(intel) Look Inside™

Open Source Software Development

Getting ready for Knights Landing!

Intel High Performance and Throughput Computing (EMEA)
Hans Pabst, March 12th 2015

Abstract

After a short introduction on what to expect from Intel's next iteration of the Many Integrated Core Architecture, the talk will portrait a selection of recent work of Intel's High Performance and Throughput Computing Team in Europe. Among the selected projects are LIBXSMM (library for multiplying batches of small matrices as common in quantum dynamics applications), LIBXSTREAM (library for stream programming), and pyMIC (module to offload computation in a Python program to the Intel Xeon Phi coprocessor).

The tutorial part of the talk shows how to run and debug an application in the Intel Software Development Emulator (Intel SDE). Compiling, running, and debugging an application in the Intel SDE can be a useful step to exercise the development tool chains such as the GNU Compiler Collection (GNU GCC) or the Intel Compiler. The talk closes with some tricks and hints on how to approach code modernization and further optimizations making an application ready for lots of cores and wide vectors.

**Open Source Software Development –
Getting ready for Knights Landing!**

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Open Source Software Development

“Getting ready for Knights Landing!”

- Selection of work (Intel Intel High Performance and Throughput Computing team*)
- Introduction of the Intel AVX-512 Instruction Set Extension with focus on HPC
- Practical steps to prepare software for Knights Landing

* This is only focusing on our team, and cannot represent all of Intel’s Open Source Software Commitment.

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

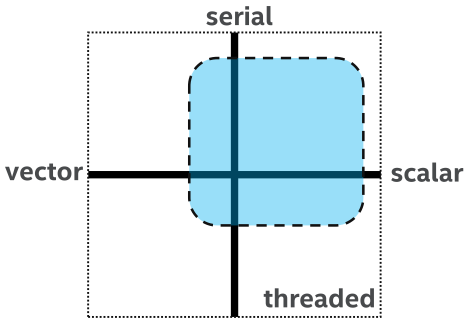
Intel Knights Landing?

The 2nd iteration of Intel Many Integrated Core Architecture

- Available today: Intel® Xeon Phi™ Coprocessor (code-named “Knights Corner”)

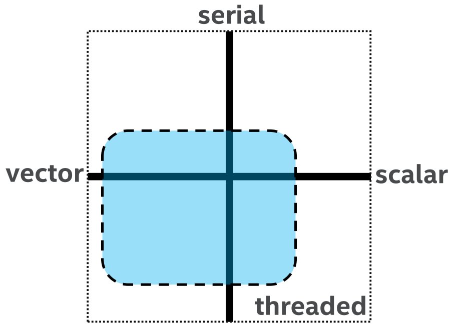
What’s publicly known? Just have a look:

- https://software.intel.com/en-us/articles/what-disclosures-has-intel-made-about-knights-landing



Most Commonly Used Parallel Processor*

Parallel, fast serial execution
Multicore and vectors



Optimized for Highly Parallel Applications

Many cores, 512 bit vectors, and higher mem. bw.
Common languages, directives, libraries and tools

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

There is a growing catalog of applications…

More than 100 applications are listed as available today or in flight

Intel® Xeon Phi™ Coprocessor

Applications and Solutions Catalog

Public Version | Rev 2 | November 2014

http://software.intel.com/XeonPhiCatalog

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Agenda

Examples of our 2014 work in the Open Source domain

- LIBXSMM – small matrix-matrix multiplications
- pyMIC – offload computation to Xeon Phi coprocessors
- LIBXSTREAM – streams, events, and offload functions

Introducing the AVX-512 Instruction Set Extension

Running an application using the Intel SDE

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel® AVX-512

Instruction Set Extension

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel Instruction Set Extension (1998-2010)

1998	1999	2004	2006	2007	2008	2009
Intel® SSE	Intel® SSE2	Intel® SSE3	Intel SSSE3	Intel® SSE4.1	Intel® SSE4.2	Intel® AES
70 new instructions 4 single-precision vector FP scalar FP instructions cacheability instructions control & conversion instructions media extensions	144 new instructions 2 double-precision vector FP 8/16/32/64 vector integer 128-bit integer memory & power management	13 new instructions FP vector calculation x87 integer conversion 128-bit integer unaligned load thread sync.	32 new instructions enhanced packed integer calculation	47 new instructions packed integer calculation & conversion better vectorization by compiler load with streaming hint	7 new instructions string (XML) processing POP-Count CRC32	7 new instructions 6 for acceleration of AES algorithm Plus carry-less 64bt multiply

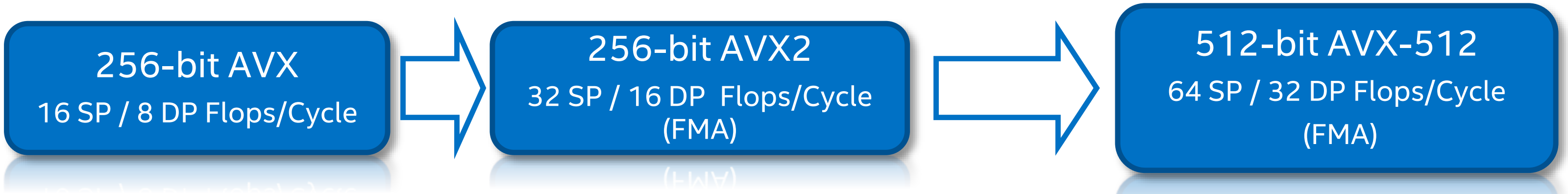
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel Instruction Set Extension (after 2010)

2011	2011	2012	2013	2014	TBD
Intel® AVX	Intel® Initial Many Core Instructions	"AVX-1.5"	Intel® AVX-2	Broadwell New Instructions	Intel® AVX-512
Promotion of 128 bit FP vector instructions to 256 bit	Hundreds of new 512 bit vector instructions only available for MIC architecture – not supported by and not compatible to x86 architecture	7 new instructions 16 bit FP support RDRAND …	Promotion of integer instruction to 256 bit - FMA - Gather - TSX/RTM	Some 5 new instruction to enhance support for random number generation, prefetching and multi-precision arithmetic	Promotion of vector instructions to 512 bits and a lot more

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel® AVX



256-bit AVX
16 SP / 8 DP Flops/Cycle

256-bit AVX2
32 SP / 16 DP Flops/Cycle (FMA)

512-bit AVX-512
64 SP / 32 DP Flops/Cycle (FMA)

AVX

256-bit basic FP
16 registers
NDS (and AVX128)
Improved blend
MASKMOV
Implicit unaligned

SNB
2011

AVX2

Float16 (IVB 2012)
256-bit FP FMA
256-bit integer
PERMD
Gather

HSW
2013

AVX-512

512-bit FP/Integer
32 registers
8 mask registers
Embedded rounding
Embedded broadcast
Scalar/SSE/AVX “promotions”
HPC additions
Transcendental support
Gather/Scatter

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

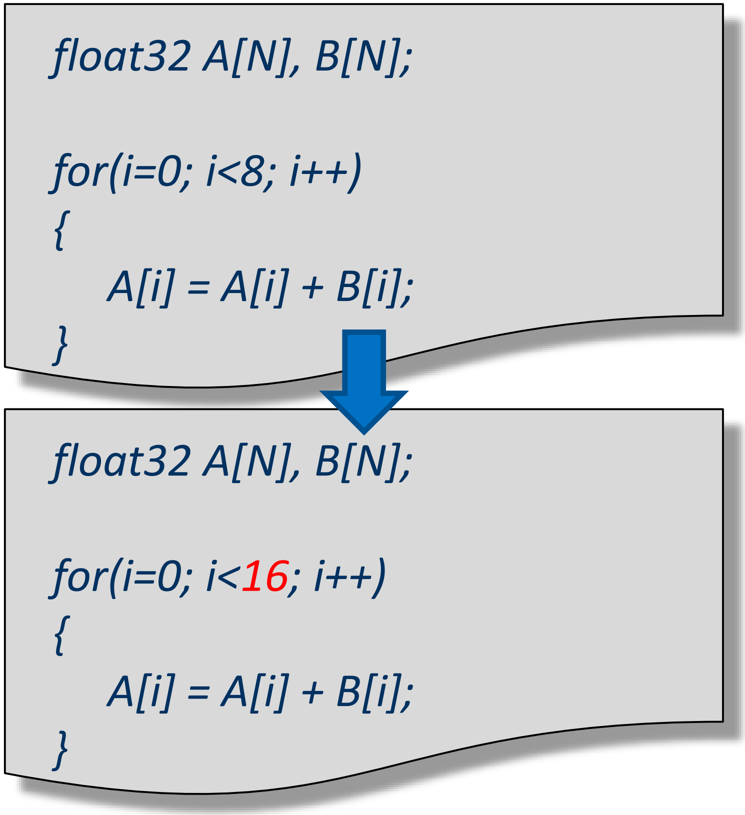
AVX-512: More and Larger Registers

AVX / AVX-2: VADDPS YMM0, YMM3, [mem]

- Up to 16 AVX registers
  - 8 in 32-bit mode
- 256-bit width
  - 8 x FP32
  - 4 x FP64

AVX-512: VADDPS ZMM0, ZMM24, [mem]

- Up to 32 AVX registers
  - 8 in 32-bit mode
- 512-bit width
  - 16 x FP32
  - 8 x FP64



```
float32 A[N], B[N];

for(i=0; i<8; i++)
{
    A[i] = A[i] + B[i];
}
```

```
float32 A[N], B[N];

for(i=0; i<16; i++)
{
    A[i] = A[i] + B[i];
}
```

There is a lot more (instructions) needed in order to effectively use the new real estate…

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Mask Registers

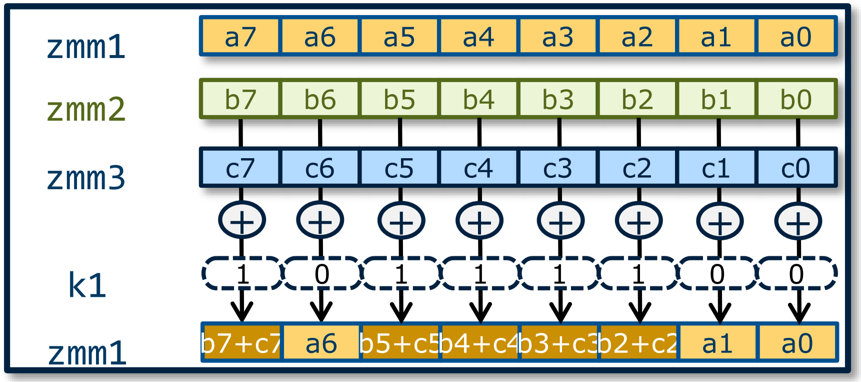
8 Mask registers of size 64-bits

- k1-k7 can be used for predication
  - k0 always has value 0xFFFFFFFFFFFFFFFF
  - k0 can be used as a destination or source for mask manipulation operations

4 different mask granularities. For instance, at 512b:

- Packed Integer Byte use mask bits [63:0]
  - `VPADDB zmm1 {k1}, zmm2, zmm3`
- Packed Integer Word use mask bits [31:0]
  - `VPADDW zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP32 and Integer Dword use mask bits [15:0]
  - `VADDPS zmm1 {k1}, zmm2, zmm3`
- Packed IEEE FP64 and Integer Qword use mask bits [7:0]
  - `VADDPD zmm1 {k1}, zmm2, zmm3`

`VADDPD zmm1 {k1}, zmm2, zmm3`



zmm1	a7	a6	a5	a4	a3	a2	a1	a0
zmm2	b7	b6	b5	b4	b3	b2	b1	b0
zmm3	c7	c6	c5	c4	c3	c2	c1	c0
k1	1	0	1	1	1	1	0	0
zmm1	b7+c7	a6	b5+c5	b4+c4	b3+c3	b2+c2	a1	a0

Why Separate Mask Registers?

- Avoids using registers as vector of Booleans
- Separate control flow and data flow
- Boolean operations on logical predicates consume less energy (separate functional unit)
- Tight encoding allows orthogonal operand
- All instructions support an extra mask operand

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Masking

VADDPS ZMM0 {k1}, ZMM3, [mem]

Mask bits are used to:

1. Suppress reading individual elements as well as signaling memory faults for masked elements
2. Avoid signaling individual FP faults per instruction
3. Avoid updating individual destination elements or force to zero (zeroing)

```
for (I in vector length)
{
   if (no_masking or mask[I]) {
      dest[I] = OP(src2, src3)
   } else {
      if (zeroing_masking)
         dest[I]  = 0
      else
         // dest[I] is preserved
   }
}
```

Caveat: vector shuffles do no suppress memory fault exceptions since the mask refers to the “output” not to the “input”.

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

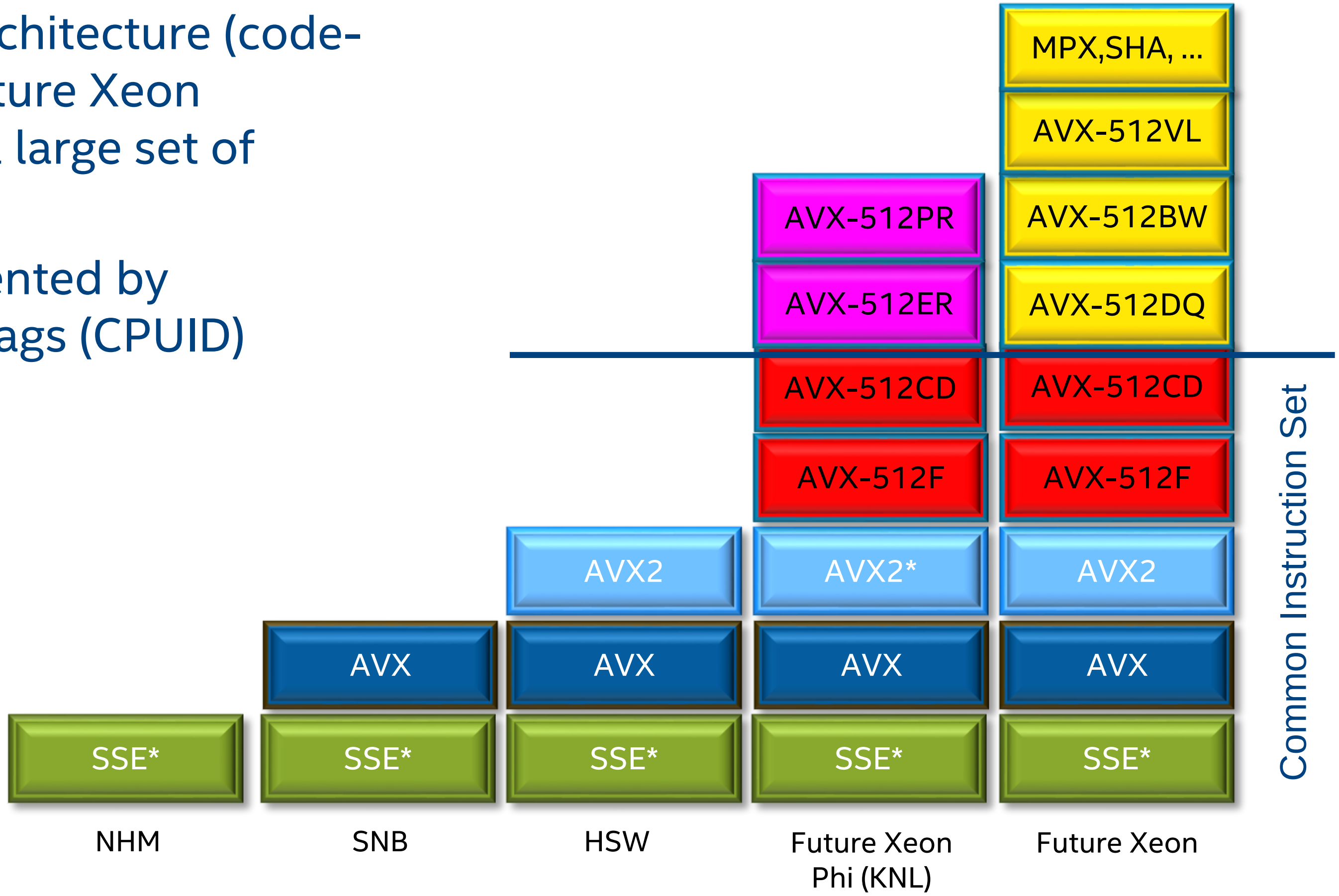
AVX-512: More Features…

- Embedded Broadcasts and Masking
- Conflict Detection Instructions
- Embedded Rounding Control
- **S**uppress **A**ll **E**xceptions (SAE)
- Compressed Displacement
- Bit Manipulation
- Shuffles
- …

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel AVX-512: Hardware Support

- Future Intel® MIC architecture (code-named KNL) and future Xeon architecture share a large set of instructions
- Subsets are represented by individual feature flags (CPUID)



NHM	SNB	HSW	Future Xeon Phi (KNL)	Future Xeon
				MPX,SHA, ...
				AVX-512VL
			AVX-512PR	AVX-512BW
			AVX-512ER	AVX-512DQ
			AVX-512CD	AVX-512CD
			AVX-512F	AVX-512F
		AVX2	AVX2*	AVX2
	AVX	AVX	AVX	AVX
SSE*	SSE*	SSE*	SSE*	SSE*

Common Instruction Set

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Instruction Subsets

AVX-512F

AVX-512 F: 512-bit Foundation instructions common between MIC and Xeon

- ❑ Comprehensive vector extension for HPC and enterprise
- ❑ All the key AVX-512 features: masking, broadcast…
- ❑ 32-bit and 64-bit integer and floating-point instructions
- ❑ Promotion of many AVX and AVX2 instructions to AVX-512
- ❑ Many new instructions added to accelerate HPC workloads

AVX-512CD

AVX-512 CD (Conflict Detection instructions)

- ❑ Allow vectorization of loops with possible address conflict

AVX-512ER

AVX-512PR

AVX-512 extensions for exponential and prefetch operations

- ❑ fast (28 bit) FP instr. for exponential, reciprocal and Transcendentals (incl. RSQRT)
- ❑ New prefetch instructions: gather/scatter prefetches and PREFETCHWT1

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Other New Instructions

MPX

Intel® MPX – Intel Memory Protection Extension

□Set of instructions to implement checking a pointer against its bounds

□Pointer Checker support in HW (today: SW-only solution)

□Debug and security features

SHA

Intel® SHA – Intel Secure Hash Algorithm

□ Fast implementation of cryptographic hashing algorithm as defined by NIST FIPS PUB 180

CLFLUSHOPT

Single Instruction – Flush a cache line

□ needed for future memory technologies

XSAVE{S,C}

Save and restore extended processor state

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Summary (F, CD, ER, PR)

AVX-512 F: new 512-bit vector ISA extension

- Common between Xeon and Xeon Phi (KNL)

AVX-512 CD Conflict detection instructions

- Will be available both for Xeon and Xeon Phi (KNL)
- Improves auto-vectorization

AVX-512 ER & PR

- 28-bit Transcendentals and new prefetch instructions
- On Xeon Phi (KNL) only

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: How to use?

Instructions are accessible using Intrinsics (built-in functions)

- No need to supply any additional compiler flags
- Supported since Intel Compiler 14 and GCC 4.8

Compiler code generation

Intel C/C++ and Fortran Compiler (Version 15 and newer)

- KNL via "MIC-AVX512" (`-xMIC-AVX512`)
- Future Xeon via "CORE-AVX512" (`-xCORE-AVX512`)

GNU Compiler Collection (GCC 4.9)

- Subset support, all Intrinsics, and basic auto-vectorization via `-mavx512f`, `-mavx512pf`, `-mavx512er`, and `-mavx512cd`
- Example for KNL: supply all flags from above (F/PF/ER/CD)

LLVM Compiler

- Full support (patch: https://groups.google.com/forum/)

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: References

Reference manual for Intel® AVX-512 instructions

- http://software.intel.com/en-us/intel-isa-extensions

Other references

- http://software.intel.com/en-us/blogs/2013/avx-512-instructions

Discussion and critics

- http://software.intel.com/en-us/forums/topic/477541
- See www.agner.org

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Intel® SDE and use cases (examples)

Intel® Software Development Emulator

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Overview

Run an application using an instruction set extension which is not (yet) available in hardware.

- Correctness testing (exercising the compiler’s code generation)
- Prepare the build system (compiler flags)
- Debugging

Technology background

- Runs an application roughly an order of magnitude slower (compared to the host system which is running the SDE)
- Not a performance tool (profiler) as it uses binary translation to the host’s instruction set

```
$ $ /path/to/sde/sde64 -knl -- /path/to/application
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Example: LIBINT Version 1.1.5

Prerequisites

- Intel Cluster Studio XE (e.g., composer_xe_2015.1.133 or newer)
- Intel SDE (sde-external-7.15.0-2015-01-11-lin)

Notes

- Unfortunately the configure script attempts to run a program; cross-compilation complicates the configure/build process
- Therefore the SDE must be used during configuration (and initial steps of make; make can be interrupted and continued without the SDE)

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Example: LIBINT (cont.)

Configuration

```
$ make realclean

$ env AR=xiar \
     FC=ifort F77=ifort F90=ifort CXX=icpc CC=icc \
     FCFLAGS="-O2 -xMIC-AVX512 -ipo" \
     CFLAGS="-O2 -xMIC-AVX512 -ipo" \
     CXXFLAGS="-O2 -xMIC-AVX512 -ipo" \
  ~/sde/sde64 -knl -- ./configure \
     --with-cc-optflags="-O2 -xMIC-AVX512 -ipo" \
     --with-cxx-optflags="-O2 -xMIC-AVX512 -ipo" \
     --with-libint-max-am=5 --with-libderiv-max-am1=4 \
     --prefix=$HOME/libint
```

Building LIBINT

```
$ make -j          (initially: ~/sde/sde64 -knl -- make)

$ make install
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Example: LIBXC Version 2.2.2

Configuration

```
$ make clean        (note: there is no “realclean”)

$ env AR=xiar \
     FC=ifort F77=ifort F90=ifort CC=icc \
     FCFLAGS="-O2 -xMIC-AVX512 -ipo" \
     CFLAGS="-O2 -xMIC-AVX512 -ipo" \
  ~/sde/sde64 -knl -- ./configure \
     --prefix=$HOME/libint
```

Building LIBXC

```
$ make -j

$ make install
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

CP2K

Prerequisites

- Intel Cluster Studio XE (composer_xe_2015.1.133, impi-5.0.2.044)
- Intel SDE (sde-external-7.15.0-2015-01-11-lin)
- LIBINT (Version 1.1.5) and LIBXC (Version 2.2.2)

Source code

- git clone https://github.com/cp2k/cp2k.git
- git checkout intel

Edit arch file

- arch/Linux-x86-64-intel-mic.psmp: **`-xHost`** → **`-xMIC-AVX512 -g`**

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

CP2K (cont.)

Building CP2K

```
$ make ARCH=Linux-x86-64-intel-host VERSION=sopt \
     LIBINTROOT=~/libint LIBXCROOT=~/libxc
```

Running CP2K

```
$ ~/sde/sde64 -knl -- exe/Linux-x86-64-intel-
  host/cp2k.sopt tests/LIBTEST/dbcsr_blocks_04.inp
```

Debugging

```
$ ~/sde/sde64 -knl -debug -- exe/Linux-x86-64-intel-
  host/cp2k.sopt tests/LIBTEST/dbcsr_blocks_04.inp
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Steps for Debugging

1. Run application with SDE’s “–debug” option; starts gdb-server and prints command on how to connect using the gdb client
2. Open gdb (or try “gdb-ia” if Intel compiler suite is source’d) but supply the /path/to/executable on command line (ensures that debug symbols are loaded)
3. Paste command shown by SDE on how to connect to the gdb-server (port), and continue (c) execution

Reference

https://software.intel.com/en-us/articles/debugging-applications-with-intel-sde

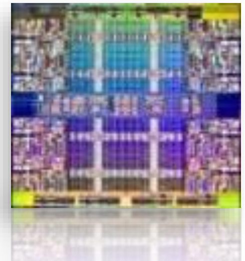
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

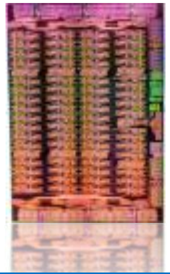
What else?

Code Modernization

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Code Modernization using Intel® Xeon Phi™ coprocessor(s)





tba

tba

Future

The world is going parallel – stick with sequential code and you will fall behind.

	Intel® Xeon® Processor E5-2600 v2 Product Family formerly codenamed **IvyBridge**	*Intel® Xeon® Processor E5-2600 v3 Product Family formerly codenamed* **Haswell**	*Intel® Xeon Phi™ x100 Product Family formerly codenamed* **Knights Corner**	*Intel® Xeon Phi™ x200 Product Family codenamed* **Knights Landing**	… **Future Xeon**
Cores*	12	18	61	72	tba
Threads/Core	2	2	4	4	tba
Vector Width	256-bit	256-bit	512-bit	512-bit	512-bit
Memory Bandwidth*	59 GB/s	68 GB/s	352 GB/s	~500 GB/s	tba

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Code Modernization: Getting Started

- Preparing the build system (compiler flags) and linking Intel Math Kernel Library (Intel MKL)

  Potentially low-hanging fruits with using industry-standard interfaces such as (Sca-)LAPACK/BLAS and FFTW3

- Evaluating parallel scalability (MPI and OpenMP) using higher core-count systems (Xeon Phi, Xeon E5, and Xeon E7)

  Initially this does not even ask for advanced profiling, just a wall clock…

- Generating and reviewing compiler vectorization reports

  Intel Compiler Version 15 reports are much more readable/correlated

  Hint: try upcoming version of Intel Advisor (vector reports, and more)

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

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 © 2015, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo 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

intel®

Backup

AVX-512

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Why True Masking?

Memory fault suppression

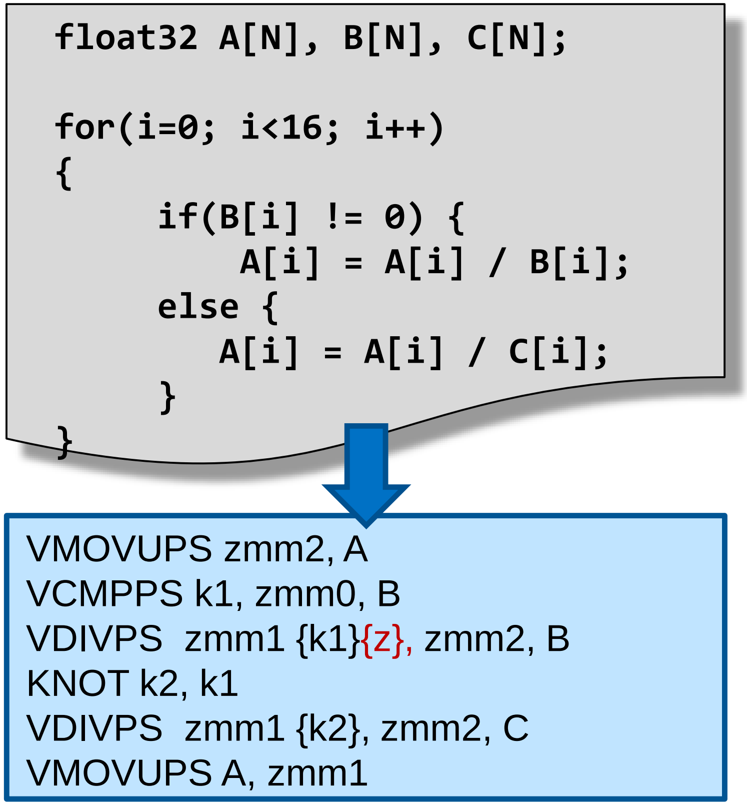
- Vectorize code without touching memory that the correspondent scalar code would not touch
  - Typical examples are if-conditional statements or loop remainders
  - AVX is forced to use VMASKMOV* (risc)

MXCSR flag updates and fault handlers

- Avoid spurious floating-point exceptions without having to inject neutral data

Zeroing/merging

- {z} bit syntax to EVEX z-bit - implies 'zeroing' – default is 'merging'
- Use zeroing to avoid false dependencies in OOO architecture
- Use merging to avoid extra blends in if-then-else clauses (predication) for great code density



```
float32 A[N], B[N], C[N];

for(i=0; i<16; i++)
{
     if(B[i] != 0) {
         A[i] = A[i] / B[i];
     else {
        A[i] = A[i] / C[i];
     }
}
```

```
VMOVUPS zmm2, A
VCMPPS k1, zmm0, B
VDIVPS  zmm1 {k1}{z}, zmm2, B
KNOT k2, k1
VDIVPS  zmm1 {k2}, zmm2, C
VMOVUPS A, zmm1
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Embedded Broadcasts and Masking

VFMADD231PS zmm1, zmm2, C {1to16}

- Scalars *from memory* are first class citizens
  - Broadcast one scalar from memory into all vector elements before operation
- Memory fault suppression avoids fetching the scalar if no mask bit is set to 1

```
float32 A[N], B[N], C;

for(i=0; i<8; i++)
{
   if ( A[i]!=0.0 )
      A[i] = A[i] + C*B[i];
}
```

Other “tuples” supported

- Memory only touched if at least one consumer lane needs the data
- For instance, when broadcast a tuple of 4 elements, the semantics check for every element being really used
  - e.g.: element 1 checks for mask bits 1, 5, 9, 13, …

```
VBROADCASTSS zmm1 {k1}, [rax]
VBROADCASTF64X2 zmm2 {k1}, [rax]
VBROADCASTF32X4 zmm3 {k1}, [rax]
VBROADCASTF32X8 zmm4, {k1}, [rax]
…
```

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Embedded Rounding Control and Suppress All Exceptions (SAE)

Embedded Rounding Control :

- MXCSR.RC can be overridden on all FP instructions
  - VADDPS ZMM1 {k1}, ZMM2, [mem] {1→16} {rne-sae}
- "Suspend All Exceptions"
  - Always implied by using embedded RC
    - NO MXCSR updates / exception reporting for any lane
  - Changes to RC without SAE via LDMXCSR
    - Not needed for most common case (truncating FP convert to int)
  - Only available for reg-reg mode and 512b operands

Main application:

- Saving, modifying and restoring MXCSR is usually slow and cumbersome
- Being able to avoid suppressions and set the rounding-mode on a per instruction basis simplifies development of high performance math software sequences (math libs)
  - E.g.: avoid spurious overflow/underflow reporting in intermediate computations
  - E.g: make sure that RM=rne regardless of the contents of MXCSR

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Compressed Displacement

VADDPS zmm1, zmm2, [rax+256]

- Observation is that displacement in generated vector code is a multiple of the actual operand size
  - An obvious side effect of unrolling

- Unfortunately, regular IA 8-bit displacement format have limited scope for 512-bit vector sizes (unrolling look-ahead of +/-2 at most)
  - So we would end up using 32-bit displacement formats too often

AVX-512 disp8*N compressed displacement

- AVX-512 implicitly encodes a 8-bit displacement as a multiple of the actual size of the memory operand
  - VADDPD zmm1 {k1}, zmm2, [rax] memory size operand is 512bits
  - VADDPD xmm1 {k1}, xmm2, [rax] memory size operand is 128bits
  - VADDPD zmm1 {k1}, xmm2, [rax] {1toN} memory size operand is 64 bits

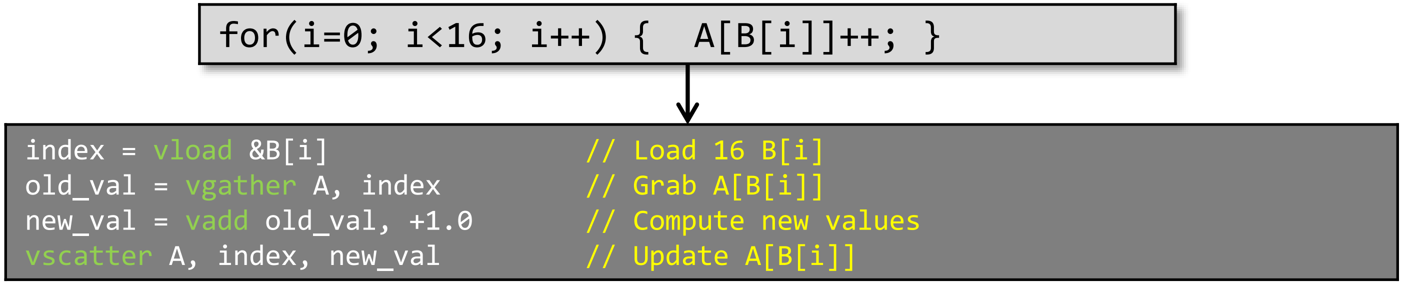
- Assembler/compiler reverts to 32-bit displacement when the real displacement is not a multiple

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Motivation for Conflict Detection

Sparse computations are common in HPC, but hard to vectorize due to race conditions

Consider the “histogram” problem:



```
for(i=0; i<16; i++) {  A[B[i]]++; }
```

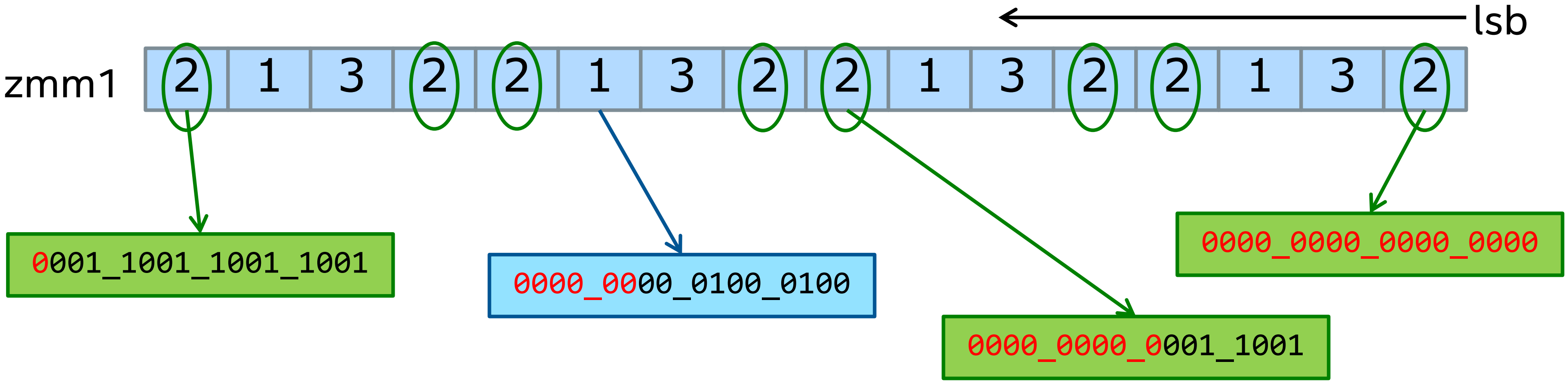
```
index = vload &B[i]                // Load 16 B[i]
old_val = vgather A, index         // Grab A[B[i]]
new_val = vadd old_val, +1.0       // Compute new values
vscatter A, index, new_val         // Update A[B[i]]
```

- Code above is wrong if any values within B[i] are duplicated
  - Only one update from the repeated index would be registered!
- A solution to the problem would be to avoid executing the sequence gather-op-scatter with vector of indexes that contain conflicts

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

VPCONFLICT{D,Q}

- VPCONFLICT{D,Q} zmm1{k1}{z}, zmm2/B(mV)
  - For every element in ZMM2, compare it against everybody and generate a mask identifying the matches (but ignoring elements to the ‘left’ of the current one –i.e. “newer”)
    - Store every mask in every element destination in ZMM1



Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Optimized Algorithm

```
for each 16 scalar iterations {
    indices = vload &index_array[i]
     vpconflictd comparisons, indices
     vplzcntd tmp_lzcnt, comparisons
     vpsubd perm_idx, all_31s, tmp_lzcnt
     temp_values = do_first_iteration(); // gather + compute
     vptestmd to_do {k0}, comparisons, all_ones // anything left?

     while (to_do) {
       vpbroadcastmd tmp, to_do
       vptestnmd mask {to_do}, comparisons, tmp
       vpermd tmp_values {mask}, perm_idx
       tmp_values = do_work(mask); // just compute!
       to_do ^= mask;
     } while(to_do);
     vscatter indices, A, tmp_values

}
```

Obtain recurrence indices

Re-do conflicting indices reusing results directly from the vector

Store results

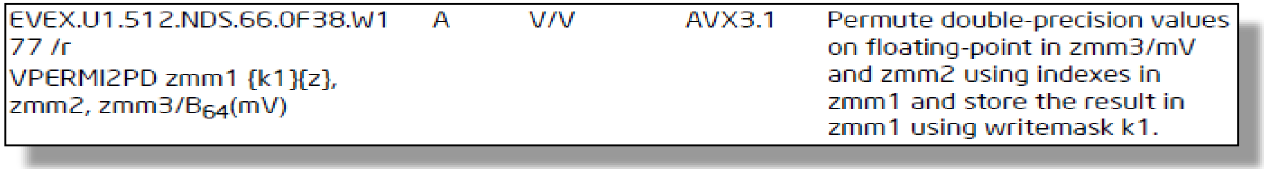
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

New 2-Source Shuffles

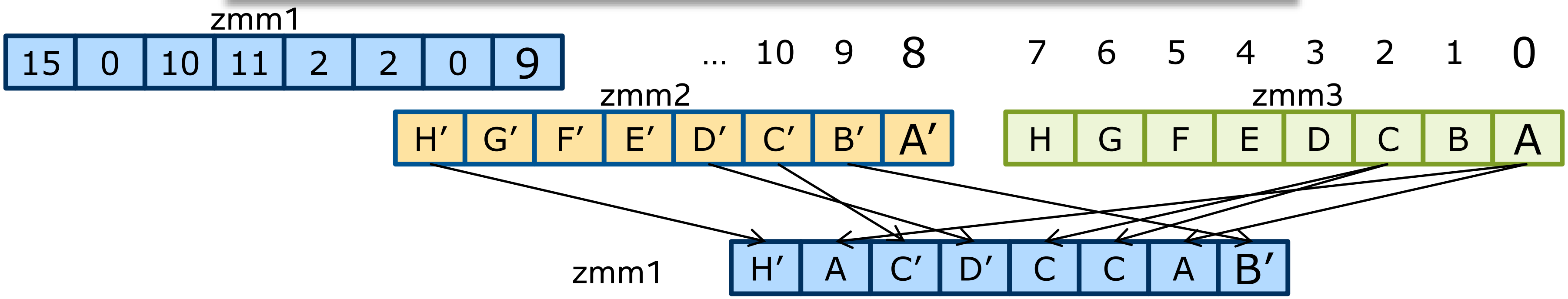
2-Src Shuffles
VSHUF{PS,PD}
VPUNPCK{H,L}{DQ,QDQ}
VUNPCK{H,L}{PS,PD}
VPERM{I,D}2{D,Q,PS,PD}
VSHUF{F,I}32X4

Long standing customer request

- 16/32-entry table lookup (transcendental support)
- AOS ⇔ SOA support, matrix transpose
- Variable VALIGN emulation



EVEX.U1.512.NDS.66.0F38.W1 77 /r VPERMI2PD zmm1 {k1}{z}, zmm2, zmm3/B_{64}(mV)	A	V/V	AVX3.1	Permute double-precision values on floating-point in zmm3/mV and zmm2 using indexes in zmm1 and store the result in zmm1 using writemask k1.



zmm1

15 0 10 11 2 2 0 9

... 10 9 8 7 6 5 4 3 2 1 0

zmm2

H′ G′ F′ E′ D′ C′ B′ A′

zmm3

H G F E D C B A

zmm1

H′ A C′ D′ C C A B′

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Expand & Compress

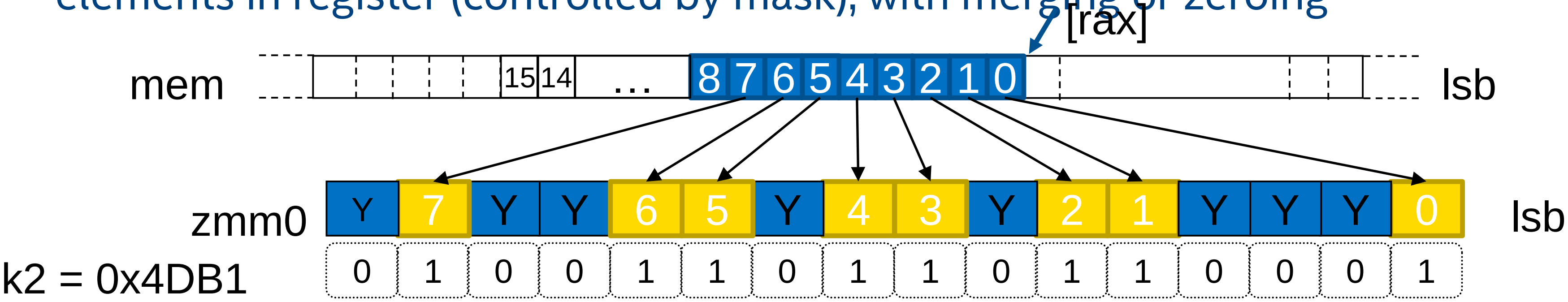
Allows vectorization of conditional loops

- Opposite operation (compress) not in AVX-512F
- Similar to FORTRAN pack/unpack intrinsics
- Provides mem fault suppression
- Faster than alternative gather/scatter

```
for(j=0, i=0; i<N; i++)
{
  if(C[i] != 0.0)
    B[i] = A[i] * C[j++];
}
```

VEXPANDPS zmm0 {k2}, [rax]

Moves compressed (consecutive) elements in register or memory to sparse elements in register (controlled by mask), with merging or zeroing



zmm0	Y	7	Y	Y	6	5	Y	4	3	Y	2	1	Y	Y	Y	0
k2 = 0x4DB1	0	1	0	0	1	1	0	1	1	0	1	1	0	0	0	1

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Bit Manipulation

Basic bit manipulation operations on mask and vector operands

- Useful to manipulate mask registers
- Have uses in cryptography algorithms

Instruction	Description
`KUNPCKBW k1, k2, k3`	Interleave bytes in k2 and k3
`KSHIFT{L,R}W k1, k2, imm8`	Shift bits left/right using imm8
`VPROR{D,Q} zmm1 {k1}, zmm2, imm8`	Rotate bits right using imm8
`VPROL{D,Q} zmm1 {k1}, zmm2, imm8`	Rotate bits left using imm8
`VPRORV{D,Q} zmm1 {k1}, zmm2, zmm3/mem`	Rotate bits right w/ variable ctrl
`VPROLV{D,Q} zmm1 {k1}, zmm2, zmm3/mem`	Rotate bits left w/ variable ctrl

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Sample: VPTERNLOG – Ternary Logic Instruction

Mimics a FPGA cell

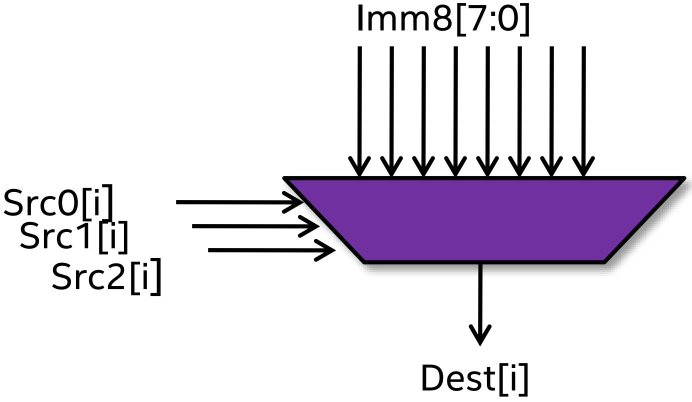
- Take every bit of three sources to obtain a 3-bit index N
  - Obtain Nth bit from imm8

VPTERNLOGD zmm0 {k2}, zmm15, zmm3/[rax], imm8

Any arbitrary truth table of 3 values can be implemented
andor, ***andxor***, ***vote***, ***parity***, bitwise-***cmov***, etc
each column in the right table corresponds to imm8

S1	S2	S3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

ANDOR	VOTE	(S1)?S3:S2
0	0	0
1	0	1
0	0	0
1	1	1
0	0	0
1	1	0
1	1	1
1	1	1



Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

Math Support

Package to aid with Math library writing

- Good value upside in financial applications
- Available in PS, PD, SS and SD data types
- Great in combination with embedded RC

30

Instruction		
VGETXEXP$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain exponent in FP format
VGETMANT$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain normalized mantissa
VRNDSCALE$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, imm8	Round to scaled integral number
VSCALEF $_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, zmm3	$X*2^{y}$, X <= getmant, Y <= getexp
VFIXUPIMM$_{\{PS,PD,SS,SD\}}$	zmm1, zmm2, zmm3, imm8	Patch output numbers based on inputs
VRCP14$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Approx. reciprocal() with rel. error 2^{-14}
VRSQRT14$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Approx. rsqrt() with rel. error 2^{-14}
VDIV$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, zmm3	IEEE division
VSQRT$_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	IEEE square root

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512 ER & AVX-512 PR

Set of segment-specific instruction extensions

- Will be supported in all future Intel® MIC processors
- First appear on KNL (some have similarities in KNC)

Address two HPC customer requests

- Ability to maximize memory bandwidth
  - Hardware prefetching is too restrictive
  - Conventional software prefetching results in instructions overhead
- Flexible support for transcendental operations - accuracy versus speed
  - Mostly division and square root
  - Differentiating factor in HPC/TPT

Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.