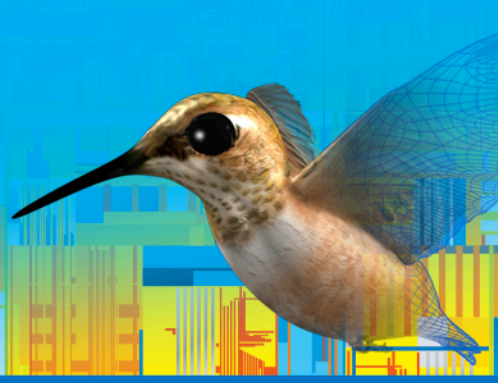
Composer XE 2013 SP1

Getting Started

Xeon Phi edition

Fall 2013



Code the Future

OPTIMIZATION NOTICE

Agenda

New User to Intel Compilers

What is Xeon Phi

Compiling for Xeon phi

Vectorize your code

Tips and tricks

Code the Future

What is Xeon Phi

The Intel® Xeon Phi™ Coprocessor has up to 61 in-order Intel® MIC Architecture processor cores running at
1GHz (up to 1.3GHz).

The Intel® MIC Architecture is based on the x86 ISA, extended with 64-bit addressing and new 512-bit wide SIMD vector instructions and registers.

Each core supports 4 hardware threads

Code the Future

OPTIMIZATION NOTICE

Why Use Intel Compilers?

Compatibility

- Platforms: Source and binary compatible with
  - Visual C++ 2008/2010/2012/2013 on Windows*
  - gcc 4.1 ~ gcc 4.8on Linux*
  - Xcode* 4.6 or 5.0 on OS X*
- ANSI C/C++ and OpenMP* compliance:
  - ISO/IEC 9899:1990 for C language
  - ISO/IEC 9899:1999 for C99
  - C++ ISO/IEC 14882:2011 for C++11
  - Partial support of OpenMP* 4.0

Performance

- Industry leading optimization technologies: auto-vectorization, PGO, IPO, processor targeting optimization
- Outstanding performance on Intel® architecture processors
- Performance libraries: Intel® IPP, Intel® MKL and Intel® TBB

Support

- World class support with secure, web-based, engineer-to-engineer support through Intel Premier Support
- Community based forum support from technical experts around the world

Code the Future

OPTIMIZATION NOTICE

Why Use Intel® Compilers?

Parallelism

Numerous tools to enable parallelism

Vector Parallelism

- Automatic Vectorization
- Vector statements (Intel® Cilk Plus)
- Lower level SIMD (pragmas, intrinsic functions)

Task Parallelism

- Language extensions (Intel® Cilk Plus)
- C++ Task Libraries (Intel® TBB)
- Automatic Parallelism
- GAP – let the compiler help you restructure code for more parallelism opportunities

Multi-threaded Performance Libraries – Intel® MKL, Intel® IPP

Code the Future

Why Use Intel® Compilers?
Performance

Our goal is performance

Performance to be gained in a variety of ways:

- The future is Multi-core (and the future is now!)
- New instructions enable new opportunities (SSE, AVX, AVX2)
- Micro architectural improvements

Intel Compilers Support the latest Features

- Be on the cutting edge of new performance features
  - Latest Instructions
  - Code generation tuned for latest microarchitecture

Highly Optimized libraries

- MKL – Math functions (BLAS, FFT, LAPACK, &c)
- IPP – (compression, video encoding, image processing &c)

Code the Future

Linux: Basic Compiler Usage

source <installdir>/bin/[compilervars.sh | compilervars.csh] [intel64 | ia32]

- Sets environment vars for compiler, libraries, headers, etc.

Compiler drivers are ‘ifort’, ‘icc’ for C, and ‘icpc’ for C++

“-O” switches compatible, but not identical to gcc

- -O2 default optimization level (gcc default is –O0)
- -O* doesn’t imply the same set of opts for gcc and Intel, but similar concepts, -O0 for debugging, -O2 for default, -O3 for more advanced optimizations

icc –help, icpc -help or ifort -help provides extensive list

Intel debugger IDB or Intel-provided GDB (extended)

- Linux: ‘idbc’ command line, ‘idb’ X11 GUI

Code the Future

Compiling for Xeon Phi

- Native Mode
  - Easiest to start
  - Add the –mmic flag to the compiler
- Offload
  - No need for any extra flags
  - Specify sections of code to be run on Xeon Phi through pragmas

Code the Future

Compiling for Xeon Phi

- Native Mode

  icpc -mmic -vec-report3 -openmp omp_native.cpp

```
omp_native.cpp(126): (col. 3) remark: loop was not vectorized: not inner loop
omp_native.cpp(126): (col. 3) remark: loop was not vectorized: not inner loop
omp_native.cpp(52): (col. 4) remark: LOOP WAS VECTORIZED
omp_native.cpp(52): (col. 4) remark: PEEL LOOP WAS VECTORIZED
omp_native.cpp(52): (col. 4) remark: REMAINDER LOOP WAS VECTORIZED
omp_native.cpp(51): (col. 3) remark: loop was not vectorized: not inner loop
omp_native.cpp(50): (col. 2) remark: loop was not vectorized: not inner loop
```

Code the Future

Compiling for Xeon Phi

- Native Mode

  Copy compiled binary over to Xeon Phi

  sudo scp a.out mic0:/tmp

  Copy over libiomp5.so as well

  sudo scp /opt/intel/lib/mic/libiomp5.so mic0:/tmp

  sudo ssh mic0

Code the Future

Compiling for Xeon Phi

- Native Mode

  Run a.out

  Fails because it can't find the openmp library

  export LD_LIBRARY_PATH=/tmp

  Depending on what libraries you use you will need to copy those into your path as well.

Code the Future

Compiling for Xeon Phi

- Offload mode

  Useful for complex applications where you only need heavy compute during certain times.

  Can control what gets run on the coprocessor.

Code the Future

Compiling for Xeon Phi

- Offload mode

  icpc –openmp –O3 –xhost offload.cpp

  No need copy over files to the coprocessor.

  Runs on host.

Code the Future

Compiling for Xeon Phi

- Offload mode

```
#pragma offload target(mic){
/* code in here runs on xeon phi
It will run on the host if there is no
coprocessor available
*/
}
```

Code the Future

Profiling
Function level and/or Loop-level

Code the Future

Function and Loop Profiler

Identify Time Consuming Functions and Loops

Compiler switch:
-profile-functions /Qprofile-functions,
Insert instrumentation calls on function entry and exit points to collect the cycles spent within the function.

Compiler switch:
-profile-loops= <inner|outer|all>
/Qprofile-loops=<inner|outer|all>

Insert instrumentation calls for function entry and exit points as well as the instrumentation before and after instrument able loops of the type listed as the option's argument.

Loop Profiler switches trigger generation of text (.dump) and XML (.xml) output files

Invocation of XML on command line:

```
java -jar loopprofviewer.jar <xml
datafile>
```

Code the Future

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

Code the Future

OPTIMIZATION NOTICE

Loop Profiler Data Viewer GUI

Loop Profile Viewer: c:\3\loop3_sample\loop_prof_1258065047.xml

File View Filter Help

Function Profile

Function	Function file:line	Time	% Time	Self time	% Self time	Call Count	% Time in Loops
_main	spec.c:286	33,737,882,427	99.80	52,724,829	0.16	1	0.09
_compressStream	bzip2.c:440	22,141,802,790	65.50	37,645,635	0.11	3	0.00
_ha		22,072,329,981	65.29	111,073,801	0.33		
_BZ		21,291,282,108	62.98	382,054	0.00		
_BZ		20,773,546,2	61.45	805,260	0.00		
_ur		11,543,35	34.15	1,509,243	0.00		
_BZ		11,519 737	34.08	2,278,698	0.01		
_BZ		11,4 1,637	33.74	69,950,540	0.21		
_mainSort	blocksort.c:805	1 841,172	33.53	1,090,262,703	3.23	25	3.20
_BZ2_decompress	decompress.c:147	171,252,448	33.05	10,916,277,582	32.29	1,177	13.81
_mainQSort3	blocksort.c:676	0,237,947,453	30.28	2,203,239,483	6.52	298,338	2.97
_mainSimpleSort	bloc		22.99	3,978,755,360			3.62
_sendMTFValues	com		16.62	3,149,709,581			0.67
_generateMTFValues	com		12.62	3,565,312,664			6.96
_mainGtU	bloc		8.55	2,388,612,638			1.79
_bsW	com		6.52	1,080,193,267			1.50
_BZ2_bzWriteClose64@28	bzlib		3.88	49,713	0.00	3	0.00
_copy_input_until_stop	bzlib.c:347	668,863,836		667,041,514	1.97	3,172	0.36
_unRLE_obuf_to_output_FAST	bzlib.c:594	337,105,368		336,452,554	1.00	3,169	0.00

Function Profile View

Menu to allow user to enable filtering or displaying the source code

Filter: Function total time > 2.0%
Filter: Function self time > 2.0%
Filter: Loops for selected function
View: Function source for selected function

Column headers allow selection to control sort criteria independently for function and loop table

Loop Profile

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

Loop Profile View

Code the Future

OPTIMIZATION NOTICE

Code the Future

OPTIMIZATION NOTICE

Code the Future

OPTIMIZATION NOTICE

Vector Report

Want to know if the compiler vectorized your code the vector report will tell you

Adding –vec-report[n] will give you output from compiler.

```
-vec-report[n]
          control amount of vectorizer diagnostic information
            n=0    no diagnostic information
            n=1    indicate vectorized loops (DEFAULT when enabled)
            n=2    indicate vectorized/non-vectorized loops
            n=3    indicate vectorized/non-vectorized loops and prohibiting
                   data dependence information
            n=4    indicate non-vectorized loops
            n=5    indicate non-vectorized loops and prohibiting data
                   dependence information
            n=6    indicate vectorized/non-vectorized loops with greater
                   details and prohibiting data dependence information
```

Code the Future

Vector Report

Compiler will not vectorize a loop if it can't be certain at compile time that it is safe to do so.

- #pragma simd
  is a way to assert to the compiler that everything in the loop is safe to vectorize. (DANGEROUS)

Code the Future

Vector Report

Let's run the vector report on the offload code and see if we can make it run faster.

```
lkmcraft@dpdmic09 beginners1$ icpc -openmp -O3 -vec-report3 offload.cpp
```

Let's change line 118 to #pragma simd
You will see that the compiler vectorized the loop now

Code the Future

Memory Alignment

– Static memory

o Allocated by compiler/linker

o Add __attribute__((aligned(n))) in front of variable declaration

o Applies to global/local static variables as well as stack/auto variables

– Dynamic memory

o Allocated by language runtime

o Use __mm_aligned_malloc(size, alignment_bytes)

o Example: buf = (char*) _mm_malloc(bufsize, 4096);

o Pair it with __mm_aligned_free()

Code the Future

Code the Future

OPTIMIZATION NOTICE

Code the Future

OPTIMIZATION NOTICE

Other hints from the compiler

- -guide

  This flag combined with –parallel will give the user feedback from the compiler which can also help you parallelize your code.

Code the Future

Linux: Documentation, Samples, and Tutorials

HTML- and PDF-based documentation:

- <install dir>/composerxe/Documentation/en_US/
  - Release_Notes[F | C].pdf
  - documentation_[f | c].htm

Samples:
<installdir>/Samples/en_US/[C++ | Fortran]/sample.htm

- Vectorization, openmp, PGO, IPO, GAP, Coarray Fortran, Cilk Plus

Tutorials: `<installdir>/composerxe/Documentation/en_US/`

/tutorials_[c | f]/index.htm

- Tutorials are based on the Samples included
- Steps user through new technologies

Code the Future

Code the Future

OPTIMIZATION NOTICE

Code the Future

OPTIMIZATION NOTICE

intel®

Software

OpenMP 4.0* support

- TEAMS pragmas, directives and clauses
- DISTRIBUTE pragmas, directive and clauses
- SIMD pragmas, directives, and clauses
- TARGET pragmas, directives and clauses for attached coprocessors (or devices)
- #pragma omp taskgroup construct
- Atomic clause seq_cst
- Six new forms of atomic capture and update:
  - o Atomic swap: {v = x; x = expr}
  - o Atomic update: x = expr binop x;
  - o Atomic capture 1: v = x = x binop expr;
  - o Atomic capture 2: v =x = expr binop x;
  - o Atomic capture 3: {x = expr binop x; v = x;}
  - o Atomic capture 4: {v = x; x = expr binop x;}
- proc_bind(<type>) clause where <type> is “spread”, “close”, or “master”
- OMP_PLACES environment variable
- OMP_PROC_BIND environment variable
- omp_get_proc_bind() API