Intel® Compiler Professional Edition
Extended Support for Parallel Programming
Agenda

• Overview
• Intel® Compiler – Features and Q2 2009 Update
• Optimization With Intel Compiler Pro
• Intel® Compiler 11.1 Preview
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• Summary, References and Call to Action
Two Product Lines for Two Kinds of Developers

Maximize parallel performance
C++ and Fortran on Windows*, Linux* and Mac OS* X
Sample: Intel® C++ Compiler Professional Edition

Available Now

Maximize parallel productivity
C++ using Visual Studio on Windows
Sample: Intel® Parallel Composer

Beta on-going, available Q2/09

Q2 2009 DPD Tools Training
© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.
*Other names and brands may be claimed as the property of others.
Intel® Parallel Studio
Intuitive development tools for multicore parallelism
For Microsoft Visual Studio* C++ architects, developers, and software innovators creating parallel Windows* applications.

Intel® Parallel Studio includes all three:
• Intel® Parallel Composer
• Intel® Parallel Inspector
• Intel® Parallel Amplifier

Microsoft Visual Studio* plug-in
End-to-end product suite for parallelism
Future scaling to manycore
Four Steps to a Parallel Application

DESIGN – Intel® Parallel Advisor Lite
Gain insight on where parallelism will most benefit existing source code – usually begins with a “hotspot” (whatif.intel.com technology)

CODE, DEBUG – Intel® Parallel Composer
Develop effective applications with a C/C++ compiler and comprehensive threaded libraries and API’s, and Parallel Debugger Extension

VERIFY – Intel® Parallel Inspector
Help ensure application reliability with proactive parallel memory and threading error checking

TUNE – Intel® Parallel Amplifier
Enhance applications with an intuitive performance analyzer and tuner

www.intel.com/go/parallel
## Why should you buy?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel® Parallel Studio</th>
<th>High Performance Tools†</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Easy to learn ↔ Maximize Performance</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Easy to learn. Get productive quickly.</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td>Many options to let you maximize performance.</td>
<td></td>
<td>➢</td>
</tr>
<tr>
<td><strong>Development environments</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Visual Studio* on Windows*</td>
<td>➢</td>
<td>➢</td>
</tr>
<tr>
<td>Windows standalone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux*, Mac OS*</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td><strong>Languages</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C/C++</td>
<td>➢</td>
<td>➢</td>
</tr>
<tr>
<td>Fortran</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td>Assembly</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td><strong>Processor Support</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IA32 + Intel® 64</td>
<td>➢</td>
<td>➢</td>
</tr>
<tr>
<td>Intel® Itanium™ Architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Licensing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single user, single version</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td>Single user, all product updates for 1 year</td>
<td></td>
<td>➢</td>
</tr>
<tr>
<td>Floating, all product updates for 1 year</td>
<td></td>
<td>➢</td>
</tr>
<tr>
<td><strong>Support</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Community support with user forum</td>
<td>➢</td>
<td></td>
</tr>
<tr>
<td>Premier support</td>
<td>optional</td>
<td></td>
</tr>
</tbody>
</table>

†Features vary by product. Not all high performance tools have all features. Check each product for details.
New Intel® Compiler 11.0 Professional Editions include:

– Intel C++ Compiler and/or Fortran Compiler 11.0
– Intel Math Kernel Library 10.1
– Intel Integrated Performance Primitives 6.0
– Intel Threading Building Blocks 2.1

Support for Windows*, Linux*, Mac OS X*

Most comprehensive multicore and standards support

OpenMP* 3.0, auto-vectorization, auto-parallelization, parallel valarray, “parallel lint”

Advanced compilers and libraries support Intel and compatible processors in a single binary
Intel Compiler Pro, Extended Support for Parallel Programming

- Enhanced OpenMP Support
  - OpenMP* 3.0, Compatibility Run-Time System, Affinity
- Parallel Debugger
- Improved Automatic Parallelization
- Vectorization for SSE4.x, AVX
- Threading for Static Linkage in Intel® IPP
- Improved Threading in Intel® MKL
- Threading Building Blocks (TBB) extended by more Parallel Templates and Data Structures
## Release Components

<table>
<thead>
<tr>
<th></th>
<th>Intel® Compiler</th>
<th>Intel® MKL</th>
<th>Intel® IPP</th>
<th>Intel® TBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pro Edition 10.1</td>
<td>10.1</td>
<td>10.0</td>
<td>5.3</td>
<td>2.0</td>
</tr>
<tr>
<td>Pro Edition 11.0</td>
<td>11.0</td>
<td>10.1</td>
<td>6.0</td>
<td>2.1</td>
</tr>
<tr>
<td>Released Nov 08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Edition</td>
<td>11.1</td>
<td>10.2</td>
<td>6.1</td>
<td>2.1 (Update)</td>
</tr>
<tr>
<td>11.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In Beta today; release planned for end of Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.*
Agenda

• Overview
• **Intel® Compiler – Features and Q2 2009 Update**
• Optimization With Intel Compiler Pro
• Intel® Compiler 11.1 Preview
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• Summary, References and Call to Action
## 11.0 Compilers – Feature Summary

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenMP™ 3.0</strong></td>
</tr>
<tr>
<td>C++ lambda functions</td>
</tr>
<tr>
<td>Parallel debugger for IA-32/Intel®64 Linux, new GUI</td>
</tr>
<tr>
<td>Option to emit FE diagnostics related to threading</td>
</tr>
<tr>
<td>Implementation of valarray using IPP for array notation</td>
</tr>
<tr>
<td>Eclipse CDT 4.0 support</td>
</tr>
<tr>
<td>New and extended switches for SSE-vectorization</td>
</tr>
<tr>
<td>Increased F2003 feature support beyond 10.1: ENUMERATOR, MIN/MAX and friends, IEEE FP, ASSOCIATE, PROCEDURE POINTER, ABSTRACT INTERFACE, TYPE EXTENDS, structure constructor change, array constructor changes, ALLOCATE scalar</td>
</tr>
<tr>
<td>Native (64 bit) Compiler for Intel®64</td>
</tr>
<tr>
<td>Additional C++0x features</td>
</tr>
<tr>
<td>Decimal floating point arithmetic</td>
</tr>
</tbody>
</table>
OpenMP* 3.0 Support

- **Intel® Compilers 11.0 fully compliant to OpenMP* 3.0**
  - Released May 2008
  - Standard documentation at [www.openmp.org](http://www.openmp.org)
- Four major extensions:
  - Tasking for unstructured parallelism
  - Loop collapsing
  - Enhanced loop scheduling control
  - Better support for nested parallelism
OpenMP* 3.0 Tasking
The most Relevant New Feature

• A task has
  – Code to execute
  – A data environment (it owns its data)
  – Eventually (when selected for execution): An assigned thread that executes the code and uses the data

• Two activities: packaging and execution
  – Each encountering thread packages a new instance of a task (code and data)
  – Some thread in the team executes the task at some later time

• A task is nothing really new to OpenMP
  – Implicitly each PARALLEL directive creates tasks but they have been transparent objects
  – OpenMP* 3.0 makes tasking explicit
void postorder(node *p) 
{
    if (p->left)
        #pragma omp task
        postorder(p->left);
    if (p->right)
        #pragma omp task
        postorder(p->right);
    #pragma omp taskwait // wait for descendants
    process(p->data);
}

Parent task suspended until children tasks complete
New Linux* Intel® Debugger (IDB)
GUI Support for Debugging Threaded Parallel Code

- Considerably enhanced Intel® Debugger for Linux* IA-32 and Intel® 64
- Eclipse Rich Client Platform based GUI
- Command line engine remains traditional IDB
- New and Enhanced Visibility of Parallelism
  - Threading/OpenMP*
  - SSE parallelism
    - Command-line version for IA-64
      - No new parallel debugging features
IDB: New Graphical User Interface (1)

- Rich user friendly multi-Window GUI
- GUI-centric debugger usage model
  - Menus and Tool bars
  - Multiple source windows
  - Context sensitive operations
    - Drag & Drop support
    - Dynamic window menus based on active selection
    - Set breakpoints via Mouse clicks
- Command line support
  - Dedicated console window
  - Command language compatible to console version (GDB style)
- True Eclipse look & feel
IDB: New Graphical User Interface (2)

Variable Views and Symbol Evaluation
IDB SSE and Vector Evaluation Window

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.
Debugging Multi-threaded Applications

- **Thread Control**
  - **Flexible Thread Execution Models**
    - Concept of “focus” thread-set for next/step/finish
    - Freezing and thawing of individual threads
    - Can detach threads from debug control
  - **Thread Grouping**
    - Smart default groupings by debugger (All, Team, Lockstep)
    - Explicit grouping by user if required
  - **Thread Group aware breakpoints**
    - Break when any member of the trigger group hit
    - Or can stop a specific thread-set only
  - **Thread Synchronizing Breakpoint “syncpoint”**

- **OpenMP* Support**
  - **Dedicated OpenMP Info**
    - Windows
      - Threads
      - Teams
      - Tasks
      - Task spawn trees
      - Barriers*
      - Taskwaits*
      - Locks
  - **Serial execution of a parallel region at debug time**
  - **Technology**
    - Uses special OpenMP RTL

User benefits:
- Provide outstanding execution control for multithreaded applications without added complexity for serial code debugging.
- Provide serialization of parallel region and detailed information on OpenMP constructs- The only debugger capable of doing this.
**IDB Parallel Run-Control Use Cases**

### Stepping parallel loops

**Problem:**
State investigation difficult. Threads stop at arbitrary positions (red line)

**Parallel Debugger Support**
Add Syncpoint to stop team threads at same location
Apply execution to $LockStep set.

**User Benefit**
Get and keep defined program State
Operations like private data comparison now meaningful

### Serial Execution

**Problem:**
Parallel loop computes a wrong result. Is it a concurrency or algorithm issue?

**Parallel Debugger Support**
Runtime access to the OpenMP num_thread property
Set to 1 for serial execution of next parallel block

**User Benefit**
Verification of an algorithm “on-the-fly” without slowing down the entire application to serial execution
On demand serial debugging without recompile/restart
The Intel® Debugger provides a new set of debugger windows supplying OpenMP* information.

- task spawn tree
- tasks
- task waits
- teams
- barriers
- locks
Diagnostic for Threading

• Switch -diag-enable (Linux*/Mac OS*), /Qdiag-enable (Windows*)
• Enables front-end diagnostics useful to parallelize a program for multi-threading
• Displays warnings about
  • references to statically allocated variables
  • assignments to statically allocated variables
  • address references of statically allocated variables

```c
void sub()
{
    static int number_of_calls=0;
    number_of_calls++;
}
```

icc -c -diag_enable thread test.c

test.c(5): warning #1711: assignment to statically allocated variable "number_of_calls"
Decimal Floating Point

Software support for IEEE* 754R Standard

- Why? Applications that involve financial computations like banking, telephone billing, insurance, e-commerce, accounting in general critically suffer from unavoidable precision issues implied by binary FP formats – no matter how many bits are used.

Sample:

```c
float a = 7/10000.0, b = 10000.0 * a;
printf("a = %x = %10.10f\n", *(unsigned int*)&a, a);
printf("b = %x = %10.10f\n", *(unsigned int*)&b, b);
a = 3a378034 = 0.0007000000    b = 40dfffff = 6.9999997504
```

Error fixed by:

```c
float a = 7/10000.0, b = 10000.0 * a;
```

For details see ISO/IEC document N1176

http://www.open-std.org/JTC1/SC22/WG14
Some Decimal Arithmetic Details

New library class types in namespace std::decimal

- `decimal32`, `decimal64`, `decimal128`
- All relevant arithmetic operations enabled via decimal arithmetic library
- Existing math functions available as new decimal versions
  - Trigonometric, hyperbolic, exponential, logarithmic, power, error, gamma
  - Nearest integer, remainder, manipulation, max/min, FMA
- New functionality
  - `quantize`, `samequantum`
  - `quantexp`
  - `printf/scanf` (added length modifiers H, D, DD)
  - `strtodN`, `wcstodN`
Intel® IPP to accelerate VALARRAY operations

- Added to C++ Standard for performance reason
- C++ standard template (STL) container class for arrays
- C++ `valarray` template consists of array operations that allows for high performance computing; these operations are designed to exploit low level hardware features, for example vectorization

The public member functions of class `valarray` (“what you can do with valarrays”) can be found e.g. at [http://www.aoc.nrao.edu/~tjuerges/ALMA/STL/html/classstd_1_1valarray.html](http://www.aoc.nrao.edu/~tjuerges/ALMA/STL/html/classstd_1_1valarray.html)
Specialized \textit{valarray} Operations

- Intel\textsuperscript{®} C++ Compiler Version 11.0 provides an optimized replacement \textit{valarray} header file
- No source changes required (if using Valarray already)
- Provide parallelism through std headers and templates \texttt{(perf\_header/c++)}
Specialized *valarray* Operations

– Provide seamless use of IPP optimized/threaded libraries through *valarray*
– Provides IPP optimized versions of several *valarray* operations
– User must explicitly add include and library search paths for 11.0
  • Complete driver support will be added in a future version of the Intel C++ compiler (adding an option to use includes and link appropriate libs)
Example for `valarray` usage

```c++
#include <valarray>
...
std::valarray<float> vi(N);
std::valarray<float> va(N);
....
vi = vi + va; // array addition
...
- Default `valarray` operator+ implemented as a loop
- IPP specialized version maps to IPP performance primitive:

<L> = “Installation directory of Linux compiler”
icpc -I<L>/perf_headers/c++ -Iipp_include_dir -c source.cpp
icpc source.o -Lipp_lib_dir -lipps -lippvm -lippcoreem64t
```
Agenda

• Overview
• Intel® Compiler – Features and Q2 2009 Update
• Optimization With Intel Compiler Pro
• Intel® Compiler 11.1 Preview
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• Summary, References and Call to Action
Optimization With the Intel Compilers Professional Edition
Getting Started

With 11.x compiler, ifortvars iccvars takes an argument:
source <install path>/bin/ifortvars.sh intel64
other targets: ia32 ia64 (Itanium)

Documentation:
<install path>/Documentation (11.0)
<install path>/Documentation/en_US (11.1)
other languages coming

ifort --help icc --help icpc --help
Don’t Blow Your Stack

Fortran 90 and OpenMP require a lot of stackspace

Typical failure: “segmentation violation: sigsegv” (linux)

Cause:  Array temporaries (F90) and PRIVATE (OpenMP)

First try:  F90:  -heap-arrays
F90:  -check arg_temp_created
OpenMP:  ulimit –s unlimited (OpenMP)
## General Optimization Options

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Linux*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable optimization</td>
<td>-O0</td>
</tr>
<tr>
<td>Optimize for speed (no code size increase), no SWP</td>
<td>-O1</td>
</tr>
<tr>
<td>Optimize for speed (default)</td>
<td>-O2</td>
</tr>
<tr>
<td>High-level optimizer ( e.g. loop unroll), -ftz (for Itanium)</td>
<td>-O3</td>
</tr>
<tr>
<td>Vectorization –[a]x(sse3</td>
<td>ssse3</td>
</tr>
<tr>
<td>Aggressive optimizations (-ipo, -O3 -no-prec-div -static -xHost)</td>
<td>-fast</td>
</tr>
<tr>
<td>Create symbols for debugging</td>
<td>-g</td>
</tr>
<tr>
<td>Generate assembly files (Linux/Mac also need –save-temps )</td>
<td>-S</td>
</tr>
<tr>
<td>Optimization report generation</td>
<td>/opt-report</td>
</tr>
<tr>
<td>OpenMP 2.5 support</td>
<td>-openmp</td>
</tr>
<tr>
<td>Automatic parallelization for OpenMP* threading</td>
<td>-parallel</td>
</tr>
</tbody>
</table>
Interprocedural Optimization

Extends optimizations across file boundaries

Without IPO

Compile & Optimize → file1.c
Compile & Optimize → file2.c
Compile & Optimize → file3.c
Compile & Optimize → file4.c

With IPO

Compile & Optimize

- ip: Only between modules of one source file
-ipo: Modules of multiple files/whole application

Q2 2009 DPD Tools Training
IPO – A Multi-pass Optimization
A Two-Step Process

Compiling

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac*/Linux*</td>
<td><code>icc -c -ipo main.c func1.c func2.c</code></td>
</tr>
<tr>
<td>Windows*</td>
<td><code>icl -c /Qipo main.c func1.c func2.c</code></td>
</tr>
</tbody>
</table>

Pass 1

Pass 2

virtual .o

Compiling

<table>
<thead>
<tr>
<th>Platform</th>
<th>Command Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac*/Linux*</td>
<td><code>icc -ipo main.o func1.o func2.o</code></td>
</tr>
<tr>
<td>Windows*</td>
<td><code>icl /Qipo main.o func1.o func2.o</code></td>
</tr>
</tbody>
</table>

executable
What you should know about IPO

O2 and O3 activate “almost” file-local IPO (-ip)

- Only a very few, time-consuming IP-optimizations are not done but for most codes, -ip is not adding anything
- Switch –ip-no-inlining disables in-lining

IPO extends compilation time and memory usage

- See compiler manual when running into limitations

In-lining of functions is most important feature of IPO but there is much more

- Inter-procedural constant propagation
- MOD/REF analysis (for dependence analysis)
- Routine attribute propagation
- Dead code elimination
- Induction variable recognition
- …many, many more
Multiple Jobs for IPO: -ipo-jobs<n>

Specifies the number <n> of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).

This option can be affected by the following compiler options:

- -ipo when applications are large enough that the compiler decides to generate multiple object files
- -ipo<m> when m is greater than 1
  - This option causes the compiler to generate <m> separate object files as input to final IPO-linkage phase
- -ipo-separate.
Profile-Guided Optimizations (PGO)

Use execution-time feedback to guide (final) optimization

Helps I-cache, paging, branch-prediction

Enabled optimizations:

- Basic block ordering
- Better register allocation
- Better decision on which functions to inline
- Function ordering
- Switch-statement optimization
PGO Usage: Three Step Process

**Step 1**

**Instrumented Compilation**

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

Instrumented executable: prog.exe

**Step 2**

**Instrumented Execution**

prog.exe (on a typical dataset)

DYN file containing dynamic info: .dyn

**Step 3**

**Feedback Compilation**

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

Merged DYN summary file: .dpi
Delete old dyn files unless you want their info included

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.*
Simple PGO Example: Code Re-Order

for (i=0; i < NUM_BLOCKS; i++)
{
    switch (check3(i))
    {
        case 3: /* 25% */
            x[i] = 3; break;
        case 10: /* 75% */
            x[i] = i+10; break;
        default: /* 0% */
            x[i] = 99; break
    }
}

“Case 10” is moved to the beginning
  - PGO can eliminate most tests&jumps for the common case – less branch mispredicts
What you should know about PGO

Instrumentation run can be up to twice as long
- In-lining disabled, trace calls overhead
- Sometimes trace-files cannot be found
- Looking at right directory?
- Clean exit() call is necessary to dump info
  - Debugger can help / break in PGO trace start/end calls

Benefit depends on control flow structure:

- Significant Benefit
- Little Benefit
Memory Reference Disambiguation

Options/Directives related to Aliasing

- `no_alias_args` (`-alias_args`)  
- `no_ansi_alias` (`-ansi_args`):
  - `fno-alias` (`-falias`): No aliasing in whole program
  - `fno-fnalias` (`-fnalias`): No aliasing within single units
  - `restrict` (`-norestrict`): C99 `-restrict` and `restrict` attribute
    - enables selective pointer disambiguation
  - `safe_cray_ptr`: No aliasing introduced by Cray-pointers
  - `assume dummy_alias`

Related: Switch `-ipo` and directive IFDEP

There are many more options – different for Windows* and Linux*, different for C/C++ and Fortran
Multithreading & OpenMP* Support

Intel Compilers (C/C++ and Fortran) support OpenMP 3.0
- -openmp
- -openmp_report{0|1|2}

Automatic parallelization
- -parallel
- -parthreshold<n>  // control over which loops to parallelize

Large set of environment variables to control run time behavior
- Very important (Intel-specific): KMP_AFFINITY – used to control mapping of OMP-threads to hardware resources (logical/physical cores)
Intel OpenMP* Compatibility Support to OpenMP* of Microsoft and Linux/GCC

New version of Intel OpenMP* RTL released with the Intel 10.1 compiler

- libiomp5
  - New OpenMP RTL shipped with Intel Compiler – compatibility library
- libguide40
  - Intel’s Compiler’s past/current/legacy OpenMP RTL
- Both binary libraries from existing single code base

Multiple compatibility usage modes

- Compile with Intel, link with libiomp5
- Compile with MS*, GNU* compilers, link libiomp5
- Compile part of app with MS, GNU and part with Intel compilers, link libiomp5

The compatibility library is the default in 11.x
Sample: 2x2x2 machine topology (2 packages/sockets, 2 cores, 2 logical threads);
OpenMP thread bindings for

```
KMP_AFFINITY=“granularity=fine,scatter”
```
Static Verifier (SV) of Intel Compiler

New diagnostics system SV has only two specific options; the others are available for /Qdiag too:

For Windows (for Linux and MacOS replace “/Q” by “-“)
- **diag-enable:sv{[1|2|3]}** brief, default, verbose
- **diag-enable:sv-include** analyze include files too
- **diag-file <file>** put results into file <file>
- **diag-disable:{,<num>}** ignore issue by number

(many more diag switches relevant for SV too)

Notes:
Initially, we planned for a totally different naming and types of options for SV – in case you have seen them, forget them now!
Not everything is working perfectly today and documentation (e.g. icc –help) needs to be added/improved
SV Sample: Constant propagation

• This example illustrates a list of constant values propagation and creation of a range value for IF-statement.
• In line 8 the range value is created for variable “b”: ( .. 0.0).
• In line 12 the list of values is propagated for variable “a”: {( .. 0.0), 3.14}.
• Variable “a” is used for “sqrt” and thus should be non-negative. One element of the range list doesn’t fulfill pre-condition and SV issues the following messages:
• t.c(12): warning #12039: possibly wrong value of argument

```c
#include <stdio.h>
#include <math.h>

int main(
    double a,b;
    b = foo();
    if (b < 0)
        a = b;
    else
        a = 3.14;
    printf("%a
    \n",sqrt(a));
    return 0;
} 
```
### Some Intel-Specific Compiler Directives

<table>
<thead>
<tr>
<th>cDECS #pragma</th>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vector/novector</code></td>
<td>IA-32, Intel64</td>
<td>Indicates to the compiler that the loop should (not) be vectorized overriding the compiler’s heuristics.</td>
</tr>
<tr>
<td><code>loop count(n)</code></td>
<td>IA-32, Intel64, IA-64</td>
<td>Place before a loop to communicate the approximate number of iterations the loop will execute. Affects software pipelining, vectorization and other loop transformations.</td>
</tr>
<tr>
<td><code>distribute point</code></td>
<td>IA-32, Intel64, IA-64</td>
<td>Placed before a loop, the compiler will attempt to distribute the loop based on its internal heuristic. Placed within a loop, the compiler will attempt to distribute the loop at the point of the pragma. All loop-carried dependencies will be ignored.</td>
</tr>
<tr>
<td><code>unroll, unroll(n), nounroll</code></td>
<td>IA-32, Intel64, IA-64</td>
<td>Place before an inner loop (ignored on non-inmost loops). #pragma unroll without a count allows the compiler to determine the unroll factor. #pragma unroll(n) tell the compiler to unroll the loop n times. #pragma nounroll is the same as #pragma unroll(0).</td>
</tr>
<tr>
<td><code>loop count n</code></td>
<td>IA-32, Intel64, IA-64</td>
<td>Hint to the compile on expected loop iteration count: (n): always n (n1, n2, n3 …) either n1, n2, n3 … expected minimum count &lt;l&gt;, average count &lt;a&gt; and maximal count &lt;u&gt;</td>
</tr>
<tr>
<td><code>ivdep</code></td>
<td>IA-32, Intel64, IA-64</td>
<td>Place before a loop to control vectorization/software pipelining. The compiler is instructed to ignore “assumed” (not proven) dependencies preventing vectorization/software pipelining. For Itanium: Assume no BACKWARD dependencies, FORWARD loop-carried dependencies still can exist w/o preventing SWP. Use with –ivdep_parallel option to exclude loop-carried dependencies completely (e.g. for indirect addressing).</td>
</tr>
</tbody>
</table>
C/C++ Compiler for Linux*

Main goals/values of Intel C/C++ Linux Compiler:

- Compatible to GCC
- In general much faster than GCC
  - In particular for typical HPC applications
- Provide critical features not offered by GCC like support for latest Intel® processors

Compatibility splits into 3 parts:

- C/C++ source code language acceptance
  - Almost done; missing rest is questionable (e.g. nested function in C supported by GCC)
- Switch-compatibility
  - Achieved for most relevant options
- Object Code interoperability
  - 100% reached today – even for complex C++ and OpenMP*

The ultimate compatibility test: Linux kernel build

- ICC 10.x and kernel 2.6.x: No manual changes to source code required but requires ‘wrapper’ script
C++ Compiler for Linux*

Intel® Compiler 10.x and later use GNU C++ run-time libraries as default
- Until 9.1: –cxxlib-icc to link in Intel (Dinkumare*) provided run-time system; discontinued in 10.0
- Another argument for compatibility!

Intel adapts to new features of latest GCC compilers quickly to guarantee compatibility; examples include:
- MudFlap – to adapt to GCC 4.1
- OpenMP* compatibility library – GCC4.2
- (in 11.0) support of a subset of C++0X-features – GCC4.3

Many new options for enhanced code security like bounds checking (which GCC doesn’t have)

Eclipse IDE / CDT integration
- Adapting to latest Eclipse and CDT releases in each new compiler version
Figure 8. Eclipse-Generated Makefile for mysql Project and the Output of the make Command
Intel® Fortran Compiler: Standard Support

Today (10.1), the Intel® Fortran Compiler fully supports Fortran 95 standard with many features from Fortran 2003 (separate discussion)

It fully supports programs written to the Fortran 90, FORTRAN 77 and FORTRAN IV standards (switches may be needed)

It can check for conformance to Fortran 2003, 95 and 90 (default is 2003 since 10.0)

If Fortran 2003 specifies different behavior than Fortran 95, the F2003 behavior is chosen (CSQRT, for example)

Compiler now is merger of former Intel 7.x compiler and Compaq Visual Fortran 6.6
New Processor Targeting since 11.0
Multiple Key Changes

- New option names
  - Old options are very unintuitive:
    - \([-a]x[K|W|N|B|P|T|O|S|L|H]\)
  - Hard to correlate to instruction set or processor
  - New switch names contain instruction set like \(-xSSE3\)

- New default enables vectorization (SSE2)
  - Faster code from out-of-the-box compilations
  - Non-expert users may never realize
  - Has been default for Intel®64 since the beginning, now extended to IA32 too
  - Explicit switch \(-mia32\) for ancient target processors

- New feature \(-xHost\)
  - Targets the processor on which you compile
Three Sets of Switches to Enable Processor-specific Extensions

1. **Switches –x<EXT> like –xSSE4_1**
   - Imply an Intel processor check
   - Run-time error message at program start when launched on processor without <EXT>

2. **Switches –m<EXT> like –mSSE3**
   - No processor check
   - Illegal instruction fault when launched on processor without <EXT>

3. **Switches –ax<EXT> like –axSSE4_2**
   - Automatic processor dispatch – multiple code paths
   - Processor check only available on Intel processors
   - Non-Intel processors take default path
   - Default path is –mSSE2 but can be modified again by another –x<EXT’> switch
### New Extension Names (Linux)

<table>
<thead>
<tr>
<th>Old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>-xsse *</td>
</tr>
<tr>
<td>-xN</td>
<td>-xsse2</td>
</tr>
<tr>
<td>-xP</td>
<td>-xsse3</td>
</tr>
<tr>
<td>-xT</td>
<td>-xssse3</td>
</tr>
<tr>
<td>-xS</td>
<td>-xsse4_1</td>
</tr>
<tr>
<td>-xH</td>
<td>-xsse4_2</td>
</tr>
<tr>
<td>-xL</td>
<td>-xsse3_atom</td>
</tr>
<tr>
<td>-xB</td>
<td>none</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Old</th>
<th>new</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>-mia32</td>
</tr>
<tr>
<td>-xK</td>
<td>-msse  *</td>
</tr>
<tr>
<td>-xW</td>
<td>-msse2</td>
</tr>
<tr>
<td>-xO</td>
<td>-msse3</td>
</tr>
</tbody>
</table>

-xHost  
Targets the processor on which you compile

* The switch is accepted but same behavior as –mia32
Agenda

• Overview
• Intel® Compiler – New Features
  – OpenMP* 3.0, Intel® Parallel Debugger
  – Support for New Architectures / New Vectorization Switches
  – Others
• Intel® Fortran Compiler 11.1 Preview
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• New Models for Parallel Programming
• Summary, References and Call to Action
Features/Changes for 11.1 Compilers

Linux

• Extended capabilities for Intel® Debugger IDB for multi-threaded applications
  – Shared Data Access Detection
    • Break on Thread Shared Data access
  – Re-entrant Function Detection
• Eclipse Conversion between GCC and ICC projects
• Almost all Fortran 2003 features
• Switch –xAVX to vectorize for future Sandy Bridge 256bit SSE instruction: Intel® Advanced Vector Extension
  – Whatif.intel.com offers AVX emulator for first run-time tests
• HUGE reduction in C++ compilation time ( !! )
New Fortran 2003 Features in 11.0

- **ENUMERATOR**
  a type declaration statement which specifies a set of named integer constants (enumerators) that are interoperable with C

- **MAX/MIN/MAXVAL/MINVAL/MAXLOC/MINLOC**
  these intrinsic functions can take character arguments now too

- **IEEE Floating Point Exception Handling**
  rounding mode changes, exception handling, HW conformance tests, FP range check

- **ASSOCIATE**
  allows a short name to reference expressions or parts of variables

- **Procedure Pointers & Abstract Interface**
  abstract interface, procedure declaration statement, procedure pointer and passed-object dummy arguments

- **Structure Constructor Change**
  component names and optional components with default initialization

- **Array Constructor Changes**
  different character lengths and type specifications to array constructors

- **ALLOCATE extensions**
  add ALLOCATE (N) where N is a scalar of intrinsic or derived type,
  ALLOCATE (character(len=N):: C) where C is declared with new syntax
  CHARACTER (LEN=:) :: C with deferred length and ALLOCATE (type (DT_TYPE ) :: D) where D is of derived type DT_TYPE
```fortran
USE, INTRINSIC :: IEEE_EXCEPTIONS
USE, INTRINSIC :: IEEE_FEATURES, ONLY: IEEE_INVALID_FLAG

TYPE(IEEE_STATUS_TYPE) STATUS_VALUE
LOGICAL, DIMENSION(3) :: FLAG_VALUE

CALL IEEE_GET_STATUS(STATUS_VALUE)
CALL IEEE_SET_HALTING_MODE(IEEE_USUAL,.FALSE.) ! in case the default is to halt on exceptions
CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)  ! First try the "fast" algorithm for inverting a matrix:
MATRIX1 = FAST_INV(MATRIX) ! This shall not alter MATRIX.
CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
IF (ANY(FLAG_VALUE)) THEN          ! "Fast" algorithm failed; try "slow" one:
    CALL IEEE_SET_FLAG(IEEE_USUAL,.FALSE.)
    MATRIX1 = SLOW_INV(MATRIX)
    CALL IEEE_GET_FLAG(IEEE_USUAL,FLAG_VALUE)
    IF (ANY(FLAG_VALUE)) THEN
        WRITE (*, *) 'Cannot invert matrix'
        STOP
    END IF
END IF
CALL IEEE_SET_STATUS(STATUS_VALUE)
```
New F2003 Features in 11.1

• Object oriented features
  – type extension
  – polymorphic entities
  – inheritance association
  – select type construct
  – deferred bindings (uses abstract types)
  – type inquiry intrinsic functions
• Type-bound procedures and operators
• Overriding
• Abstract Type
• Deferred-length character
• Selective accessibility of components
• Some miscellaneous additions
Sample Feature: CLASS Declaration

- CLASS is a variant of TYPE that can match a type and any extension of that type
- Valid for pointer, allocatable and dummy argument only

```fortran
TYPE :: point
    REAL :: x
    REAL :: y
END TYPE point

TYPE, EXTENDS(point) :: point_3D ! In 11.0
    REAL :: z
END TYPE point

TYPE(point) :: p1 ! Matches only point
CLASS(point) :: p2 ! Matches point and point_3D
```
F2003: What’s Missing after 11.1 ?

- FINAL and GENERIC type-bound procedures
- User-defined derived type I/O
- Parameterized derived types

Fortran 2003 References:

Agenda

• Overview

• Intel® Compiler – New Features
  – OpenMP® 3.0, Intel® Parallel Debugger
  – Support for New Architectures / New Vectorization Switches
  – Others

• Intel® Fortran Compiler Update

• Intel® Libraries Update

• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture

• New Models for Parallel Programming

• Summary, References and Call to Action
## Release Components

<table>
<thead>
<tr>
<th></th>
<th>Intel® Compiler</th>
<th>Intel® MKL</th>
<th>Intel® IPP</th>
<th>Intel® TBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pro Edition 10.1</td>
<td>10.1</td>
<td>10.0</td>
<td>5.3</td>
<td>2.0</td>
</tr>
<tr>
<td>Pro Edition 11.0</td>
<td>11.0</td>
<td>10.1</td>
<td>6.0</td>
<td>2.1</td>
</tr>
<tr>
<td>Released Nov 08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Edition 11.1</td>
<td>11.1</td>
<td>10.2</td>
<td>6.1</td>
<td>2.1 (Update)</td>
</tr>
<tr>
<td>In Beta today; release planned for end of Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.*
**Intel® MKL 10.1 – New Features**

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem Core Processor Optimization</td>
</tr>
<tr>
<td>Direct Sparse Solver extensions</td>
</tr>
<tr>
<td>Documentation Improvements</td>
</tr>
<tr>
<td>FFT Extensions</td>
</tr>
<tr>
<td>Improved IDE Integration</td>
</tr>
<tr>
<td>Internationalization</td>
</tr>
<tr>
<td>Add LAPACK Callback functions</td>
</tr>
<tr>
<td>Support Head DUO OMP library as default threading</td>
</tr>
<tr>
<td>VML extensions</td>
</tr>
<tr>
<td>Sparse BLAS extensions</td>
</tr>
<tr>
<td>Consistency with C++ complex standard library</td>
</tr>
</tbody>
</table>
MKL 10.1 Nehalem Perf. Improvement Samples
Comparing Release 10.0 to 10.1 (new with 11.0 compilers)

- **BLAS**
  - 50% improvement for SGEMM
  - 30% improvement for right-side cases of DTRSM
- **VML and VSL**
  - Up to 17% improvement for the following VML functions: Asin, Asinh, Acos, Acosh, Atan, Atan2, Atanh, Cbrt, CIS, Cos, Cosh, Conj, Div, ErfInv, Exp, Hypot, Inv, InvCbrt, InvSqrt, Ln, Log10, MulByConj, Sin, SinCos, Sinh, Sqrt, Tanh
  - Up to 67% improvement for uniform random number generation
  - Up to 10% improvement for VSL distribution generators based on Wichmann-Hill, Sobol, and Niederreiter BRNGs (64-bit only)

New Features (coming in Comp. Pro 11.1)

• Improved Functionality
  – BLAS L1/L1 threaded
  – PARDISO single precision support
  – access to forwd/backwd solve results
  – LAPACK progress routines
  – VML new functions
  – PARDISO out-of-core performance improved
  – LINPACK improvements

• Improved Interfaces
  – Fortran95 BLAS and LAPACK interfaces pre-built for Intel compiler
  – FFTW interface now in integrated
  – Sparse Matrix format conversion routines
Intel® Math Kernel Library 10.2

- Optimized versions of the following functions available for Intel® Advanced Vector Extensions (Intel® AVX)
  - BLAS: DGEMM
  - FFTs
  - VML: exp, log, and pow
- mkl_enable_instructions() function to access AVX extensions.
- Details in users' guide and reference manual.
- Whitepaper with examples and information for accessing emulator
<table>
<thead>
<tr>
<th>Component</th>
<th>Intel® Compiler</th>
<th>Intel® MKL</th>
<th>Intel® IPP</th>
<th>Intel® TBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pro Edition 10.1</td>
<td>10.1</td>
<td>10.0</td>
<td>5.3</td>
<td>2.0</td>
</tr>
<tr>
<td>Pro Edition 11.0</td>
<td>11.0</td>
<td>10.1</td>
<td>6.0</td>
<td>2.1</td>
</tr>
<tr>
<td>Released Nov 08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Professional Edition 11.1</td>
<td>11.1</td>
<td>10.2</td>
<td>6.1</td>
<td>2.1</td>
</tr>
<tr>
<td>In Beta today; release planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for end of Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intel IPP 6.0 Features – Highlights

• Processor optimizations and Support
  – Intel® Atom™ Processor family
  – Intel® Core™ i7 (Nehalem) optimizations

• New higher-level libraries for lossless data compression (bzip2, zlib, lzo, gzip, etc)

• New Deferred Mode Image Processing Layer (DMIP) for large images

• Threaded Static Libraries Support

• Unified Image codec (UIC) to integrate all image related codecs into one interface (Sample)

• New cryptography interfaces for simplified integration into OpenBSD* and Windows* cryptography frameworks

• Full integration of the Intel® Compatibility OpenMP* run-time library for greater Windows/Linux* cross-platform compatibility

• New Data Integrity Functions based on operations over finite fields for error-correcting coding
## Intel® IPP Threading Sample: Compression

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Data Compression</th>
<th>Threading Technology:</th>
<th>Threading Mechanism:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ipp-zlib</td>
<td>Compression</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Decompression</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Ipp-gzip</td>
<td>Compression</td>
<td>- Multi-file threading&lt;br&gt;- Multi-chunk threading</td>
<td>- OpenMP for Windows&lt;br&gt;- OpenMP and pthreads for Linux</td>
</tr>
<tr>
<td></td>
<td>Decompression</td>
<td>- Multi-file threading&lt;br&gt;- Multi-chunk threading</td>
<td>- OpenMP for Windows&lt;br&gt;- OpenMP and pthreads for Linux</td>
</tr>
<tr>
<td>Ipp-bzip2</td>
<td>Compression</td>
<td>Multi-chunk threading</td>
<td>OpenMP for Linux &amp; Windows</td>
</tr>
<tr>
<td></td>
<td>Decompression</td>
<td>Multi-chunk threading</td>
<td>OpenMP for Linux &amp; Windows</td>
</tr>
<tr>
<td>Ipp-lzo</td>
<td>Compression</td>
<td>Multi-chunk threading</td>
<td>OpenMP for Linux &amp; Windows</td>
</tr>
<tr>
<td></td>
<td>Decompression</td>
<td>Multi-chunk threading</td>
<td>OpenMP for Linux &amp; Windows</td>
</tr>
</tbody>
</table>

*Complex but transparent threading implementation – easiest way to multi-threading*
Intel IPP 6.1 Highlights (CompPro 11.1)

- **Support Intel® Advanced Vector Extensions (Intel® AVX)**
- **IntelliSense* Capability**
- **Unified Image Classes (UIC) implementation and new features to support DXT1, DXT3, DXT5 texture compression**
- **IPP-Crypto- support to RSA_SSA1.5 nor RSA_PKCSv1.5**
- **New design for data compression deflate/inflate APIs to better support compatibility with A Support (HD Photo): IPP PCT Functions**
- **New chm format in manuals**
- **Advanced lighting functionality**
- **3D Geometric Transforms: Super Sampling method**
- **Noise Detection Function**
Intel® Threading Building Blocks

- **Benefit from 30 years of threading best known methods** to simplify implementation of scalable, high performance threaded applications
  - Make multi-threaded application development practical
    - A C++ template library that uses familiar task patterns, not threads
    - A high level abstraction requiring less code for threading without sacrificing performance
    - Reduce maintenance as number of cores increase
  - Maximize application performance
    - Appropriately scales to the number of cores available
  - Utilize one threading library for 32 & 64 bit Windows*, Linux* and Mac OS* X
    - The thread library API is portable across Linux, Windows, or Mac OS platforms
    - Works with all C++ compilers (e.g., Microsoft, GNU and Intel)
  - You have a choice
    - Commercial version that includes support
    - Open source version
      - Available at [www.opentbborg](http://www.opentbborg)
      - Book available from O’Reilly publishing

```
TBB helped KnowledgeMiner achieve speeds 3x faster on a 8 core system. In addition, a strict redesign of KnowledgeMiner for parallel computing is giving a total speedup over the previous version 400x. This astonishing change in speed allows KnowledgeMiner to operate in almost real time something we didn’t previously think was possible.
```

Frank Lemke  
Founder and President,  
KnowledgeMiner Software

---

<table>
<thead>
<tr>
<th>Windows*</th>
<th>Linux*</th>
<th>Mac OS* X</th>
<th>IA32</th>
<th>Intel64</th>
<th>IA64</th>
<th>Multicore</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.
**Intel® Threading Building Blocks 2.1**

Improved/Enhanced/New Features in **BOLD**

**Generic Parallel Algorithms**
- `parallel_for`
- `parallel_reduce`
- `parallel_do (new)`
- `pipeline`
- `parallel_sort`
- `parallel_scan`

**Concurrent Containers**
- `concurrent_hash_map`
- `concurrent_queue`
- `concurrent_vector`

**Task scheduler**

**Synchronization Primitives**
- `atomic operations`
- `scoped locks`

**Miscellaneous**
- `tick_count`
- `tbb_thread`

**Memory Allocation**
- `tbb_allocator (new)`
- `cache_aligned_allocator`
- `scalable_allocator`

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.
Agenda

• Overview
• Intel® Compiler – New Features
  – OpenMP* 3.0, Intel® Parallel Debugger
  – Support for New Architectures / New Vectorization Switches
  – Others
• Intel® Fortran Compiler Update
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• Summary, References and Call to Action
Nehalem Optimizations

- Nehalem (Intel® Core i7 and Intel® Xeon® 5500 Processor) changes relevant for SW tuning
  - New instruction set extension SSE4.2
  - Removal of most data access alignment restrictions
  - Enhanced and improved Core™ u-architecture
    - Larger internal buffers like Memory Store Buffer
    - Improved Loop Stream Detector (LSD)
    - Extended Macro-Fusion
  - Some faster instructions like REP
  - Simultaneous Multi-Threading
  - Cache hierarchy extended to three levels
  - QPI – Intel® Quick Path Technology
  - Non-uniform memory access (NUMA) for multiple socket CPUs
Optimization Guidelines For Nehalem

- Many new features introduced that you get for free
  - Better branch prediction + faster miss-prediction correction
  - Improvements on unaligned loads + cache-line splits
  - Improvements on store forwarding
  - Memory bandwidth increase
  - Reduced memory latency
  - Etc...

- No large differences in tuning guidelines relevant for Intel® Core™ 2 processor architecture
  - For details see Intel® 64 and IA-32 Architectures Optimization Reference Manual:
    http://www.intel.com/products/processor/manuals/
Compiler Pro and Nehalem

• Since 11.0 release, compiler support “SSE4_2” extension to specifically tune for Nehalem
  Linux: -xsse4_2, -axsse4_2, -msse4_2
• Specific compiler changes
  – Automatic vectorization makes use of SSE4.2 set
  – Support for new intrinsics for manual coding
    • File nmmintrin.h contains declarations
  – Support for manual CPU dispatching
    • __declspec(cpu_specific(core_i7_sse4_2))
  – Code generation exploits benefits of architectural changes
• Libraries Intel® MKL and IPP
  – Performance critical routines specifically tuned for Nehalem architecture (instruction set, memory hierarchy incl. NUMA, SMT, u-architecture)

Evolving Process: New CompPro releases will extend features
Intel® SSE4.2 ISA Extensions

Accelerated String and Text Processing
- Faster string/XML parsing
- Faster search and pattern matching
- 4 new instructions

Accelerated Searching & Pattern Recognition of Large Data Sets
- Improved performance for Genome Mining, Handwriting recognition. Fast Hamming distance / Population count

New Communications Capabilities
- Hardware based CRC instruction
- Accelerated Network attached storage
- Improved power efficiency for Software I-SCSI, RDMA, and SCTP

SSE4.2: In total 7 new instructions to accelerate string processing & specific application kernels as well as one generic compare instruction filling a gap
### PCMPGTQ - Vectorization

```c
__int64 a[N], b[N], c[N], d[N], e[N];
for (i = 0; i < N; i++)
a[i] = (b[i] > c[i]) ? d[i] : e[i];
```

**Before: No Vectorization Possible**

<table>
<thead>
<tr>
<th></th>
<th>New Instructions allow for vectorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>xorl %eax, %eax</td>
<td>xorl %eax, %eax</td>
</tr>
<tr>
<td>..B2.2:</td>
<td>..B2.2:</td>
</tr>
<tr>
<td>movq b(,%rax,8), %rdx</td>
<td>movdqa b(,%rax,8), %xmm0</td>
</tr>
<tr>
<td>cmpq c(,%rax,8), %rdx</td>
<td>pcmpgtq c(,%rax,8), %xmm0</td>
</tr>
<tr>
<td>jle ..B2.4</td>
<td>movdqa e(,%rax,8), %xmm1</td>
</tr>
<tr>
<td>..B2.3:</td>
<td>pblendvb %xmm0, d(,%rax,8), %xmm1</td>
</tr>
<tr>
<td>movq d(,%rax,8), %rdx</td>
<td>movdqa %xmm1, a(,%rax,8)</td>
</tr>
<tr>
<td>jmp ..B2.5</td>
<td>addq $2, %rax</td>
</tr>
<tr>
<td>..B2.4:</td>
<td>cmpq $1024, %rax</td>
</tr>
<tr>
<td>movq e(,%rax,8), %rdx</td>
<td>jl ..B2.2</td>
</tr>
<tr>
<td>..B2.5:</td>
<td></td>
</tr>
<tr>
<td>movq %rdx, a(,%rax,8)</td>
<td></td>
</tr>
<tr>
<td>incq %rax</td>
<td></td>
</tr>
<tr>
<td>cmpq $1024, %rax</td>
<td></td>
</tr>
<tr>
<td>jl ..B2.2</td>
<td></td>
</tr>
</tbody>
</table>

**Speedups**
- Intel® 64: 1.5x
- IA-32: 3.0x

*As measured in Intel labs*
STTNI – Faster String Processing

STTNI == String and Text New Instructions
  • Operates on strings of bytes (8bit) or words (16bit)
    – String Compare
    – Detect possible substrings within a string
    – Find occurrences from a set of values
    – Find occurrences from a set of ranges

• Useful for lexing, tokenizing, RegEx evaluation, virus scanning, intrusion (IPS/IDS)
  – State machines/automata processing 8 or 16 bit input streams

• Targets re-entrant states and chains in state machines
STTNI – String Length Sample

- Sample: `pcmpistri` : Partially inlined `strlen` implementation
  - Avoids call overhead for short strings (common case)
  - Avoids the excessive code bloat from fully inlining

```
mov       ecx, edx
and       edx, 0x0FFFFFFF0
pxor      xmm0, xmm0
pcmpeqb   xmm0, XMMWORD PTR [edx]
pmovmskb  eax, xmm0
and       ecx, 0xF
shr       eax, cl
bsf       eax, eax
jne       ..L1

mov       eax, edx
add       edx, ecx
call      __intel_sse4_strlen

..L1:
```

```
__intel_sse4_strlen:
  add       eax, 16
  movdqa    xmm0, XMMWORD PTR [eax]
  pcmpistri xmm0, xmm0, 58
  jae       __intel_sse4_strlen

  sub       ecx, edx
  add       eax, ecx
  ret
```
STRLEN() Acceleration

*As measured in Intel labs

Over 5x speedup* on long strings

*As measured in Intel labs
Unaligned Loads/Stores

• Unaligned loads are as fast as aligned loads
• Optimized accesses that span two cache-lines
• Generating misaligned references less of a concern
  – One instruction can replace sequences of up to 7
  – Fewer instructions
  – Less register pressure
• Increased opportunities for several optimizations
  – Vectorization
  – memcpy/memset
  – Dynamic Stack alignment less necessary for 32-bit stacks
Unaligned Loads/Stores - Sample

for (i = 0; i < NUM; i++)
  dst[i] = src1[i] * src2[i+1] + src3[i+1];

Code1: no Core i7 optimizations (16 instructions per loop iteration):

  Code2: generate more 16-bytes loads (13 insts per iteration):

  loop_label:
  movups xmm1, XMMWORD PTR [_src2+4+eax*4]
  movsd xmm4, QWORD PTR [_src2+20+eax*4]
  movss xmm2, DWORD PTR [_src2+28+eax*4]
  movhps xmm2, QWORD PTR [_src2+32+eax*4]
  ...
  shufps xmm4, xmm2, 132
  ...
  add eax, 8
  cmp eax, 1024
  jb loop_label

*As measured in Intel labs

~11% Speedup*
Unaligned Loads/Stores Performance

- **CPU2000 Estimated Data**
  - No performance regressions
  - 168.wupwise +8%
  - 172.mgrid +21%
  - 178.galgel +3%
  - 301.apsi +5%
  - **Overall fp Geomean** +2.78%

- **CPU2006 Estimated Data**
  - No performance regressions
  - 436.cactusADM +11%
  - 437.leslie3d +9%
  - 454.calculix +8%
  - 459.GemsFDTD +12%
  - **Overall fp Geomean** +2.6%

- Source: Intel. Data estimated based on measurements on preproduction hardware; per SPEC* Run Rules section 4

---

**No more split sequences**

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.
Non-Uniform Memory Access (NUMA)

- Expect most Intel® Xeon® 5500 processor platforms to use NUMA
- Locality matters
  - Remote memory access latency ~1.7x than local memory
  - Local memory bandwidth can be up to 2x greater than remote
- Operating systems differ in allocation strategies + APIs

**Diagram:**

- Local Memory Access
- Remote Memory Access

**Legend:**

- DRAM
- CPU0
- CPU1
- Intel® QPI = Intel® QuickPath Interconnect

Q2 2009 DPD Tools Training

© 2009, Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.
Nehalem
OpenMP Considerations
Using KMP_AFFINITY

KMP_AFFINITY environment variable

- Available on Linux and Windows. (Mac OS* X as of 10.5 does not support thread pinning)
- Hierarchy: Machine/Node, Socket(s), Core(s), SMT/Thread context
KMP_AFFINITY Environment Variable

The “OS Processor” (/var/cpuinfo) is a leaf node in the tree

KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]

Most Common KMP_AFFINITY Settings:

- KMP_AFFINITY=scatter
  - Consecutive threads on alternating sockets
  - Use when no sharing of thread data

- KMP_AFFINITY=compact,1
  - Consecutive threads on different physical cores on the same socket
  - Use when thread share data

- KMP_AFFINITY=disable if 3rd party affinity tools used, eg dplace
KMP_AFFINITY Environment Variable

KMP_AFFINITY=[<modifier>,...] <type>[,<permute>][,<offset>]

Modifier:

- `noverbose` (default), verbose – information on sockets, cores, thread contexts

- `granularity` = [ [thread | fine] | core (default) ] – determines if following binding cares about thread contexts. “core” lumps the core thread context(s) into a set, don’t care if it’s on one or the other or both thread contexts. “fine” allows bindings to exact OS “processor”
KMP_AFFINITY Environment Variables

KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]

**<type>**

**type = compact**
assigns the OpenMP thread \( n + 1 \) to a free thread context as close as possible to the thread context where the \( n \) OpenMP thread was placed.

**type = scatter**
Distributes the threads as evenly as possible across the entire system.
scatter is the opposite of compact

**type = disabled**
Completely disables the thread affinity interfaces.

**type = explicit**
Specifying explicit assigns OpenMP threads to a list of OS proc IDs that have been explicitly specified by using the proclist= modifier, which is required for this affinity type.
**KMP_AFFINITY=fine,scatter**

- **Order (OpenMP Global Thread ID)**

  Sample: 2x2x2 machine topology (2 packages/sockets, 2 cores, 2 logical threads);
  OpenMP thread bindings for

  \[ \text{KMP_AFFINITY} = \text{”granularity=fine,scatter”} \]
KMP\_AFFINITY=fine,compact

Order (OpenMP Global Thread ID)

Sample: 2x2x2 machine topology (2 packages/sockets, 2 cores, 2 logical threads);
OpenMP thread bindings for

KMP\_AFFINITY=“granularity=fine,compact”
permute and offset combinations
Types compact and scatter, permute and offset are allowed; however, if you specify only one integer, the compiler interprets the value as a permute specifier. Both permute and offset default to 0.

The permute specifier controls which levels are most significant, from the bottom up. Zero is the leaf level (thread context), 1 next level ‘up’ (core), etc. The root node of the tree is not considered a separate level for the sort operations.

The offset specifier indicates the starting position for thread assignment.
KMP\_AFFINITY=compact,1

Order (OpenMP Global Thread ID)

Sample: 2x2x2 machine topology (2 packages/sockets, 2 cores, 2 logical threads); OpenMP thread bindings for

\texttt{KMP\_AFFINITY=“granularity=fine,compact,permute=1”}
Confirming Your Affinity

Add ‘verbose’ to your KMP_AFFINITY modifier and run

```bash
rwgreen@dpd21:~> export KMP_AFFINITY=verbose,scatter
rwgreen@dpd21:~> ./pi_omp
starting
```

```bash
OMP: Info #149: KMP_AFFINITY: Affinity capable, using global cpuid instr info
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: {0,1,2,3}
OMP: Info #156: KMP_AFFINITY: 4 available OS procs
OMP: Info #157: KMP_AFFINITY: Uniform topology
OMP: Info #159: KMP_AFFINITY: 2 packages x 2 cores/pkg x 1 threads/core (4 total cores)
OMP: Info #160: KMP_AFFINITY: OS proc to physical thread map ([] => level not in map):
OMP: Info #168: KMP_AFFINITY: OS proc 0 maps to package 0 core 0 [thread 0]
OMP: Info #168: KMP_AFFINITY: OS proc 2 maps to package 0 core 1 [thread 0]
OMP: Info #168: KMP_AFFINITY: OS proc 1 maps to package 3 core 0 [thread 0]
OMP: Info #168: KMP_AFFINITY: OS proc 3 maps to package 3 core 1 [thread 0]
OMP: Info #147: KMP_AFFINITY: Internal thread 0 bound to OS proc set {0}
OMP: Info #147: KMP_AFFINITY: Internal thread 1 bound to OS proc set {1}
OMP: Info #147: KMP_AFFINITY: Internal thread 2 bound to OS proc set {2}
OMP: Info #147: KMP_AFFINITY: Internal thread 3 bound to OS proc set {3}
```
Building Affinity Into the Executable

Coming in 11.1 Update compiler

-par-affinity==[modifier,...]type[,permute][,offset]

Sets affinity of executable, ignores KMP_AFFINITY

-par-num-threads=n
  sets number of threads for OMP parallel regions, ignores OMP_NUM_THREADS
Operating System Differences

• Operating systems allocate data differently
• Linux*
  – Malloc reserves the memory
  – Assigns the physical page when data touched
• Microsoft Windows*
  – Malloc assigns the physical page on allocation
  – This default allocation policy is not NUMA friendly
• Microsoft Windows has NUMA Friendly API’s
  • VirtualAlloc reserves memory (like malloc on Linux*)
    – Physical pages assigned at first use

• For more details:
  http://www.opensolaris.org/os/community/performance/numa/
Shared Memory Threading Example: TRIAD

Parallelized time consuming hotspot “TRIAD” using OpenMP

```c
main() {
  ...
  #pragma omp parallel
  {
    //Parallelized TRIAD loop...
    #pragma omp parallel for private(j)
    for (j=0; j<N; j++)
      a[j] = b[j]+scalar*c[j];
  } //end omp parallel
  ...
} //end main
```

Parallelizing hotspots may not be sufficient for NUMA
Shared Memory Threading Example – II (Linux*)

`KMP_AFFINITY=compact,0,verbose`

```c
main() {
    ... 
    #pragma omp parallel
    {
        #pragma omp for private(i)
        for(i=0; i<N; i++)
            { a[i] = 10.0; b[i] = 10.0; c[i] = 10.0; }
    } //end omp parallel ...
    ... 
    //Parallelized TRIAD loop...
    #pragma omp parallel for private(j)
    for (j=0; j<N; j++)
        a[j] = b[j] + scalar*c[j];
} //end main
```

Each thread initializes its data pinning the pages to local memory

Environment variable to pin affinity

Same thread that initialized data uses data
References for KMP_AFFINITY

Documentation set,
“Intel Fortran Compiler User Guide”
Optimizing Applications
Using Parallelism: OpenMP Support
Libraries, Directives, Clauses, and Environmental Variables
OpenMP Library Support
Thread Affinity Interface

Or search for “AFFINITY” in the docs
Agenda

• Overview
• Intel® Compiler – New Features
  – OpenMP® 3.0, Intel® Parallel Debugger
  – Support for New Architectures / New Vectorization Switches
  – Others
• Intel® Fortran Compiler Update
• Intel® Libraries Update
• Using Intel® Compiler Pro Edition to Tune for Nehalem Architecture
• Summary, References and Call to Action
Summary & Call to Action

• Intel® Compiler Professional Edition Version 11.0 provides best of class
  – C++ and Fortran Compilers
  – Performance Libraries Intel® IPP and Intel® MKL
  – Intel® Threading Building Blocks

• Please contact presenter to join beta program for 11.1 compiler
  – Added Nehalem support
  – Considerable reduction in compile time
  – Enhanced debugger for Linux*
References

- Intel® User Forums for all developer tools, for HPC on processors from Intel, ...

- Intel® Compiler Professional Edition product page

- Intel® 64 and IA-32 Architectures Optimization Reference manual
  http://download.intel.com/design/processor/manuals/248966.pdf