Intel® Parallel Studio XE 2011
Windows and Linux

Value Proposition

What

Leading Software Development Tools for
Optimized Application Performance, Scalable Parallelism, and
Confidence

Why

- **Performance** - Intel tools are key to utilizing processor performance
- **Scale forward** - Your application investment extends to tomorrow’s platforms
- **Confidence** - Trusted software tools you can count on for best results

How

- **Leading** C++ and Fortran performance optimizing compilers libraries, performance libraries, and analysis tools
- **Advanced** parallel programming models to develop code for Intel® Xeon® Processors today, easily extends to Intel® MIC architecture
- **Trusted** software correctness tools for application quality and ensured confidence
## Intel® Parallel Studio XE 2011
Powerful Tools Provide Comprehensive Coverage

<table>
<thead>
<tr>
<th>Phase</th>
<th>Productivity Tool</th>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Intel® Parallel Advisor for C++ Windows</td>
<td>Threading design assistant</td>
<td>• Simplifies, demystifies, and speeds parallel application design</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Available for Intel® Parallel Studio XE Windows and Intel® C++ Studio Windows (ESD)</td>
</tr>
<tr>
<td>Build &amp; Debug</td>
<td>Intel® Composer XE</td>
<td>C/C++ and Fortran compilers and performance libraries</td>
<td>• Enabling solution to achieve the application performance and scalability benefits of multicore and forward scale to manycore</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Integrated Performance Primitives</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Math Kernel Library</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Threading Building Blocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Cilk™ Plus</td>
<td></td>
</tr>
<tr>
<td>Verify</td>
<td>Intel® Inspector XE</td>
<td>Memory &amp; threading dynamic analysis for code quality</td>
<td>• Increased productivity, code quality, and lowers cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Static Security Analysis for code quality (Studio products only)</td>
<td>• Finds memory, threading, and security defects before they happen</td>
</tr>
<tr>
<td>Tune</td>
<td>Intel® VTune™ Amplifier XE</td>
<td>Performance Profiler for optimizing application performance and scalability</td>
<td>• Remove guesswork, saves time, makes it easier to find performance and scalability bottlenecks</td>
</tr>
</tbody>
</table>

**Optimized Performance, Scalable Parallelism, Confidence**
Intel® Composer XE - Agenda
Compilers, Performance Libraries, Debugging Tools

- Product Overview 15 min
- Optimization Essentials with the Intel Compiler 45 min
  - Optimization basics
  - Compiling for AVX (Sandybridge)
  - Floating point model (FP accuracy, consistency control)
- Forcing/Controlling Vectorization with pragma SIMD 30 min
- Intel Programming Models – 30 min
  - Coarray Fortran
  - Intel® Cilk Plus
- Intel® Math Kernel Library 30 min
  - MKL overview/summary
  - Latest Features in 10.3
  - Current work

Supported Environments
Microsoft Visual Studio* 2005, 2008, 2010 Eclipse CDT and Mac XCode*
Intel Composer XE - Overview
Compilers, Performance Libraries, Debugging Tools

Intel® C++ Composer XE 2011
- Intel® C++ Compiler XE 12.1
- Intel® Threading Building Blocks
- Intel® Cilk™ Plus
- Intel® Math Kernel Library
- Intel® Integrated Performance Primitives

Intel® Fortran Composer XE 2011
- Intel® Fortran Compiler XE 12.1
- Intel® Math Kernel Library
- Intel® Integrated Performance Primitives

**Leading Performance Optimizing Compilers**
- Intel C++ and Fortran Compilers
- Intel® Integrated Performance Primitives, Intel® Math Kernel Library libraries
- Profile-Guided Optimization (PGO)
- Interprocedural Optimization (IPO)
- Guided-auto parallelism (GAP)
- High-Performance Parallel Optimizer (HPO)
- SIMD Pragma
- C++ Array Notations

**Standard Support**
- OpenMP*
- C++0x
- Support for key parts of the latest Fortran and C++ standards, Visual Studio* 2010 Shell for Visual Fortran*

**Compatibility – Mix and Match**
- Binary and source compatible with Microsoft* Visual Studio* C++ and GCC*

Windows*, Linux*, Mac OS*, 32-bit & 64-bit multicore processor support
Intel® AVX support

Supported Environments
Microsoft Visual Studio* 2005, 2008, 2010 Eclipse CDT and Mac XCode*
Updated Compilers and Libraries Produce Industry Leading Performance

- Intel v12.1 compilers improve performance compared with:
  - Competitive compilers
  - Previous version Intel compilers

<table>
<thead>
<tr>
<th></th>
<th>Intel v12.1 Compiler on Windows* vs. nearest competitor</th>
<th>Intel v12.1 Compiler on Linux* vs. nearest competitor</th>
<th>Intel v12.1 Compiler on Windows vs. v12.0</th>
<th>Intel v12.1 Compiler on Linux vs. v12.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ Integer(^1)</td>
<td>47% faster</td>
<td>12% faster</td>
<td>11% faster</td>
<td>6% faster</td>
</tr>
<tr>
<td>C/C++ Floating Point(^1)</td>
<td>21% faster</td>
<td>9% faster</td>
<td>3% faster</td>
<td>1% faster</td>
</tr>
<tr>
<td>Fortran(^2)</td>
<td>24% faster</td>
<td>17% faster</td>
<td>22% faster</td>
<td>27% faster</td>
</tr>
</tbody>
</table>

Notes:
\(^1\) C/C++ performance measured using SPECint\(_\text{base2006}\) estimated RATE benchmark running on a 64 bit operating system

\(^2\) Fortran performance measured using Polyhedron\(^*\) benchmark running on a 64 bit operating system. In this performance measurement, “faster” refers to percent reduction in time-to-completion.
What’s New: Intel® Composer XE 12.1
More Standards and Compatibility Support

**Expanded Standards Support**
- First with IA conformance of both binary and decimal floating-point specs. Enables greater floating point consistency between processors
  - IEEE Standard 754-2008 for Floating-Point Arithmetic
  - Technical Report ISO/IEC TR 24732, Extension for the programming language C to support decimal floating-point arithmetic
  - Floating point number computations yield the same result whether hardware, software, or a combination of the two
- Key portions of the latest C++ and Fortran standards
  - Enhanced GCC support of C++ 0X standards Variadic templates, enables templates to take variable number of arguments
  - Lambda support
  - Fortran coarray support for distributed-memory systems
- OpenMP* 3.1 support

**Enhanced Compatibility Support**
- Microsoft Visual Studio* 2010 Shell for Visual Fortran*
A Family of Parallel Programming Models

Developer Choice

- **Intel® Cilk™ Plus**: C/C++ language extensions to simplify parallelism
  - Open sourced
  - Also an Intel product

- **Intel® Threading Building Blocks**: Widely used C++ template library for parallelism
  - Open sourced
  - Also an Intel product

- **Domain-Specific Libraries**:
  - Intel® Integrated Performance Primitives
  - Intel® Math Kernel Library

- **Established Standards**: Message Passing Interface (MPI)
  - OpenMP®
  - Coarray Fortran
  - OpenCL®

- **Research and Development**: Intel® Concurrent Collections
  - Offload Extensions
  - Intel® Array Building Blocks
  - Intel® SPMD Parallel Compiler

**Choice of high-performance parallel programming models**

- Libraries for pre-optimized and *parallelized functionality*

- Intel® Cilk™ Plus and Intel® Threading Building Blocks supports composable parallelization of a wide variety of applications.

- OpenCL® addresses the needs of customers in specific segments, and provides developers an additional choice to maximize their app performance.

- MPI supports distributed computation, combines with other models on nodes
“By just utilizing standard programming on both Intel® Xeon processor and Intel® MIC architecture based platforms, the performance met multi-threading scalability expectations and we observed near-theoretical linear performance scaling with the number of threads.” – Hongsuk Yi, Heterogeneous Computing Team Leader, KISTI Supercomputing Center

“SGI understands the significance of inter-processor communications, power, density and usability when architecting for exascale. Intel has made the leap towards exaflop computing with the introduction of Intel® Many Integrated Core (MIC) architecture. Future Intel® MIC products will satisfy all four of these priorities, especially with their expected ten times increase in compute density coupled with their familiar X86 programming environment.” – Dr. Eng Lim Goh, SGI CTO
Agenda

Optimization Essentials with the Intel Compiler 45 min

- Optimization basics
- Compiling for AVX (Sandybridge)
- Floating point model (FP accuracy, consistency control)
Optimization Basics

Typical Compiler Optimizations

Vectorization – the ‘Make or Break’ for IA performance

Other supporting optimizations that aide Vectorization

Auto-Parallelization

Tools to help – Guided Auto-Parallelization
Typical Compiler Optimizations
## Common Optimization Switches

<table>
<thead>
<tr>
<th>Optimization Type</th>
<th>Windows*</th>
<th>Linux* Mac OS*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable optimization</td>
<td>/Od</td>
<td>-O0</td>
</tr>
<tr>
<td>Optimize for speed (no code size increase)</td>
<td>/O1</td>
<td>-O1</td>
</tr>
<tr>
<td>Optimize for speed (default)</td>
<td>/O2</td>
<td>-O2</td>
</tr>
<tr>
<td>High-level optimizer, including prefetch, unroll</td>
<td>/O3</td>
<td>-O3</td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td>/Zi</td>
<td>-g</td>
</tr>
<tr>
<td>Inter-procedural optimization</td>
<td>/Qipo</td>
<td>-ipo</td>
</tr>
<tr>
<td>Profile guided optimization (multi-step build)</td>
<td>/Qprof-gen /Qprof-use</td>
<td>-prof-gen /prof-use</td>
</tr>
<tr>
<td>Optimize for speed across the entire program **warning: -fast def’n changes over time</td>
<td>/fast (same as: /O3 /Qipo /Qprec-div- /QxHost)</td>
<td>-fast (same as: -ipo -O3 -no-prec-div -static -xHost)</td>
</tr>
<tr>
<td>OpenMP 3.0 support</td>
<td>/Qopenmp</td>
<td>-openmp</td>
</tr>
<tr>
<td>Automatic parallelization</td>
<td>/Qparallel</td>
<td>-parallel</td>
</tr>
</tbody>
</table>
High-Level Optimizer (HLO)

Compiler switches:
- `-O2`, `-O3` (Linux*)

/`O2`, `/`O3` (Windows*),

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 references with register references
Vectorization – the ‘Make or Break’ for IA Performance
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

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

Diagram:
- Vector addition: `a[i]` and `b[i]` yield `a[i]+b[i]`
Vectorization is Achieved through SIMD Instructions & Hardware

**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

**Intel® AVX**
- Vector size: 256bit
- Data types: 32 and 64 bit floats
- VL: 4, 8, 16
- Sample: Xi, Yi 32 bit int or float
- First introduced in 2011
What Kinds of Applications can use Vectorization?

- 3D Modeling & Visualization
- Bioinformatics
- Broadcast & Film
- Database & Business Intelligence
- Digital Content Creation
- Defense & Security
- Engineering Design
- Energy
- Financial Analytics
- Game Development
- GIS & Satellite Imagery
- Medical Imaging & Analysis
- Science & Research
- Signal Processing
- Telecommunications
## Comparison of Ways Applications can Take Advantage of Vectorization

<table>
<thead>
<tr>
<th>Method</th>
<th>Effort Required</th>
<th>Code Maintainability</th>
<th>Performance Potential</th>
<th>Scale Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly/Intrinsics</td>
<td>Most</td>
<td>Least</td>
<td>Best</td>
<td>No</td>
</tr>
<tr>
<td>Existing libraries such as Intel® IPP, Intel® MKL</td>
<td>Least</td>
<td>Most</td>
<td>Best</td>
<td>Yes</td>
</tr>
<tr>
<td>Intel Compiler Auto-Vectorization</td>
<td>Least</td>
<td>Most</td>
<td>Good</td>
<td>Yes</td>
</tr>
<tr>
<td>High-level Constructs such as Intel® Cilk™ Plus, Fortran DO CONCURRENT</td>
<td>Moderate</td>
<td>Most</td>
<td>Best</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Compiler Based Vectorization

### Extension Specification

<table>
<thead>
<tr>
<th>Feature</th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Streaming SIMD Extensions 2 (Intel® SSE2) as available in initial Pentium® 4 or compatible non-Intel processors</td>
<td>sse2</td>
</tr>
<tr>
<td>Intel® Streaming SIMD Extensions 3 (Intel® SSE3) as available in Pentium® 4 or compatible non-Intel processors</td>
<td>sse3</td>
</tr>
<tr>
<td>Supplemental Streaming SIMD Extensions 3 (SSSE3) as available in Intel® Core™ 2 Duo processors</td>
<td>ssse3</td>
</tr>
<tr>
<td>Intel® SSE4.1 as first introduced in Intel® 45nm Hi-K next generation Intel Core™ micro-architecture</td>
<td>sse4.1</td>
</tr>
<tr>
<td>Intel® SSE4.2 Accelerated String and Text Processing instructions supported first by Intel® Core™ i7 processors</td>
<td>sse4.2</td>
</tr>
<tr>
<td>Extensions offered by Intel® ATOM™ processor : Intel SSSE3 (!! and MOVBE instruction)</td>
<td>sse3_atom</td>
</tr>
<tr>
<td>Intel® Advanced Vector Extensions (Intel® AVX) as available in 2nd generation Intel Core processor family</td>
<td>avx</td>
</tr>
</tbody>
</table>
Basic Vectorization – Switches [1]

{L&M} -x<extension>     {W}: /Qx<extension>
Targeting Intel® processors - specific optimizations for Intel® processors
Compiler will try to make use of all instruction set extensions up to and
including <extension>; for Intel® processors only!
Processor-check added to main-program
Application will not start (will display message), in case feature is not available

{L&M}: -m<extension>    {W}: /arch:<extension>
No Intel processor check
Does not perform Intel-specific optimizations
Application is optimized for and will run on both Intel and non-Intel processors
Missing check can cause application to fail in case extension not available

{L&M}: -ax<extension>   {W}: /Qax<extension>
Dual-code paths – a ‘baseline’ and ‘optimized, processor-specific’ path
Optimized code path for Intel® processors defined by <extension>
Baseline code path defaults to -msse2 (Windows: /arch:sse2)
• The baseline code path can be modified by -m or -x (/Qx or /arch) switches
Basic Vectorization – Switches [2]

The default is `-msse2` (Windows: `/arch:sse2`)

Activated implicitly for `-O2` or higher

Implies the need for a target processor with Intel® SSE2

For 32 bit compilation, `-mia32` (Windows `/arch:ia32`) can be used in case target processor misses SSE2 (like Intel® Pentium™ 3)

Special switch `-xHost` (Windows: `/QxHost`)

Compiler checks host processor and makes use of ‘latest’ instruction set extension available

Avoid for builds being executed on multiple, unknown platforms

Multiple extensions can be used in combinations like `-ax<ext1>,<ext2>` (Windows: `/Qax<ext1>,<ext2>`)

Can result in more than 2 code paths

For 32bit compilation, use `-mia32` ( `/arch:ia32`) in case baseline code path should support too very early processors not supporting SSE2 (e.g. Intel® Pentium™ 3);
Vectorization – More Switches and Directives

Disable vectorization

Globally via switch: {L&M}: \(-\text{no-vec}\)  {W}: /Qvec-

For a single loop: directive \texttt{novector}

- Disabling vectorization here means not using packed SSE/AVX instructions. The compiler still might make use of the corresponding instruction set extensions.

Enforcing vectorization for a loop - overwriting the compiler heuristics:

\#pragma vector always

- will enforce vectorization even if the compiler thinks it is not profitable to do so (e.g due to non-unit strides or alignment issues)
- Will not enforce vectorization if the compiler fails to recognize this as a semantically correct transformation
- Using directive \#pragma vector always assert will print error message in case the loop cannot be vectorized and will abort compilation
Vectorization Switches – Some Notes

Former vectorization switches like –xW, /QxT, /QaxP etc are considered ‘deprecated’ and will not be supported anymore in the future

• See appendix or compiler documentation for mapping between old and new names

It is not possible anymore to generate vector code exclusively for the initial SSE (32 bit FP) instruction set (introduced by Intel® Pentium™ 3 processor)

The instruction set extension name for Intel® Atom™ processors (SSE3_ATOM) is misleading: Since the architecture supports up to SSSE3, e.g. switch –xsse3_atom will make use of SSSE3 too

• In case the code should be optimized for Intel® Atom processors but should run too on all processors supporting up to SSSE3, add option –minstruction=nomovbe (Windows: /Qinstruction:nomovbe) to avoid the use of the Atom-specific instruction MOVBE
Compiler Reports – Vectorization Report

Compiler switch:
/\texttt{Qvec-report}\langle n\rangle \quad (\text{Windows})
-vec-report\langle n\rangle \quad (\text{Linux})

Set diagnostic level dumped to stdout

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

Note:
For Linux, \texttt{-opt_report_phase hpo} provides additional diagnostic information for vectorization
Other Optimizations that Support or Complement Vectorization

Vectorization relies on

- NO Dependencies in data between loop iterations
  - or dependencies that can be removed by simple methods
- Regular, predictable data patterns for all operands
  - No pointer chasing, indirect accesses
- Vector lengths that are large enough AND
- Data is aligned on natural boundaries (16, 32, or 64 bytes)
  - cache line matching boundary, can use cache as staging
- Prefetch directives – load pipelining
- Streaming stores – store optimization

COST MODEL – is it worth it to vectorize???

Advanced optimizations help with some of these
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

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

<table>
<thead>
<tr>
<th></th>
<th>Windows*</th>
<th>Linux*</th>
<th>Mac OS*</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Qip</td>
<td>-ip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qipo</td>
<td>-ipo</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interprocedural Optimizations (IPO)

Usage: Two-Step Process

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

| Linking         | Linux*                  | icc -ipo main.o func1.o
|                 | Windows*                | icl /Qipo main.o func1.o

Pass 1

Pass 2

mock object

executable
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();
```

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
PGO Usage: Three Step Process

Step 1
Compile + link to add instrumentation
```
icc -prof_gen prog.c
```
Instrumented executable: foo.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: foo.exe
Parallelization

Obviously, we now have multi-core and many-core

How to utilize those cores?
• Treat as another system: process level parallelism
• 1 MPI process per core
  ‒ disadvantage: larger memory footprint, unneeded redundancy

• Threading: map threads to cores
  ‒ Smaller memory footprint, sharing of data possible
  ‒ Auto-parallel – let the compiler do it
  ‒ OpenMP, Pthreads, Win threads
  ‒ Higher level language abstractions and constructs
Auto-Parallelization

Compiler automatically translates portions of serial code into equivalent multithreaded code with using these options:

-parallel /Qparallel

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 option:

-par-report 0|1|2|3
/Qpar-report: 0|1|2|3

0 is report disabled, 3 maximum diagnostics level
Compiler Reports – Optimization Report

Compiler option:

Control the level of detail in the report:
- `opt-report[0|1|2|3]` (Linux, Mac OS X)
  /Qopt-report[0|1|2|3] (Windows)

- `opt-report-phase[=phase]` (Linux)
  /Qopt-report-phase[=phase] (Windows)

*phase* can be:

**ipo** - Interprocedural Optimization

**ilo** – Intermediate Language Scalar Optimization

**hpo** – High Performance Optimization (cache blocking, data access)

**hlo** – High-level Optimization (loop transforms, etc.)

**all** – All optimizations (default if opt-report-phase not used)
Optimization Report Example

```
icc -O3 -opt-report-phase=hlo -opt-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
...
```

–vec-report2 (/Qvec-report2) to get a vectorization report
GAP – Guided Auto Parallelization
A New Tool in
Intel Parallel Composer XE 2011
GAP – Guided Automatic Parallelization

Key design ideas:

- Use compiler to help detect what is blocking optimizations – in particular vectorization, parallelization and data transformations – gives advice on how to change code, add directives, add compiler options
  - Extend diagnostic message for failed vectorization and parallelization by specific hints to fix problem
- Not a separate tool, part of the compiler

It is not:

- Automatic vectorizer or parallelizer
  - in fact, no code is generated to accelerate analysis
- GAP does not ask the programmer to change algorithms, transformation ordering or internal heuristics of compiler
  - It is restricted to changes applied to the program to be compiled
Workflow with Compiler as a Tool

Application Source C/C++/Fortran → Compiler → Application Binary + Opt Reports → Performance Tools → Identify hotspots, problems

Application Source + Hotspots → Compiler in advice-mode → Advice messages

Modified Application Source → Compiler (extra options) → Improved Application Binary

Simplifies programmer effort in application tuning
**GAP – How it Works (linux)**

Selection of most Relevant Switches

Multiple compiler switches to activate and fine-tune guidance analysis

Activate messages individually for vectorization, parallelization, data transformations or all three

- `guide [=level]`
- `guide-vec [=level]`
- `guide-par [=level]`
- `guide-data-trans [=level]`

Optional argument `level = 1,2,3,4` controls extend of analysis

Control the source code part for which analysis is done

- `guide-opts = <arg>`

Samples:

- `guide-opts = "bar.f90, 'module_1::func_solve` “`

Control where the message are going

- `guide-file = <file_name>`
1 module scalar_dep
2   integer, parameter :: size=100000
3   real(8), dimension(size) :: a
4   real(8)                  :: b
5
6 contains
7 subroutine test_scalar_dep(n)
8   integer n
9   integer i
10  integer t
11
12  do i = 1, n
13   if (a(i) >= 0) then
14     t = i
15   end if
16
17   if (a(i) > 0) then
18     a(i) = t * (1 / (a(i) * a(i)))
19   end if
20  end do
21 end subroutine test_scalar_dep
22 end module scalar_dep

$ ifort -parallel -O3 --xhost --vec-report -c scalar_dep.f90

scalar_dep.f90(12): (col. 9) remark: loop was not vectorized: existence of vector dependence.

scalar_dep.f90(19): (col. 17) remark: vector dependence: assumed ANTI dependence between t line 19 and t line 15.

scalar_dep.f90(15): (col. 17) remark: vector dependence: assumed FLOW dependence between t line 15 and t line 19.
GAP Sample - Vector

scalar_dep.f90(12): remark #30515: (VECT) Assign a value to the variable(s) "t" at the beginning of the body of the loop in line 12. This will allow the loop to be vectorized. [VERIFY] Make sure that, in the original program, the variable(s) "t" read in any iteration of the loop has been defined earlier in the same iteration.

Number of advice-messages emitted for this compilation session: 1.

END OF GAP REPORT LOG
module scalar_dep
  integer, parameter :: size=100000
  real(8), dimension(size) :: a
  real(8)                  :: b
contains
  subroutine test_scalar_dep(n)
    integer n
    integer i
    integer t
    do i = 1, n
      if (a(i) >= 0) then
        t = i
      end if
    end do
    if (a(i) > 0) then
      a(i) = t * (1 / (a(i) * a(i)))
    end if
  end subroutine
end module scalar_dep

$ ifort -parallel -O3 -par-report3 -c scalar_dep.f90
scalar_dep.f90(12): (col. 9) remark: loop was not parallelized: existence of parallel dependence.
scalar_dep.f90(19): (col. 17) remark: parallel dependence: assumed ANTI dependence between t line 19 and t line 15.
scalar_dep.f90(15): (col. 17) remark: parallel dependence: assumed FLOW dependence between t line 15 and t line 19.
**GAP Sample Messages - PARALLEL**

scalar_dep.f90(12): remark #30523: (PAR) Assign a value to the variable(s) "t" at the beginning of the body of the loop in line 12. This will allow the loop to be parallelized. [VERIFY] Make sure that, in the original program, the variable(s) "t" read in any iteration of the loop has been defined earlier in the same iteration.

[ALTERNATIVE] Another way is to use "!dir$ parallel private(t)" to parallelize the loop. [VERIFY] The same conditions described previously must hold.

scalar_dep.f90(12): remark #30525: (PAR) Insert a "!dir$ loop count min(64)" statement right before the loop at line 12 to parallelize the loop. [VERIFY] Make sure that the loop has a minimum of 64 iterations.

Number of advice-messages emitted for this compilation session: 2.

END OF GAP REPORT LOG

GAP REPORT LOG OPENED ON Thu May 20 15:22:14 2010
module scalar_dep
2    integer, parameter :: SIZE=100000
3    real(8), dimension(size) :: a
4    real(8) :: b
5
6    contains
7    subroutine test_scalar_dep(n)
8        integer n
9        integer i
10       integer t
11
12       !dir$ loop count min(64)
13       do i = 1, n
14
15          t = i
16
17          if (a(i) > 0) then
18              a(i) = t * (1 / (a(i) * a(i)))
19          end if
20       end do
21    end subroutine test_scalar_dep
22 end module scalar_dep
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++;
}

...
Another Vectorization Example

```c
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 hoist 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
User Mandated Vectorization
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
Requirements for Vectorization

Must be an inner loop.

Straight-line code (masked assignments OK)

Avoid:
- Function/subroutine calls (unless inlined)
- Non-mathematical operators (sqrt, sin, exp,... OK)
- Data-dependent loop exit conditions
  - Iteration count must be known at entry to loop
- Loop carried data dependencies
  - Reduction loops are OK
- Non-contiguous data (indirect addressing; non-unit stride)
  - inefficient

Directives/pragmas can help:
- #pragma ivdep ...... ignore potential dependencies
- #pragma vector always ignore efficiency heuristics
- aligned assume data aligned
- Compiler can generate runtime alignment and dependency tests for simple loops (but less efficient)

See http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/
## Vectorizable math functions

<table>
<thead>
<tr>
<th></th>
<th>acos</th>
<th>ceil</th>
<th>fabs</th>
<th>round</th>
</tr>
</thead>
<tbody>
<tr>
<td>acosh</td>
<td>cos</td>
<td>floor</td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>asin</td>
<td>cosh</td>
<td>fmax</td>
<td>sinh</td>
<td></td>
</tr>
<tr>
<td>asinh</td>
<td>erf</td>
<td>fmin</td>
<td>sqrt</td>
<td></td>
</tr>
<tr>
<td>atan</td>
<td>erfc</td>
<td>log</td>
<td>tan</td>
<td></td>
</tr>
<tr>
<td>atan2</td>
<td>erfinv</td>
<td>log10</td>
<td>tanh</td>
<td></td>
</tr>
<tr>
<td>atanh</td>
<td>exp</td>
<td>log2</td>
<td>trunc</td>
<td></td>
</tr>
<tr>
<td>cbrt</td>
<td>exp2</td>
<td>pow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Also float versions, such as sinf()

Uses short vector math library, libsvml

New entry points for AVX, e.g.
- __svml_pow4
- __svml_powf8
- cf for SSE
- __svml_pow2
- __svml_powf4

Many routines in the libsvml math library are more highly optimized for Intel microprocessors than for non-Intel microprocessors.

Copyright© 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.*
Problems with Pointers

Hard for compiler to know whether arrays or pointers might be aliased (point to the same memory location)

- Aliases may hide dependencies that make vectorization unsafe

In simple cases, compiler may generate vectorized and unvectorized loop versions, and test for aliasing at runtime

Otherwise, compiler may need help:

- `-fargument-noalias` (`-Qalias-args-`) & similar switches
- “restrict” keyword with `-restrict` or `-std=c99` (`-Qrestrict` or `#pragma ivdep` `#Qstd=c99`)
- or by inlining

Less of a problem for Fortran due to stricter language rules

```c
void saxpy (float *x, float *y, float *restrict z, float *a, int n) {
    int i;
    #pragma ivdep
    for (i=0; i<n; i++) z[i] = *a*x[i] + y[i];
}
```
Guidelines for Writing Vectorizable Code

Prefer simple “for” loops

Write straight line code. Avoid:
• most function calls
• branches that can’t be treated as masked assignments.

Avoid dependencies between loop iterations
• Or at least, avoid read-after-write dependencies

Prefer array notation to the use of pointers
• Without help, the compiler often cannot tell whether it is safe to vectorize code containing pointers.
• Try to use the loop index directly in array subscripts, instead of incrementing a separate counter for use as an array address.

Use efficient memory accesses
• Favor inner loops with unit stride
• Minimize indirect addressing
• Align your data where possible
  - to 32 byte boundaries (for AVX instructions)
  - else to 16 bytes, or at least to “natural” alignment
How to Align your Data

_declsvec(align(n, [offset]))
Instructs the compiler to create the variable so that it is aligned on an “n”-byte boundary, with an “offset” (Default=0) in bytes from that boundary

void* __mm_malloc (int size, int n)
Instructs the compiler to create a pointer to memory such that the pointer is aligned on an n-byte boundary

and tell the compiler...

#pragma vector aligned | unaligned

Vectorize using aligned or unaligned loads and stores for vector accesses, overriding compiler’s cost model

__assume_aligned(a,n)
Instructs the compiler to assume that array a is aligned on an n-byte boundary

n=16 for SSE, n=32 for AVX
...and in Fortran

!dir$ attributes align:32 ::varname

and tell the compiler...

!dir$ assume_aligned varname:32
```c
declspec(align(32)) float B[MAX];

void funca(float * B)
{
    __assume_aligned(B,32);
    A=_mm_malloc(sizeof(float)*MAX,32);
    for (I=0;I<MAX;I++)
        A[I]+=B[I];
    _mm_free(A);
}
```
Sample: Align The Data – 2nd Way

declspec(align(32)) float B[MAX];

void funca(float * B)
{
    A=_mm_malloc(sizeof(float)*MAX,32);
    #pragma vector aligned

    for (I=0;I<MAX;I++)
        A[I]+=B[I];

    _mm_free(A);
}

Intel® Compilers: some useful loop optimization pragmas/directives

- **IVDEP**
  - Ignore vector dependency
- **LOOP COUNT**
  - Advise typical iteration count(s)
- **UNROLL**
  - Suggest loop unroll factor
- **DISTRIBUTE POINT**
  - Advise where to split loop
- **PREFETCH**
  - Hint to prefetch data
- **VECTOR**
  - Vectorization hints
  - Aligned: Assume data is aligned
  - Always: Override cost model
  - Nontemporal: Advise use of streaming stores
- **NOVECTOR**
  - Do not vectorize
- **PARALLEL**
  - Override efficiency heuristics for auto-parallelization

Use where needed to help the compiler, guided by optimization reports
Switches that may help vectorization

- `-O3` (`/O3`) performs other loop transformations first
- `-ipo` (`/Qipo`) may inline, or get dependency, loop count or alignment information from calling functions
- `-xavx` (`/QxAVX`) use all available instructions
  - `-xhost` (`/QxHOST`)
- `-fno-alias` (`/Oa`) assume pointers not aliased (dangerous!)
- `-fargument-noalias` assume function arguments not aliased
  (`/Qalias-args-`)
- `-fansi-alias` assume different data types not aliased
  (`/Qansi-alias`)
- `-guide` (`/Qguide`) get advice on how to help the compiler to vectorize loops
Further Information


http://software.intel.com/en-us/articles/requirements-for-vectorizable-loops/


And the User Forums and Knowledge Base,


http://software.intel.com/en-us/articles/tools
Summary

The Intel® Compiler supports the new Intel® Advanced Vector Extensions (Intel® AVX) instruction set

• On Intel® processors, compile with /QxAVX (Windows*) or –xavx(Linux*) to take advantage of many new features automatically, without source code changes

• Automatic vectorization takes advantage of double width vector registers to increase throughput for floating point operations

• Code that vectorizes for SSE instructions should vectorize for Intel® AVX instructions

Visit http://intel.com/software/products
Consistency of Floating Point Results
or
Why doesn’t my application always give the same answer?

Developer Products Division

Software Solutions Group

Intel Corporation

November, 2011
Agenda

• Overview
• Floating Point (FP) Model
• Performance impact
• Runtime math libraries
Overview

• Some customers will not move to a new platform unless
  – Existing QA criteria are met
  – they can exactly reproduce the results from their old platform
  – Optimized builds exactly reproduce debug builds

• The right compiler options can deliver consistent, closely reproducible results whilst preserving good performance
  – Across IA-32, Intel® 64, and other IEEE-compliant platforms
  – Across optimization levels
  – -fp-model is recommended option

We encourage use of -fp-model (/fp:) switches by customers for whom floating point consistency and reproducibility are important
Floating Point (FP) Programming Objectives

• **Accuracy**
  - Produce results that are “close” to the correct value
  - Measured in relative error, possibly in ulp

• **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

These options usually conflict!
Judicious use of compiler options lets you control the tradeoffs.
Agenda

• Overview
• Floating Point (FP) Model
• Performance impact
• Runtime math libraries
Floating Point Semantics

- The –fp-model (/fp:) switch lets you choose the floating point semantics at a coarse granularity. It lets you specify the compiler rules for:
  - Value safety
  - FP expression evaluation
  - FPU environment access
  - Precise FP exceptions
  - FP contractions

In the past (still available), a mix of many, many switches ( -mp, -mp1, -pc64, -pc80 etc ) had to be used
- Not very structured, not well documented, partially inconsistent
- Replace/remove these options asap
The -fp-model switch

• -fp-model
  - fast [=1] allows value-unsafe optimizations (default)
  - fast=2 allows additional approximations
  - precise value-safe optimizations only
    (also source, double, extended)
  - except enable floating point exception semantics
  - strict precise + except + disable fma

• Replaces –mp, -float-consistency, etc

  -fp-model precise -fp-model source
  - recommended for ANSI/ IEEE standards compliance C++ & Fortran

“Floating Point Calculations and the ANSI C, C++ and Fortran Standard”
Value Safety

• In SAFE (precise) mode, the compiler may not make any transformations that could affect the result, e.g. the following is prohibited:

\[(x + y) + z \not\equiv x + (y + z)\]  \hspace{1cm} \text{general reassociation is not value safe}

• UNSAFE (fast) mode is the default
  - The variations implied by “unsafe” are usually very tiny

• VERY UNSAFE (fast=2) mode enables riskier transformations
Value Safety

• In SAFE mode, the compiler may not make any transformations that could affect the result, e.g. all the following are prohibited.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x / x ⇔ 1.0</td>
<td>x could be 0.0, ∞, or NaN</td>
</tr>
<tr>
<td>x – y ⇔ - (y – x)</td>
<td>If x equals y, x – y is +0.0 while – (y – x) is -0.0</td>
</tr>
<tr>
<td>x – x ⇔ 0.0</td>
<td>x could be ∞ or NaN</td>
</tr>
<tr>
<td>x * 0.0 ⇔ 0.0</td>
<td>x could be -0.0, ∞, or NaN</td>
</tr>
<tr>
<td>x + 0.0 ⇔ x</td>
<td>x could be -0.0</td>
</tr>
<tr>
<td>(x + y) + z ⇔ x + (y + z)</td>
<td>General reassociation is not value safe</td>
</tr>
<tr>
<td>(x == x) ⇔ true</td>
<td>x could be NaN</td>
</tr>
</tbody>
</table>

• UNSAFE mode is the default
• VERY UNSAFE mode enables riskier transformations
Value Safety

Affected Optimizations, e.g.

- Reassociation
- Flush-to-zero
- Expression Evaluation, various mathematical simplifications
- Approximate divide and sqrt
Reassociation

- Addition & multiplication are “associative” (& distributive)
  - \( a+b+c = (a+b) + c = a + (b+c) \)
  - \( a*b + a*c = a * (b+c) \)

- These transformations are equivalent *mathematically*
  - but *not* in finite precision arithmetic

- Reassociation can be disabled in its entirety
  - ⇒ for standards conformance (C left-to-right)
  - Use `-fp-model precise`
  - May carry a significant performance penalty
    (other optimizations also disabled)

  - `-assume protect_paren` (Fortran only)
    - Respects the order of evaluation specified by parentheses
Flush-to-zero (FTZ) -ftz --no-ftz options

- Flush to zero, hardware sets (flushes) all denormals to zero in SSE units
- DAZ similar, flushes to zero on data load
- -no-ftz default at -O0, -ftz at -O1 and above
  - -fp-model precise or equivalent override -ftz (set --no-ftz)
- Sets [avoids setting] the hardware flush-to-zero mode
  - On IA-32, FTZ is only set after a successful runtime processor check
  - For IA-32 and Intel 64 this only affects SSE and AVX code. There is no FTZ control for x87 (denormals always generated)
  - option -ftz available for both C and Fortran

- Must compile main with this switch to have an effect

- FTZ is NOT a guarantee that denormals in a program are flushed to zero!! It is an optimization that ALLOWS denormals to be flushed to zero.
Reductions

• Parallel implementations imply reassociation (partial sums)
  - Not value safe, but can give up to 7-8x perf advantage
• -fp-model precise
  - disables vectorization of reductions
  - does not affect OpenMP* or MPI* reductions
    These remain value-unsafe (programmer’s responsibility)

```c
float Sum(const float A[], int n )
{
    float sum=0;
    for (int i=0; i<n; i++)
        sum = sum + A[i];
    return sum;
}
```
FP Expression Evaluation

• In the following expression, what if a, b, c, and d are mixed data types (single and double for example)

\[ a = (b + c) + d \]

Four possibilities for intermediate rounding, (corresponding to C99 FLT_EVAL_METHOD)

- Indeterminate (-fp-model fast)
- Use precision specified in source (-fp-model source)
- Use double precision (C/C++ only) (-fp-model double)
- Use long double precision (C/C++ only) (-fp-model extended)

• Or platform-dependent default (-fp-model precise)

• The expression evaluation method can significantly impact performance, accuracy, and portability!
## FP Expression Evaluation under \(-fp\)-model precise

<table>
<thead>
<tr>
<th></th>
<th>IA32</th>
<th>Intel64</th>
<th>Itanium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Double (C/C++) Source (Fortran)</td>
<td>Source</td>
<td>Extended (C/C++) Source (Fortran)</td>
</tr>
<tr>
<td>Linux</td>
<td>Extended (C/C++) Source (Fortran)</td>
<td>Source</td>
<td>Extended (C/C++) Source (Fortran)</td>
</tr>
<tr>
<td>OS X</td>
<td>Extended (C/C++) Source (Fortran)</td>
<td>Source</td>
<td>NA</td>
</tr>
</tbody>
</table>

- \(-fp\)-model precise generally gives the best available performance
- When SSE2 is enabled on IA32, use \(-fp\)-model source for maximum performance
The Floating Point Unit (FPU) Environment

- FP Control Word Settings
  - Rounding mode (nearest, toward $+\infty$, toward $-\infty$, toward 0)
  - Exception masks (inexact, underflow, overflow, divide by zero, denormal, invalid)
  - Flush-to-zero (FTZ), Denormals-are-zero (DAZ)
  - x87 precision control (single, double, extended)
    - but beware of changing this!

- Status Flags
  - 1→1 mapping to exception masks
FPU Environment Access

- Affected Optimizations, e.g.
  - Constant folding
  - FP speculation
  - Partial redundancy elimination
  - Common subexpression elimination
  - Dead code elimination
  - Conditional transform, i.e.
    if (c) x = y; else x = z; → x = (c) ? y : z;
FPU Environment Access

• When access disabled (default):
  - compiler assumes default FPU environment
    - Round-to-nearest
    - All exceptions masked
    - No FTZ/DAZ
  - Compiler assumes program will NOT read status flags

• If user might change the default FPU environment, inform compiler by setting FPU environment access mode!!
  - Access may only be enabled in value-safe modes, by:
    - `-fp-model strict` or
    - `#pragma STDC FENV_ACCESS ON`
  - Compiler treats control settings as unknown
  - Compiler preserves status flags
  - Some optimizations are disabled
**Precise FP Exceptions**

- **When Disabled (default):**
  - Code may be reordered by optimization
  - FP exceptions might not occur in the “right” places
  - Especially important for x87 arithmetic

- **When enabled by**
  - `-fp-model strict`
  - `-fp-model except`
  - `#pragma float_control(except, on)`
  - The compiler must account for the possibility that any FP operation might throw an exception
    - Inserts `fwait` instructions for x87
    - Disables optimizations such as FP speculation
    - May only be enabled in value-safe modes
  - Does not unmask exceptions
    - Must do that separately
      - `-fpe0` for Fortran
      - `-fp-trap=common`
Example

double x, zero = 0.;
feenableexcept(FE_DIVBYZERO);

for( int i = 0; i < 20; i++ )
  for( int j = 0; j < 20; j++ )
    x = zero ? (1./zero) : zero;

Problem: FP exception from (1./zero) despite explicit protection
- The invariant (1./zero) gets speculatively hoisted out of loop by optimizer, but the "?" alternative does not
- exception occurs before the protection can kick in
- NOTE: does not occur for AVX due to masked vector operations

Solution: Disable optimizations that lead to the premature exception
- icc -fp-model precise -fp-model except (or icc -fp-model strict) disables all optimizations that could affect FP exception semantics
- icc -fp-speculation safe disables just speculation where this could cause an exception
- #pragma floatcontrol around the affected code block (see doc)
**Floating Point Contractions**

- affects the generation of FMA instructions on AVX2 ( -xcore-avx2 )
  - Enabled by default -fma, disable with -no-fma
  - Disabled by -fp-model strict or C/C++ #pragma
  - -[no-]fma switch overrides -fp-model setting
  - Intel compiler does NOT support 4-operand AMD*-specific fma instruction)

- When enabled

  - The compiler may generate FMA for combined multiply/add
    - Faster, more accurate calculations
    - Results may differ in last bit from separate multiply/add

- When disabled

  - The compiler must generate separate multiply/add with intermediate rounding
Agenda

- Overview
- Floating Point (FP) Model
- Performance impact
- Runtime math libraries

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel http://www.intel.com/performance/resources/limits.htm
Typical Performance Impact of -fp-model source

- Measured on SPECCPU2006fp benchmark suite:
- -O2 or –O3
- Geomean reduction due to
  -fp-model precise –fp-model source
  **in range 12% - 15%**

- Intel Compiler XE 2011 ( 12.0 )
- Measured on Intel Xeon® 5650 system with dual, 6-core processors at 2.67Ghz, 24GB memory, 12MB cache, SLES* 10 x64 SP2

**Use -fp-model source (/fp:source) to improve floating point reproducibility whilst limiting performance impact**
Agenda

• Overview
• Floating Point (FP) Model
• Performance impact
• Runtime math libraries
Math Library Functions

• Different implementations may not have the same accuracy
  - On Intel 64:
    - libsvml for vectorized loops
    - libimf (libm) elsewhere
    - Processor-dependent code within these libraries, dispatched at runtime
  - On Itanium:
    - Inlined code for many functions (to allow software pipelining)
    - libimf calls elsewhere

• No official standard (yet) dictates accuracy or how results should be rounded (except for division & sqrt)

• -fp-model precise helps generate consistent calls, eg within loops
  - Does not currently make vectorized loop consistent with non-vectorized
New in 12.x, Selection of libimf libraries
-`fimf-precision` -`fimf-arch-consistency`

-`fimf-precision=<$high|medium|low>$`
  - Default is off (compiler chooses)
    - Typically high for scalar code, medium for vector code
    - “low” typically halves the number of mantissa bits
    - “high” ~0.55 ulp; “medium” < 4 ulp (typically 2)

-`fimf-arch-consistency=<$true | false>$`
  - Will produce consistent results on all microarchitectures or processors within the same architecture
  - Run-time performance may decrease
  - Default is false (even with `--fp-model precise` !)

Windows form:
- `/Qimf-precision:medium`
- `/Qimf-arch-consistency:true` etc
-prec-div and -prec-sqrt Options

Both override the -fp-model settings

Default is -no-prec-sqrt, and somewhere between -prec-div and -no-prec-div

[-no]-prec-div /Qprec-div[-]

Enables[disables] various divide optimizations
  •  \( x / y \leftrightarrow x \times (1.0 / y) \)
  •  Approximate divide and reciprocal

[-no]-prec-sqrt /Qprec-sqrt[-]

Enables[disables] approximate sqrt and reciprocal sqrt
Math Libraries – known issues

• Differences could potentially arise between:
  - Different compiler releases, due to algorithm improvements
    - Use –fimf-precision
    - another workaround, use later RTL with both compilers
  - Different platforms, due to different algorithms or different code paths at runtime
    - Libraries have internal processor dispatch
    - Independent of compiler switches
    - use -fimf-arch-consistency=true
  - Expected accuracy is maintained
    - 0.55 ulp for libimf
    - < 4 ulp for libsvml (vectorized loops)
• Adherence to an eventual standard for math functions would improve consistency but at a cost in performance.
More detail

• Can specify at the function level
  – -fimf-precision=<high|medium|low>[:fnlist]
    – e.g. -fimf-precision=low:func1,func2,func3
  – -fimf-arch-consistency=<true | false> [:fnlist]

• Can specify desired accuracy in different ways:
  – -fimf-max-error=ulps‡[:fnlist]
    – Maximum relative error
    – E.g. -fimf-max-error:0.6 for high accuracy
  – -fimf-absolute-error=value[:fnlist]
    – Max absolute error specified as a floating-point number
  – -fimf-accuracy-bits=bits[:fnlist]
    – Required accuracy specified as a number of mantissa bits

‡ulps = Units in the Last Place
Implementation

• No new run-time libraries, but new entry points
  – High accuracy functions typically have names ending in _ha
  – Low accuracy functions typically have names ending in _ep
    – “extra performance”
    – About half the number of bits of the high accuracy version
    – Currently implemented for libsvm1, but not libm
  – Bit-wise reproducible functions typically have names starting with __bwr or terminating in _br
    – In some cases, the _ha functions are bit-wise reproducible and so no _br version is needed

• New compile-time library libiml_attr
  – Tells the compiler which libm entry point to call
  – .so or .dll located in bin directory
Further Information

• Microsoft Visual C++* Floating-Point Optimization

• The Intel® C++ and Fortran Compiler Documentation, “Floating Point Operations”

  “Floating Point Calculations and the ANSI C, C++ and Fortran Standard”

Coarray Fortran Fundamentals

Simple extension to Fortran to make Fortran into a robust and efficient parallel programming language

Single-Program, Multiple-Data programming model
• Single program is replicated a fixed number of times
• Each program instance has its own set of data objects – called an “IMAGE”
• Each image executes asynchronously
• Extensions to normal Fortran array syntax to allow images to reference data in other image(s)

Part of the Fortran 2008 standard

Shared-memory in Fortran Composer XE for Windows* and Linux*

Distributed-memory supported in Linux only, Intel® Cluster Studio product line

*Other brands and names are the property of their respective owners.
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-images=x
ifort /Qcoarray /Qcoarray-num-images=x
along with other options. Sets number of images to “x”.
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, env var: FOR_COARRAY_NUM_IMAGES

Environment variable can set number of images

Environment variable overrides -coarray-num-images compiler option

Linux host> export FOR_COARRAY_NUM_IMAGES=2

./hello_image

Window host> set FOR_COARRAY_NUM_IMAGES=2

hello_image.exe

Hello from image 1 out of 2 total images

Hello from image 2 out of 2 total images
CAF Fundamentals: Determining Number of Images, num_images()

Intrinsic function num_images() returns an integer result, the total number of images in the CAF program:

$> cat hello_num_images.f90
program hello_num_images
  write(*,*) "Hello there are ", num_images()," total images"
end program hello_num_images

$> ifort -coarray -coarray-num-procs=4 hello_num_images.f90
$> ./a.out
Hello there are 4 total images
Hello there are 4 total images
Hello there are 4 total images
Hello there are 4 total images
Coarray Fundamentals: this_image()

Images have a logical ordering from 1 to N

Integer function this_image() without an argument returns unique logical ordering from 1 to N

• More complex image mappings possible: 2D, 3D, etc with arguments

```fortran
$> cat hello_this.f90
program hello_this_image
   write(*,*) "Hello from image ", this_image()
end program hello_this_image
$> ifort -coarray -coarray-num-procs=4 hello_this.f90
$> ./a.out
Hello from image 1
Hello from image 3
Hello from image 2
Hello from image 4

Remember, the images are inherently asynchronous
Some Advice Before Some Examples

CAF behavior rule of thumb: when questioning the behavior of CAF ask “what would the Fortran semantics imply here” – follow Fortran rules

The “[]” codimension syntax is a visual clue to where communication to remote images is performed (implies OVERHEAD, implies possible performance drops)

There are many restrictions to where coarrays can be used: Simply put: any attempt to alias a coarray with a non-coarray object are prohibited:

• Pointers that are not coarrays
• Non-coarray dummy args passed coarrays
• Passing coarray object to C or another language
• COMMON, EQUIVALENCE, etc
SYNC ALL statement global barrier: requires all images to join the synchronization point

sync images() allows synchronization with a subset of images. The image set is an integer scalar holding an image index, an integer array of rank 1 holding distinct image indices, or an asterisk to indicate all images,

Critical sections can be created, bounded by CRITICAL ; END CRITICAL

SYNC MEMORY ensures any changed data that is held in temporary storage (cache, registers) or in transit between images is made visible to the other image
Additional Synchronization

LOCK and UNLOCK statements provide fine-grained control

ERROR STOP stops execution on all images immediately with error code

Implicit global synchronization at ALLOCATE, DEALLOCATE of coarrays

• When coarray is allocated on one image, wait until all images allocate their copy. Otherwise, one image could attempt to access unallocated coarray data on another image

• Similar on DEALLOCATE: Wait to remove the coarray data until all images synch and deallocate: otherwise, other images could try to access deallocated coarray data
CAF Fundamentals - Input/Output

Each image has its own set of connected units

Default output unit is preconnected on all images
• Assumption is that processor will merge the streams

Default input unit is preconnected on image 1 only
Further Reading

Coarrays in the next Fortran Standard

The New Features of Fortran 2008

Fortran 2008 Standard (current draft)
• http://j3-fortran.org/doc/standing/links/007.pdf
**Intel® Cilk™ Plus**

**What is it?**
- Compiler supported solution offering a tasking system via 3 simple keywords
- Includes array notation to specify vector code
- Reducers - powerful parallel data structures to efficiently prevent races
- Based on 15 years of research at MIT
- Pragmas to force vectorization of loops and attributes to specify functions that can be applied to all elements of arrays

**Key Benefits**
- Simple syntax which is very easy to learn and use
- Array notation guarantees fast vector code
- Fork/join tasking system is simple to understand and mimics serial behavior
- Low overhead tasks offer scalability to high core counts
- Reducers give better performance than mutex locks and maintain serial semantics
- Mixes with Intel® TBB and Intel® ArBB for a complete task and vector parallel solution
Intel® Cilk™ Plus keywords

Cilk Plus adds three keywords to C and C++:

_cilk_spawn
_cilk_sync
_cilk_for

If you #include <cilk/cilk.h>, you can write the keywords as cilk_spawn, cilk_sync, and cilk_for.

Cilk Plus runtime controls thread creation and scheduling. A thread pool is created prior to use of Cilk Plus keywords.

The number of threads matches the number of cores by default, but can be controlled by the user.
cilk_spawn and cilk_sync

cilk_spawn gives the runtime *permission* to run a child function asynchronously.

• No 2\textsuperscript{nd} thread is created or required!
• If there are no available workers, then the child will execute as a serial function call.
• The scheduler may *steal* the parent and run it in parallel with the child function.
• The parent is not *guaranteed* to run in parallel with the child.

cilk_sync waits for all children to complete before execution proceeds from that point.
• There are implicit cilk_sync points – will discuss later
A simple example

Recursive computation of a Fibonacci number:

```c
int fib(int n)
{
    int x, y;

    if (n < 2) return n;

    x = cilk_spawn fib(n-1);
    y = fib(n-2);
    cilk_sync;
    return x+y;
}
```

Execution can continue while `fib(n-1)` is running.

Asynchronous call must complete before using `x`.
cilk_for loop

Looks like a normal for loop.

```
cilk_for (int x = 0; x < 1000000; ++x) { ... }
```

Any or all iterations may execute in parallel with one another.

All iterations complete before program continues.

Constraints:

• Limited to a single control variable.
• Must be able to jump to the start of any iteration at random.
• Iterations should be independent of one another.
Serialization

Every Cilk Plus program has an equivalent serial program called the *serialization*

The serialization is obtained by removing `cilk_spawn` and `cilk_sync` keywords and replacing `cilk_for` with `for`

- The compiler will produce the serialization for you if you compile with `/Qcilk-serialize` (Windows)

Running with only one worker is equivalent to running the serialization.
Serial Semantics

A deterministic Cilk Plus program will have the same semantics as its serialization.

• Easier regression testing
• Easier to debug:
  - Run with one core
  - Run serialized
• Composable
• Strong analysis tools (Cilk Plus-specific versions will be posted on WhatIf)
  - race detector
  - parallelism analyzer
int compute(const X& v);
int main()
{
    const std::size_t n = 1000000;
    extern X myArray[n];
    // ...

    int result = 0;
    for (std::size_t i = 0; i < n; ++i)
    {
        result += compute(myArray[i]);
    }
    std::cout << "The result is: "
                << result
                << std::endl;
return 0;
}
Summing Example in Intel® Cilk™ Plus

```cpp
int compute(const X& v);
int main()
{
    const std::size_t n = 1000000;
    extern X myArray[n];
    // ...

    int result = 0;
    cilk_for (std::size_t i = 0; i < n; ++i)
    {
        result += compute(myArray[i]);
    }
    std::cout << "The result is: "
              << result
              << std::endl;
    return 0;
}
```

Race!
Locking Solution

```cpp
int compute(const X& v);
int main()
{
    const std::size_t n = 1000000;
    extern X myArray[n];
    // ...

    mutex L;
    int result = 0;
    cilk_for (std::size_t i = 0; i < n; ++i)
    {
        int temp = compute(myArray[i]);
        L.lock();
        result += temp;
        L.unlock();
    }
    std::cout << "The result is: "
               << result
               << std::endl;
    return 0;
}
```

Problems

Lock overhead & lock contention.
int compute(const X& v);

int main()
{
    const std::size_t ARRAY_SIZE = 1000000;
    extern X myArray[ARRAY_SIZE];
    // ...

cilk::reducer_opadd<int> result;
    cilk_for (std::size_t i = 0; i < ARRAY_SIZE; ++i) {
        result += compute(myArray[i]);
    }
    std::cout << "The result is: " << result.get_value() << std::endl;
    return 0;
}
Reducer Library

Intel® Cilk™ Plus’s hyperobject library contains many commonly used reducers:

- `reducer_list_append`
- `reducer_list_prepend`
- `reducer_max`
- `reducer_max_index`
- `reducer_min`
- `reducer_min_index`
- `reducer_opadd`
- `reducer_ostream`
- `reducer_basic_string`
- ...

You can also write your own using `cilk::monoid_base` and `cilk::reducer`. 
Reducer Benefits/Limitations

Benefits

• Reducers do not suffer from lock contention
• Reducers retain serial semantics
  - For example, a list or ostream reducer will retain the same order you would get from serial execution
  - Even a correct locking solution cannot guarantee this

Limitations

• Operations on a reducer must be associative to behave deterministically
  - Refer to the operators supported by a particular reducer class for safe operations to use
  - Floating point types may get different results run-to-run
• If using custom data types for reducers, refer to the header for the specific reducer for requirements
Array Notations provide a syntax to specify sections of arrays on which to perform operations.

Syntax: $[<\text{lower bound}> : <\text{length}> : <\text{stride}>]$  

Simple example:
- $a[0:N] = b[0:N] * c[0:N]$;  
- $a[:] = b[:] * c[:] // \text{if } a, b, c \text{ are declared with size N}$

The Intel® C++ Compiler’s automatic vectorization can use this information to apply single operations to multiple elements of the array using Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX).
- Default is SSE2. Use compiler options (\text{/Qx, /arch, /Qax}) to change the target.

More advanced example:
- $x[0:10:10] = \sin(y[20:10:2])$;
Intel® Cilk™ Plus Array Notations

Example

void foo(double * a, double * b, double * c, double * d, double * e, int n) {
    for(int i = 0; i < n; i++)
        a[i] *= (b[i] - d[i]) * (c[i] + e[i]);
}

void goo(double * a, double * b, double * c, double * d, double * e, int n) {
    a[0:n] *= (b[0:n] - d[0:n]) * (c[0:n] + e[0:n]);
}

icl -Qvec-report3 -c test-array-notations.cpp
Intel® Cilk™ Plus Elemental Functions

The compiler can’t assume that user-defined functions are safe for vectorization.

Now you can make your function an elemental function which indicates to the compiler that such a function can be applied to multiple elements of an array in parallel safely.

Specify __declspec(vector) on both function declarations and definitions as this will affect name-mangling.
**Intel® Cilk™ Plus Elemental Functions Example**

```c
double user_function(double x);
_declspec(vector) double elemental_function(double x);

void foo(double *a, double *b, int n) {
    a[0:n] = user_function(b[0:n]);
}

void goo(double *a, double *b, int n) {
    a[0:n] = elemental_function(b[0:n]);
}
```

`icl /Qvec-report3 /c test-elemental-functions.cpp`


`test-elemental-functions.cpp(9) (col. 2): remark: LOOP WAS VECTORIZED.`
What’s New: Intel® Cilk™ Plus v1.1
Implemented with Commercial Support; Simplifies Going Parallel

- Enhanced performance and utilization of future Intel CPU features
- SIMD pragma loops, vector length, and elemental functions support
- Mac OS* support

```c
int fib (int n) {
    if (n <= 2)
        return n;
    else {
        int x,y;
        x = fib(n-1);
        y = fib(n-2);
        return x+y;
    }
}
```

```c
int fib (int n) {
    if (n <= 2)
        return n;
    else {
        int x,y;
        x = __cilk_spawn fib(n-1);
        y = fib(n-2);
        __cilk_sync;
        return x+y;
    }
}
```

Turn serial code

Open spec at: cilkplus.org

Parallel loops made easy

Into parallel code
What’s New: Intel® Cilk™ Plus
Increased Performance and Scalability

• Improved SIMD pragma loops and elemental functions vectorization support provides enhanced scalability and performance

• Enhanced performance and productivity with new Holder Hyperobjects for per-thread temporary storage feature

• SIMD pragma loops and elemental functions support for nested loops, array notation, switch statements, and break/continue statements

• More architectural and scalable way to define vector lengths with new SIMD pragma clause “vectorlengthfor” support of vectorization of loops and elemental functions

• Expanded Mac OS support
Intel® Math Kernel Library Overview
Intel® Math Kernel Library
The industry leading math library*

Includes linear algebra, FFT, vector math and statistics functions for science, engineering and financial applications

Parallelized for maximum performance -- vectorized, multicore-threaded and distributed multiprocessor aware

Just re-link with the newest version to take full advantage of new processors with minimal programming effort

Provides standard C and Fortran APIs for Windows, Linux and Mac OS

"Intel MKL is indispensable for any high-performance computer user on x86 platforms."

Prof. Jack Dongarra, Innovative Computing Lab, University of Tennessee

*Source: 2011 Evans Data N. American developer survey
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

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

**FFTs**
- Mixed radix, multi-dimensional transforms

**Cluster DFT**
- For Distributed Memory systems

**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

**Summary Statistics**
- Basic statistics, Estimation of Dependencies, Data with Outliers, Missing Values

**PDEs (Partial Differential Equations)**
- Trigonometric transform and Poisson solvers.
Continued Performance Improvement using Intel® Math Kernel Library
SMP LINPACK Performance (4 threads)

Up to 1.9x performance improvements with Intel® Advanced Vector Extensions optimizations

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Intel MKL 10.1 Update 1</th>
<th>Intel MKL 10.2 Update 6</th>
<th>Intel MKL 10.3 Update 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>1.00x</td>
<td>1.01x</td>
<td>1.78x</td>
</tr>
<tr>
<td>10000</td>
<td>1.00x</td>
<td>1.00x</td>
<td>1.81x</td>
</tr>
<tr>
<td>20000</td>
<td>1.00x</td>
<td>1.00x</td>
<td>1.90x</td>
</tr>
</tbody>
</table>

Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 10.1.1, 10.2.6 & 10.3.2; Hardware: Intel® Core® i7-2600 Processor, 3.40Ghz, 8MB L2 cache, 4GB Memory; Operating System: Fedora 14 x86_64; Benchmark Source: Intel Corporation.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/technology/performance/benchmark_limitations.htm.

Refer to our Optimization Notice for more information regarding performance and optimization choices in Intel software products at: http://software.intel.com/en-us/articles/optimization-notice/

* Other brands and names are the property of their respective owners.
Performance Improves using Intel® Math Kernel Library versus ATLAS*
DGEMM on Intel® Desktop Processor

Intel® MKL offers significant performance boost over ATLAS*
Performance scales as number of CPU cores increase
Performance up to 93% of CPU Gflop/s peak!

Configuration Info - Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 10.3.7 ATLAS 3.8.4; Hardware: Intel® Core® i7-2600 Processor, 3.40Ghz, 8 MB L2 cache, 4 GB Memory; Operating System: Fedora 14 x86_64; Benchmark Source: Intel Corporation.
Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark limitations.htm.
* Other brands and names are the property of their respective owners.
Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804
**FFTs**

2D FFT Performance Improves using Intel® Math Kernel Library versus FFTW* on Intel® Desktop Processor

- Intel® MKL offers a generally higher performance increase over FFTW*
- Performance scales as number of CPU cores increase

Configuration Info:
- Versions: Intel® Math Kernel Library (Intel® MKL) 10.3.7, FFTW 3.3
- Hardware: Intel® Core® i7-2600 Processor, 3.40GHz, 8 MB L2 cache, 4 GB Memory; Operating System: Fedora 14 x86_64
- Benchmark Source: Intel Corporation

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, refer to www.intel.com/performance/resources/benchmark_limitations.htm.

* Other brands and names are the property of their respective owners.

Optimization Notice: Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110604

Visit the Intel MKL website for more information: http://www.intel.com/software/products/mkl
What's new in Intel® MKL 10.3

Support for Intel® Advanced Vector Extensions (Intel® AVX)
- Faster floating point operations in BLAS, LAPACK, FFTs, VML and VSL

C interfaces for LAPACK and PARDISO
- C LAPACK interfaces supporting row-major ordering and support for c-style (zero-based) array indexing for PARDISO arrays

New Intel® Summary Statistics Library
- New domain covering a broad range of statistics functions

Dynamic accuracy control for VML
- New interfaces for all VML functions that include parameters for setting accuracy mode

Additional optimizations
- BLAS, LAPACK, PARDISO, FFTs, and VSL
Summary Statistics: Key Functionality & Features

Functionality

Basic statistics
- Moments, skewness, kurtosis, variation coefficient, quantiles and order statistics.

Estimation of Dependencies
- Variance-covariance/correlation matrix, partial variance-covariance/correlation matrix, pooled/group variance-covariance/correlation matrix.

Data with Outliers
- Detection of outliers in “noised” data, robust (to noise) estimates of the covariance matrix and mean

Missing Values
- Restoring statistical characteristics in presence of missed observations

Features

Out-of-Memory Datasets
- Addresses cases when data comes in portions or when whole dataset doesn’t fit into memory

Various Data Storage Formats
- Flexibility for users, in-row/in-column packed, full/packed matrix

Enhanced accuracy and performance due to modern algorithms
## Linking

- static linking
- dynamic linking
- custom dynamic linking
- runtime dispatching dynamic libraries

<table>
<thead>
<tr>
<th>Feature</th>
<th>Dynamic Linkage</th>
<th>Static Linkage</th>
<th>Custom Dynamic Linkage</th>
<th>Single Dynamic Library Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Updates</td>
<td>Automatic</td>
<td>Automatic</td>
<td>Recompile and redistribute</td>
<td>Automatic</td>
</tr>
<tr>
<td>Optimization</td>
<td>All Processors</td>
<td>All Processors</td>
<td>All Processors</td>
<td>All Processors</td>
</tr>
<tr>
<td>Build</td>
<td>Link to import libraries</td>
<td>Link to static libraries</td>
<td>Build separate import libraries, which are created automatically</td>
<td>Link only to mkl_rt library (Linux – libmkl_rt.so*&lt;br&gt;Mac OS – libmkl_rt.dylib)</td>
</tr>
<tr>
<td>Calling</td>
<td>Regular Names</td>
<td>Regular Names</td>
<td>Regular Names</td>
<td>Regular Names</td>
</tr>
<tr>
<td>Total Binary Size</td>
<td>Large</td>
<td>Small</td>
<td>Small</td>
<td>Largest</td>
</tr>
<tr>
<td>Executable Size</td>
<td>Smallest</td>
<td>Small</td>
<td>Smallest</td>
<td>Smallest</td>
</tr>
<tr>
<td>Multi-threaded/ thread safe</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

In Dynamic libraries linking for Linux, corresponding OpenMP libs should also be linked except in sequential case.
Layered model approach for better control

- **Interface Layer**
  - Compiler: Intel / GNU
  - LP64 / ILP64

- **Threading Layer**
  - Threaded / alternate OpenMP
  - Sequential

- **Computational Layer**

- **Run-time Layer**

**Ex 1**: Static linking using Intel® Fortran Compiler, BLAS, Intel® 64 processor on Linux
$ifort myprog.f libmkl_intel_lp64.a libmkl_intel_thread.a libmkl_core.a libiomp5.so

**Ex 2**: Dynamic linking with Intel® C++ compiler on Windows*
c:\>icl myprog.c mkl_intel_lp64_dll.lib mkl_intel_thread_dll.lib mkl_core_dll.lib libiomp5md.dll

**Ex 3**: Using MKL Dynamic Interface with Intel® C++ compiler on Mac*
$icc myprog.c libmkl_rt.dylib

**Note**: We strongly recommend linking the OpenMP run-time library dynamically

Check your understanding

We’re providing tools, not guarantees

We don’t expect repeatable results across
• OS
• architecture
• different numbers of threads
• all supported CPUs (e.g., Intel MIC)

Still required of Intel MKL user
• align memory: CPU-dependent, 32-bytes for Intel® AVX
• set codepath
• enable run-to-run repeatability to get static task scheduling
Intel Composer XE - Summary

• Intel Composer XE provides you advanced optimization techniques for the latest processors.

• Intel Programming Models (e.g. Intel® Cilk™ Plus and Coarray Fortran) support parallelization and vectorization

• Evaluations of all Intel® Parallel Studio XE products at:

  http://www.softwareproducts.intel.com

http://www.intel.com/software/products/support/
Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS
OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR
WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR
INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Performance tests and ratings are measured using specific computer systems and/or components
and reflect the approximate performance of Intel products as measured by those tests. Any
difference in system hardware or software design or configuration may affect actual performance.
Buyers should consult other sources of information to evaluate the performance of systems or
components they are considering purchasing. For more information on performance tests and on

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino
Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386,
Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside,
NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel
XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCcharm, Pentium, Pentium
Inside, skooool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon
Inside are trademarks of Intel Corporation in the U.S. and other countries.

http://intel.com/software/products

*Other names and brands may be claimed as the property of others.
## Intel® Parallel Studio XE Family Suites

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Intel® Parallel Studio XE</th>
<th>Intel® C++ Studio XE</th>
<th>Intel® Fortran Studio XE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Inspector XE Memory and Thread Checker</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® Static Security Analysis</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intel® VTune™ Amplifier XE Performance Profiler</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compilers</th>
<th>Intel® C++ Compiler</th>
<th>Intel® Fortran Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Intel® Integrated Performance Primitives</th>
<th>Intel® C++ Studio XE</th>
<th>Intel® Fortran Studio XE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Intel® Threading Building Blocks</th>
<th>Intel® C++ Studio XE</th>
<th>Intel® Fortran Studio XE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Intel® Math Kernel Library</th>
<th>Intel® C++ Studio XE</th>
<th>Intel® Fortran Studio XE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool</th>
<th>Intel® Parallel Advisor C++ Windows available (separate download)</th>
<th>Intel® Parallel Studio XE</th>
<th>Intel® C++ Studio XE</th>
<th>Intel® Fortran Studio XE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Windows MSRP</th>
<th>Windows SSR</th>
<th>Linux MSRP</th>
<th>Linux SSR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1,899</td>
<td>$1,499</td>
<td>$1,599</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$759</td>
<td>$599</td>
<td>$639</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$2,249</td>
<td>$1,499</td>
<td>$1,799</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$899</td>
<td>$599</td>
<td>$719</td>
<td></td>
</tr>
</tbody>
</table>

*Other brands and names are the property of their respective owners.*
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY
ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS
DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR
IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES
RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY
PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on
Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using
specific computer systems, components, software, operations and functions. Any change to any of
those factors may cause the results to vary. You should consult other information and performance
tests to assist you in fully evaluating your contemplated purchases, including the performance of that
product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk
are trademarks of Intel Corporation in the U.S. and other countries.

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that
are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and
other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on
microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended
for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for
Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information
regarding the specific instruction sets covered by this notice.

Notice revision #20110804