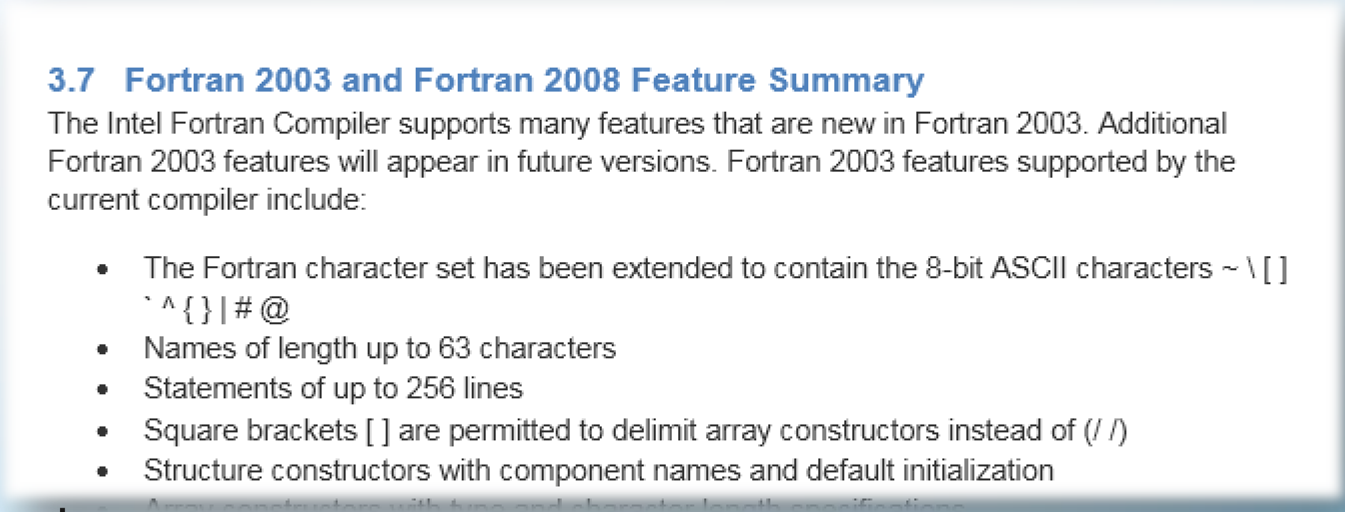
- New:
  - Default initialization of polymorphic variables
  - The keyword **MODULE** may be omitted from **MODULE PROCEDURE** in a generic interface block when referring to an external procedure

- The following features will be added later:
  - User defined derived type I/O
  - Parameterized derived types
  - Transformational intrinsics, such as **MERGE** and **SPREAD**, in initialization expressions

Intel® C++ and Fortran Compiler XE 13.x
Standards Conformance – Fortran 2008

- Most important features already there
- New:
  - **ATOMIC_DEFINE** and **ATOMIC_REF**

Refer to the Release Notes for the full list!



3.7 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

And also here:
http://software.intel.com/en-us/articles/intel-fortran-compiler-support-for-fortran-language-standards/

C++11 Support

New C++11 features

enabled by switch

`/Qstd=c++11` (Windows), `-std=c++11` (Linux, OS X)

(old *`std=c++0x`* switch still working)

- RVALUE references
- Variadic templates
- Extern templates
- Hexadecimal Floating Constants
- Atomic Types
- Right angle brackets
- Extended friend declarations
- Mixed string literal concatenations
- Support for long long
- Static assertions
- Universal character name literals
- Strongly-typed enums
- Lambda functions
- …

Pointer Checker

Intel® Parallel Studio XE

Intel® Cluster Studio XE

- Finds buffer overflows and dangling pointers before memory corruption occurs
- Powerful error reporting
- Integrates into standard debuggers (Microsoft, gdb, Intel)

Dangling pointer

```
{
   char *p, *q;
   p = malloc(10);
   q = p;
   free(p);
   *q = 0;
 }
```

Buffer Overflow

```
{
   char *my_chp = "abc";
   char *an_chp = (char *) malloc (strlen((char *)my_chp));
   memset (an_chp, '@', sizeof(my_chp));
}
```

CHKP: Bounds check error

```
Traceback:
./a.out(main+0x1b2) [0x402d7a] in file mems.c at line 13
```

Pointer Checker Highlights Programming Errors For More Secure Applications

Optimization Notice

Intel® Math Kernel Library 11.1
New Features

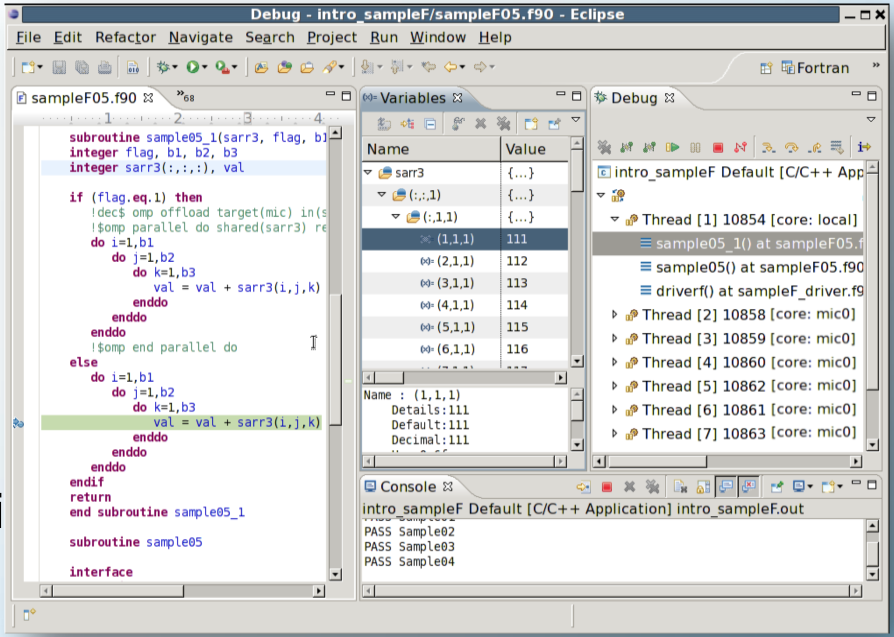
- Conditional Numerical Reproducibility (CNR) for unaligned memory
  - Balances performance with reproducible results by allowing greater flexibility in code branch choice and by ensuring algorithms are deterministic. More information: training site or the Intel® MKL User's Guide).
  - This release extends the feature to remove memory alignment requirements
  - Memory alignment is still recommended for best performance

Debugging on Linux* and OS X*

- Intel® Debugger (Intel® IDB) is deprecated:
  - No changes since Composer XE 2013 (13.0)
  - May be removed in a future release
- New debugger solution is based on GNU Project Debugger (GDB)* and includes additional improvements & features:
  - Improved Fortran support
  - GNU GDB 7.5 based
  - Python* 2.6 or greater required
- Linux only:
  - BTrace: Trace back branches taken before a crash
  - PDBX: Parallel Debug Extensions for data race detection
  - Pointer Checker: Assist in finding pointer issues
  - Intel® Transactional Synchronization Extensions

Intel® MIC Architecture Debugging (Linux*)

- Integration into Eclipse IDE*:
  - Supports C/C++ & Fortran
  - Support for offload extensions
  - Multiple coprocessor cards
- Command line debugging of nati possible with GDB


Debug - intro_sampleF/sampleF05.f90 - Eclipse
File Edit Refactor Navigate Search Project Run Window Help
Fortran
sampleF05.f90
Variables
Debug
Console

```fortran
      subroutine sample05_1(sarr3, flag, b1
      integer flag, b1, b2, b3
      integer sarr3(:,:,:), val

      if (flag.eq.1) then
         !dec$ omp offload target(mic) in(s
         !$omp parallel do shared(sarr3) re
         do i=1,b1
            do j=1,b2
               do k=1,b3
                  val = val + sarr3(i,j,k)
               enddo
            enddo
         enddo
         !$omp end parallel do
      else
         do i=1,b1
            do j=1,b2
               do k=1,b3
                  val = val + sarr3(i,j,k)
               enddo
            enddo
         enddo
      endif
      return
      end subroutine sample05_1

      subroutine sample05

      interface
```

Name	Value
sarr3	{...}
(:,:,1)	{...}
(:,1,1)	{...}
(1,1,1)	111
(2,1,1)	112
(3,1,1)	113
(4,1,1)	114
(5,1,1)	115
(6,1,1)	116

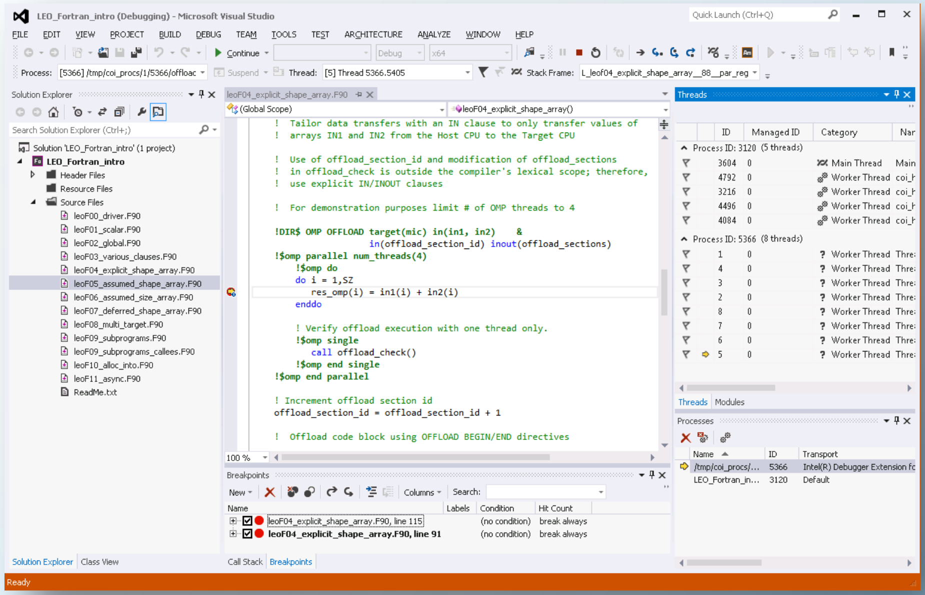
```
Name : (1,1,1)
    Details:111
    Default:111
    Decimal:111
```

```
intro_sampleF Default [C/C++ App
  Thread [1] 10854 [core: local]
    sample05_1() at sampleF05.f
    sample05() at sampleF05.f90
    driverf() at sampleF_driver.f9
  Thread [2] 10858 [core: mic0]
  Thread [3] 10859 [core: mic0]
  Thread [4] 10860 [core: mic0]
  Thread [5] 10862 [core: mic0]
  Thread [6] 10861 [core: mic0]
  Thread [7] 10863 [core: mic0]
```

```
intro_sampleF Default [C/C++ Application] intro_sampleF.out
PASS Sample02
PASS Sample03
PASS Sample04
```

Intel® MIC Architecture Debugging (Windows*)

- Integration into Microsoft Visual Studio* 2012:
  - Supports C/C++ & Fortran
  - Support for offload extensions
  - Multiple coprocessor cards


LEO_Fortran_intro (Debugging) - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Continue
Debug
x64
Suspend
Solution Explorer
Search Solution Explorer (Ctrl+;)
Threads
Processes
Call Stack
Breakpoints
Solution Explorer
Class View
Threads
Modules
New
Columns
Search:
100 %
Ready

Process: [5366] /tmp/coi_procs/1/5366/offloac
Thread: [5] Thread 5366.5405
Stack Frame: L_leof04_explicit_shape_array__88__par_reg

Solution 'LEO_Fortran_intro' (1 project)
- LEO_Fortran_intro
  - Header Files
  - Resource Files
  - Source Files
    - leoF00_driver.F90
    - leoF01_scalar.F90
    - leoF02_global.F90
    - leoF03_various_clauses.F90
    - leoF04_explicit_shape_array.F90
    - leoF05_assumed_shape_array.F90
    - leoF06_assumed_size_array.F90
    - leoF07_deferred_shape_array.F90
    - leoF08_multi_target.F90
    - leoF09_subprograms.F90
    - leoF09_subprograms_callees.F90
    - leoF10_alloc_into.F90
    - leoF11_async.F90
    - ReadMe.txt

leoF04_explicit_shape_array.F90 — (Global Scope) — leoF04_explicit_shape_array()

```fortran
!  Tailor data transfers with an IN clause to only transfer values of
!  arrays IN1 and IN2 from the Host CPU to the Target CPU

!  Use of offload_section_id and modification of offload_sections
!  in offload_check is outside the compiler's lexical scope; therefore,
!  use explicit IN/INOUT clauses

!  For demonstration purposes limit # of OMP threads to 4

!DIR$ OMP OFFLOAD target(mic) in(in1, in2)    &
                  in(offload_section_id) inout(offload_sections)
!$omp parallel num_threads(4)
   !$omp do
   do i = 1,SZ
      res_omp(i) = in1(i) + in2(i)
   enddo

   ! Verify offload execution with one thread only.
   !$omp single
      call offload_check()
   !$omp end single
!$omp end parallel

! Increment offload section id
offload_section_id = offload_section_id + 1

!  Offload code block using OFFLOAD BEGIN/END directives
```

Breakpoints

Name	Labels	Condition	Hit Count
leoF04_explicit_shape_array.F90, line 115		(no condition)	break always
leoF04_explicit_shape_array.F90, line 91		(no condition)	break always

Threads

	ID	Managed ID	Category	Nar
Process ID: 3120 (5 threads)				
	3604	0	Main Thread	Main
	4792	0	Worker Thread	coi_h
	3216	0	Worker Thread	coi_h
	4496	0	Worker Thread	coi_h
	4084	0	Worker Thread	coi_h
Process ID: 5366 (8 threads)				
	1	0	Worker Thread	Thre:
	4	0	Worker Thread	Thre:
	3	0	Worker Thread	Thre:
	2	0	Worker Thread	Thre:
	8	0	Worker Thread	Thre:
	7	0	Worker Thread	Thre:
	6	0	Worker Thread	Thre:
⇨	5	0	Worker Thread	Thre:

Processes

Name	ID	Transport
/tmp/coi_procs/...	5366	Intel(R) Debugger Extension fc
LEO_Fortran_in...	3120	Default

- Debugging native coprocessor applications also possible with remote attach

Optimization Notice

Intel® Xeon Phi™ Coprocessors Offload Extensions

Language Extensions for Offload New Features

- Three new clauses
  - `status` - affects handling of offload failures
  - `mandatory` - when coprocessor unavailable for offload and:
    - No `status` → code aborts
    - With `status` → user code directs action
  - `optional` - when coprocessor unavailable for offload and:
    - No `status` → code runs on CPU
    - With `status` → query value only – no user directed action available
- User-specified offload clauses override `-offload` compiler option settings

Language Extensions for Offload New Features

- New `–offload (/Qoffload)` option w/keywords:
  - `none` - Offload directives are ignored and cause warnings at compile-time
  - `mandatory` (default) - Offload directives processed When coprocessor is unavailable for offload → code aborts
  - `optional` - Any offload directives are processed When coprocessor is unavailable → code runs on CPU
- These options are overridden by user-specified offload clauses

Optimization Notice

Language Extensions for Offload New Features

- New environment variables
  - OFFLOAD_DEVICES: Restrict process to using only the coprocessors specified
  - OFFLOAD_INIT: Hint to the offload RTL when to initialize coprocessors
  - OFFLOAD_REPORT: Enables different levels of tracing and statistical information from offload.
  - OFFLOAD_ACTIVE_WAIT: Controls keeping host CPU busy during DMA transfers

OpenMP* 4.0 RC2

C++ Features
Intel® Composer XE 2013 SP1

- OpenMP* 4.0 SIMD Constructs
  - #pragma omp simd [clause] // transform loops into SIMD loops
    - *{for-loops}*
    - *clause is one of: safelen, linear, aligned, private, lastprivate, reduction, collapse(n)*
  - #pragma omp for simd [clause] // transform loops into SIMD loops and execute with a thread team
    - *{for-loops}*
    - *clause is one of: safelen, linear, aligned, private, lastprivate, reduction, collapse(n)*
  - #pragma omp declare simd [clause] // create a SIMD version of a function
    - *{function declaration or definition}*
    - *clause is one of: simdlen, linear, aligned, uniform, reduction, inbranch, notinbranch*
    - *Restriction: if the function is called in a SIMD loop, it cannot result in the execution of an OpenMP construct (but the function itself can be called from an OpenMP parallel region)*

C++ Features

Intel® Composer XE 2013 SP1

- OpenMP* 4.0 TARGET Constructs
  - #pragma omp target data [clause] // create a device data environment for the extent of the target region
    - *{structured block}*
    - *clause is one of: device(integer-expression), map(list), if(scalar-expression)*
  - #pragma omp target [clause] // create a device data environment and execute the construct on the device
    - *{structured block}*
    - *clause is one of: device(integer-expression), map(list), if(scalar-expression)*
  - #pragma omp declare target // map functions and variables to a device; create device-specific versions of functions
    - *{variable and function declarations}*
  - #pragma omp target update *motion-clause* [clause] // update variables between the device data environment and the host data environment
    - *motion-clause is one of: to(list), from(list)*
    - *clause is one of: device(integer-expression), if(scalar-expression)*

Fortran Features
Intel® Composer XE 2013 SP1

- OpenMP* 4.0 SIMD Constructs
  - !$omp simd [clause] ! transform loops into SIMD loops
    - *{do-loops}*
    - *[!$omp end simd] ! Optional end directive*
    - *clause is one of: safelen, linear, aligned, private, lastprivate, reduction, collapse(n)*
  - !$omp do simd [clause] ! transform loops into SIMD loops and execute with a thread team
    - *{do-loops}*
    - *[!$omp end do simd] ! Optional end directive*
    - *clause is one of: safelen, linear, aligned, private, lastprivate, reduction, collapse(n)*
  - !$omp declare simd [clause] ! create a SIMD version of a procedure
    - *{ function or subroutine definition }*
    - *clause is one of: simdlen, linear, aligned, uniform, reduction, inbranch, notinbranch*
    - *Restriction: if the procedure is called in a SIMD loop, it cannot result in the execution of an OpenMP construct (but the procedure itself can be called from an OpenMP parallel region)*

Fortran Features
Intel® Composer XE 2013 SP1

- OpenMP* 4.0 TARGET Constructs
  - !$omp target data [clause] ! create a device data environment for the extent of the target region
    - *{structured block}*
    - *clause is one of: device(integer-expression), map(list), if(scalar-expression)*
  - !$omp target [clause] ! create a device data environment and execute the construct on the device
    - *{structured block}*
    - *clause is one of: device(integer-expression), map(list), if(scalar-expression)*
  - !$omp declare target (list)// map functions and variables to a device; create device-specific versions of functions
    - *list is a comma-separated list of variables, procedures, and named common blocks*
  - !$omp target update *motion-clause* [clause] // update variables between the device data environment and the host data environment
    - *motion-clause is one of: to(list), from(list)*
    - *clause is one of: device(integer-expression), if(scalar-expression)*

Coarrays with Intel® Xeon Phi™ Coprocessors

Coarrays with Intel® Xeon Phi™ Coprocessors (cont.)

- Restrictions on coarray w/offload regions
  - coindexing is ***not*** allowed
    - All accesses to coarrays within an offload region must be to the local copy of the coarray
  - SYNC ALL, SYNC MEMORY, SYNC IMAGES, or LOCK/UNLOCK use is***not*** allowed in an offload region
  - Coarrays must ***not*** be allocated or deallocated within an offload region

Coarrays with Intel® Xeon Phi™ Coprocessors (cont.)

- *Heterogeneous* coarray application
  - Some images run on the Intel® 64 host system and some run on an Intel® Xeon Phi™ coprocessor
- Requirements Overview (*heterogeneous*)
  - Refer to the RNs for specifics
  - Compile with:
    - `-coarray=coprocessor`
    - `-coarray-config-file=<file>` to specify coarray (CAF) configuration file

Coarrays with Intel® Xeon Phi™ Coprocessors (cont.)

- Requirements Overview (*heterogeneous*) (cont.)
  - Creates host & Intel® MIC architecture native executable – must copy native exe to coprocessor
  - Coarray (CAF) configuration file - provides critical MPI config information
    - Path to native executable on the coprocessor
    - Number of images to run on the host and coprocessor
  - Run with:
    - Environment variable I_MPI_MIC set to ENABLE

Coarrays with Intel® Xeon Phi™ Coprocessors (cont.)

- Native coarray application
  - Runs exclusively on the coprocessor
  - Compile with `-coarray` and `-mmic`
  - Coarray (CAF) configuration file is not required
  - Compilation creates single Intel® MIC architecture native executable image
  - Copy image along with referenced library shared objects to the card - execute as normal native executable

Fortran 2003 Support: User Defined Derived Type Input and Output

User Defined I/O

Intel® Composer XE 2013 SP1

- Fortran 2003 Standard feature
- Custom subroutines can be used to handle input or output for objects of a derived type
- Motivation:
  - Allows user control of the way input and output is handled for derived type variables
  - The default I/O handlers can not be used for some derived types
    - Component variables printed in order of declaration
    - No pointers or allocatables

User Defined I/O

Intel® Composer XE 2013 SP1

- User defined I/O subroutines may be specified with an explicit interface or as a type bound generic procedure.
- Supports Formatted, Unformatted, Namelist, and List Directed I/O
  - read(formatted), write(formatted)
  - read(unformatted), write(unformatted)
- For Formatted I/O, format specified with dt edit descriptor
  - write (6, fmt='(dt"my_type"(3,4))') foo
- Formatted user defined I/O subroutines may only use formatted I/O statements, and unformatted user defined I/O subroutines may only use unformatted I/O statements

User Defined I/O
Intel® Composer XE 2013 SP1

- All User Defined I/O subroutines must handle :
  - the derived type variable,
  - Unit – the I/O unit number
  - iostat
  - iomsg
- Formatted I/O routines also handle
  - Iotype - “Listdirected”, “Namelist”, or “DT”
  - v_list – array containing values passed through the DT edit descriptor, defaults to a zero sized array
  - If I/O procedure could be called recursively, the routine must be labeled as recursive

User Defined I/O Example

```
module ball_mod
    type ball
        integer, pointer :: x, y
        double precision :: r
        integer          :: id
    contains
      procedure :: write_ball
      generic   :: write(formatted) => write_ball
    end type
    interface read(formatted)
        module procedure read_ball
    end interface=
Contains
  ...
  ! subroutines write_ball & read_ball
  ! are on the following slide
end module
```

Type can not be output with default I/O

User Defined I/O Example

```
 subroutine write_ball (this, unit, iotype, vlist,
iostat, iomsg)
      class(ball), intent(in)               :: this
      integer, intent(in)                   :: unit
      character (len=*), intent(in)         :: iotype
      integer, intent(in)                   :: vlist(:)
      integer, intent(out)                  :: iostat
      character (len=*), intent(inout)      :: iomsg
      !Local
      character (len=18) :: pfmt
      write (pfmt, '(a,i2,a,i2,a)' ) &
         '(a,i', vlist(1), ',i', vlist(2), ',a,f8.3)'
      write (unit, fmt=pfmt, iostat=iostat,&
            iomsg=iomsg) "ball position = &
            ", this%x, this%y, " area =", &
            3.14 * this%r**2
   end subroutine
```

User Defined I/O Example

```
 subroutine read_ball (this, unit, iotype, vlist,
iostat, iomsg)
      class(ball), intent(inout)              :: this
      integer, intent(in)                     :: unit
      character (len=*), intent(in)           :: iotype
      integer, intent(in)                     :: vlist(:)
      integer, intent(out)                    :: iostat
      character (len=*), intent(inout)        :: iomsg

      if (iotype == "DTradius") then
         read (unit, fmt="(f8.4)",iostat=iostat, &
               iomsg=iomsg) this%r
      else if (iotype == "DTid") then
         read (unit, fmt="(i6)",iostat=iostat, &
              iomsg=iomsg) this%id
      endif
  end subroutine
```

User Defined I/O Example

```fortran
program main
   use ball_mod
   integer, target :: x = 4, y = 5
   type(ball) :: ball_foo
   ball_foo%x => x
   ball_foo%y => y
   write (*, '(a)') "ball radius="
   read (*, '(dt"radius")') ball_foo
   write (*, '(dt(2,2))') ball_foo
end program main
```

```
:~> ifort ball.f90
:~> ./a.out
ball radius=
2.d0
ball position =   4 5 area =   12.560
```