Intel® Advanced Vector Extensions 2015/2016 Support in GNU Compiler Collection

GNU Tools Cauldron 2014

Presented by Kirill Yukhin of Intel, July 2014

(kirill.yukhin@intel.com)

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 © 2014, 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

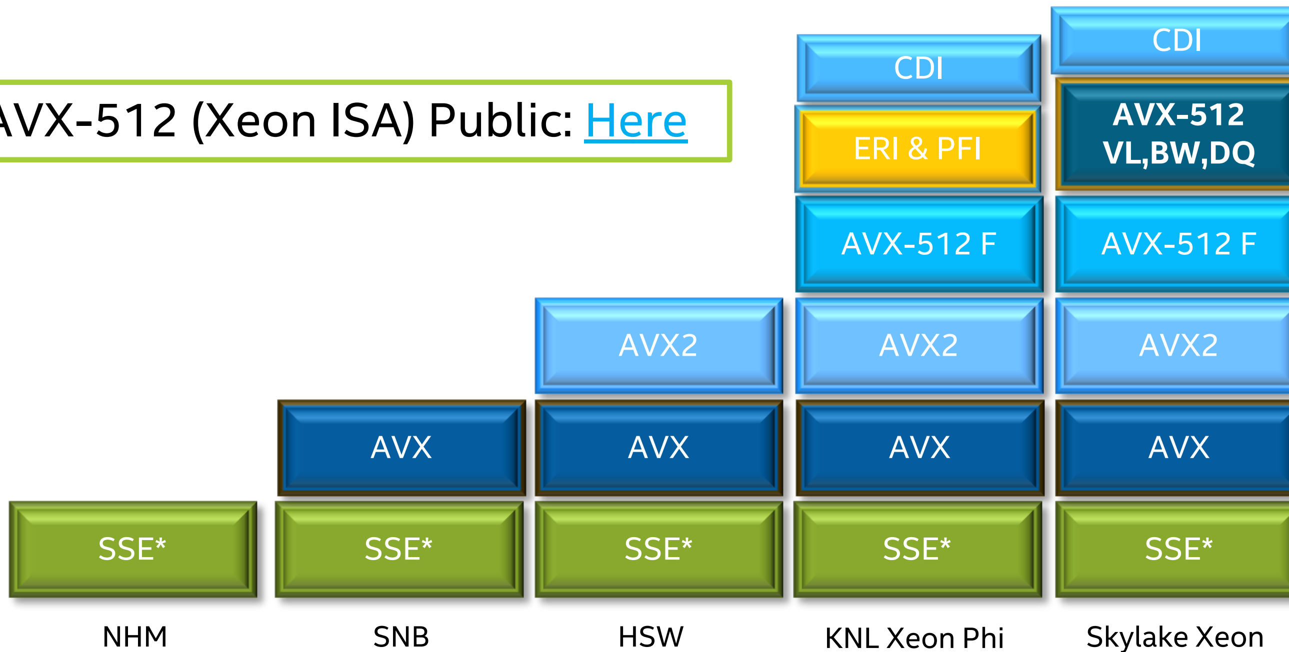
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, 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

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

New ISA: What Is Where?

AVX-512 (Xeon ISA) Public: Here



Will stay exclusive to the Xeon Phi line

Complex & versatile big cores

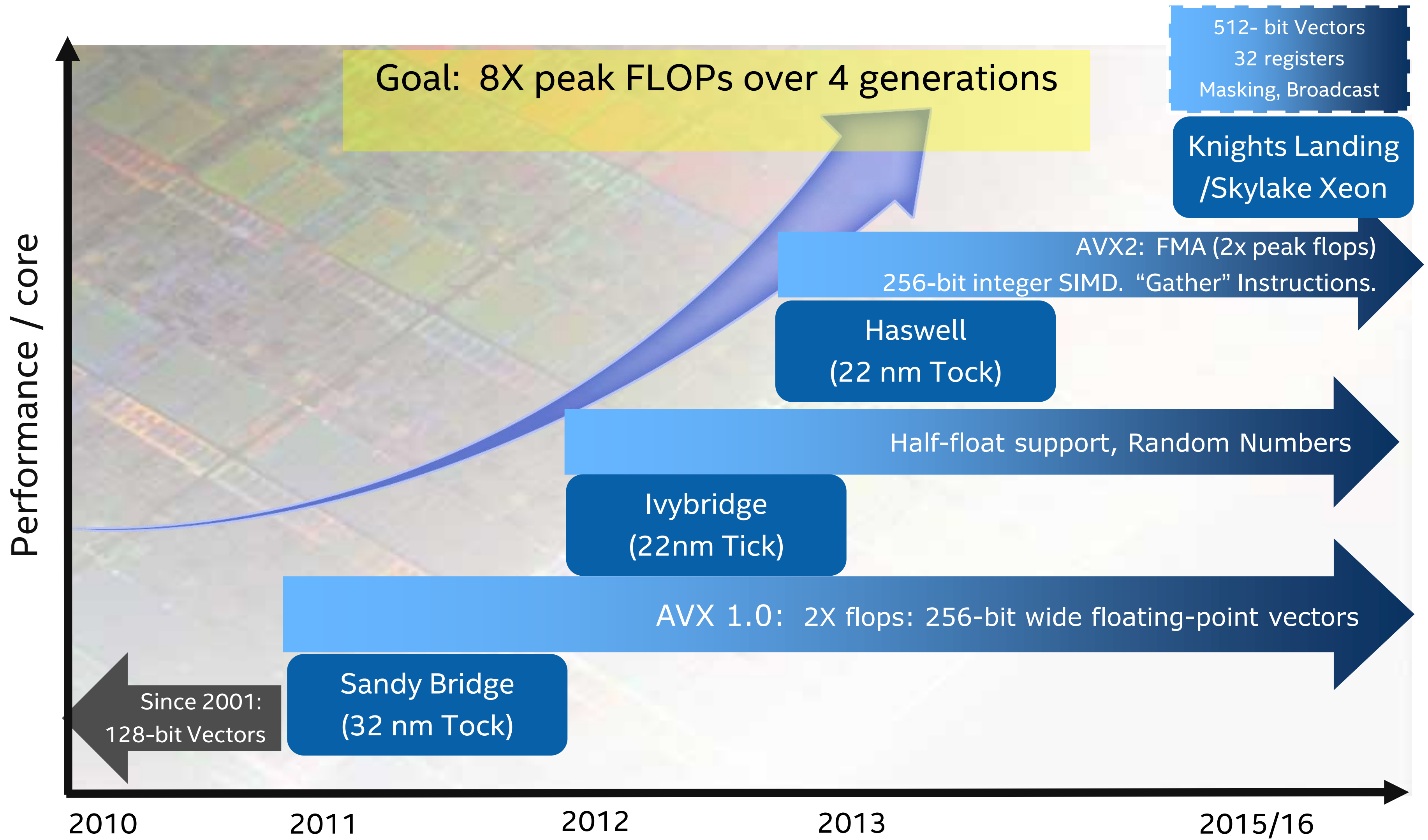
- Big focus on latency and single-thread
- State-of-the-art SIMD support: AVX-512 F + CDI + AVX-512 {VL, DQ, BW}
- Best balance of performance for any workload

Small & efficient cores

- Big focus on throughput and many-threads
- State-of-the-art SIMD support for HPC: AVX-512 F + CDI + ERI + PFI
- Industry performance-per-watt leadership

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

Intel® Advanced Vector Extensions



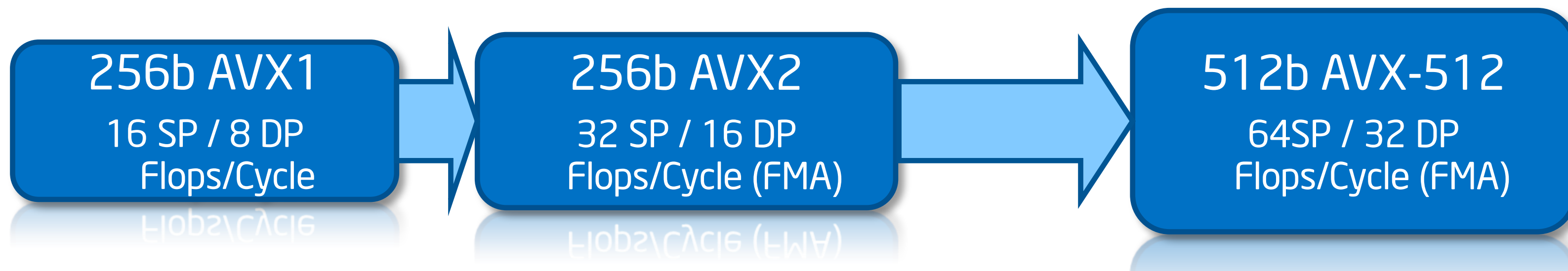
Roadmap illustration - subject to change

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

Introducing AVX-512

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

Intel® AVX Technology



AVX	AVX2
256-bit basic FP	Float16 (IVB 2012)
16 registers	256-bit FP FMA
NDS (and AVX128)	256-bit integer
Improved blend	PERMD
MASKMOV	Gather
Implicit unaligned	

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

SNB
2011

HSW
2013

Future Processors (KNL & SKX)
in planning, subject to change

Optimization Notice

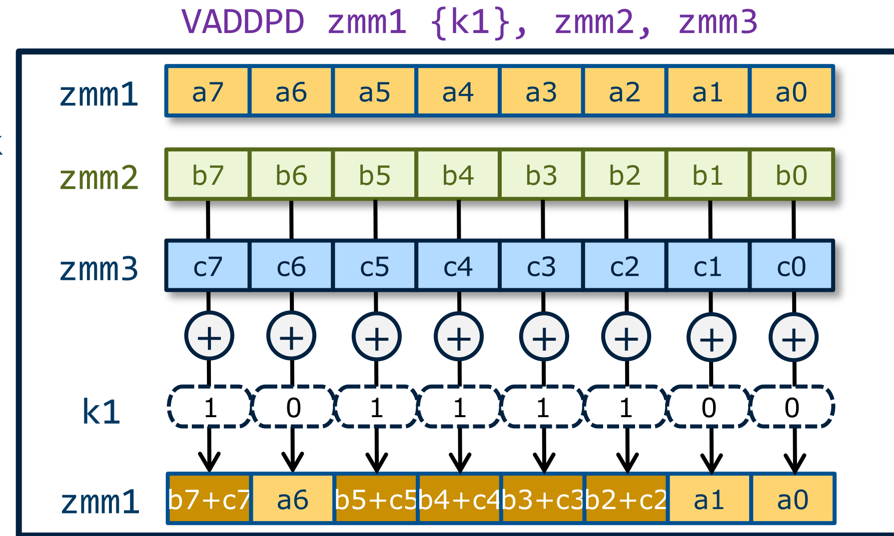
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

intel

AVX-512 Mask Registers

8 Mask registers of size 64-bits

- k1-k7 can be used for predication
  - 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`

		Vector Length		
		128	256	512
element size	Byte	16	32	64
	Word	8	16	32
	Dw ord/SP	4	8	16
	Qw ord/DP	2	4	8

Optimization Notice

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

intel

AVX-512 Features (II): Masking

VADDPS ZMM0 {k1}, ZMM3, [mem]

- Mask bits used to:
  1. *Suppress individual elements read from memory*
     - hence not signaling any memory fault
  2. *Avoid actual independent operations within an instruction happening*
     - and hence not signaling any FP fault
  3. *Avoid the individual destination elements being updated,*
     - or alternatively, force them 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 not suppress memory fault Exceptions as mask refers to “output” not to “input”

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

Embedded Broadcasts

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 Features: Embedded Rounding Control & SAE (Suppress All Exceptions)

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 <u>any</u> 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

Optimization Notice

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

AVX-512F, CDI, ERI & PRI

AVX-512 F

AVX-512 F: 512-bit instructions common between Xeon Phi 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

CDI

AVX-512 CDI (Conflict Detection): Available on Xeon Phi first

- Allow vectorization of loops with possible address conflict
- Will show up on Xeon in SKL or CNL (follow up to SKL)

ERI & PRI

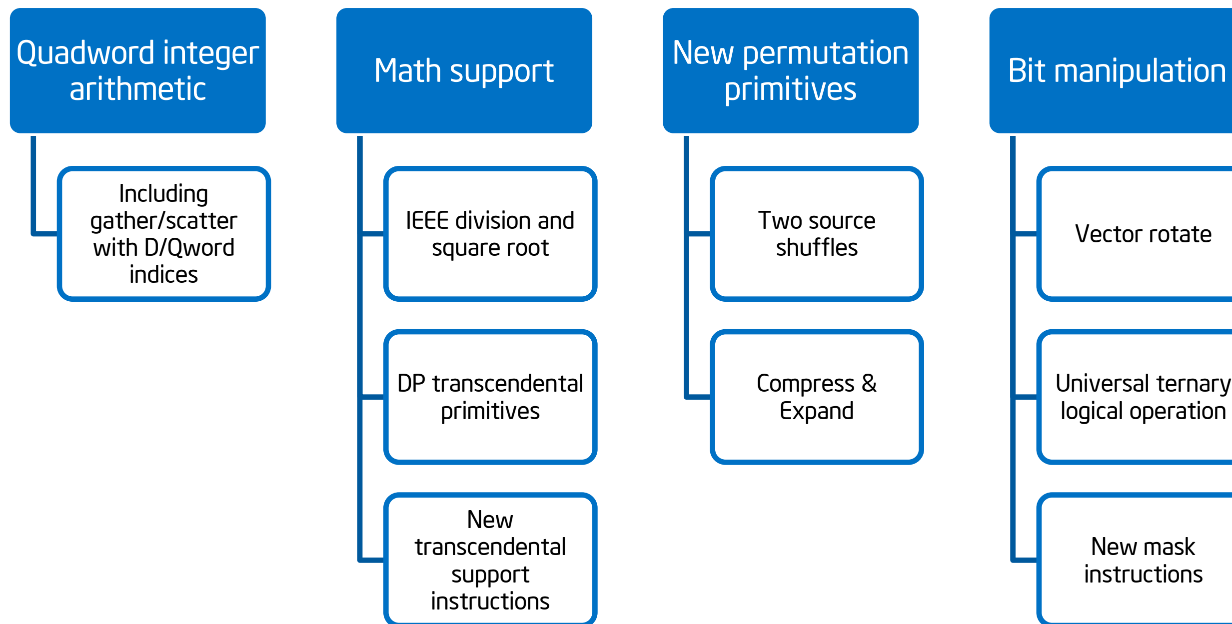
AVX-512 ERI & PRI: Available on Xeon Phi only

- 28-bit precision RCP, RSQRT and EXP transcendentals
- 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-512F Designed for HPC

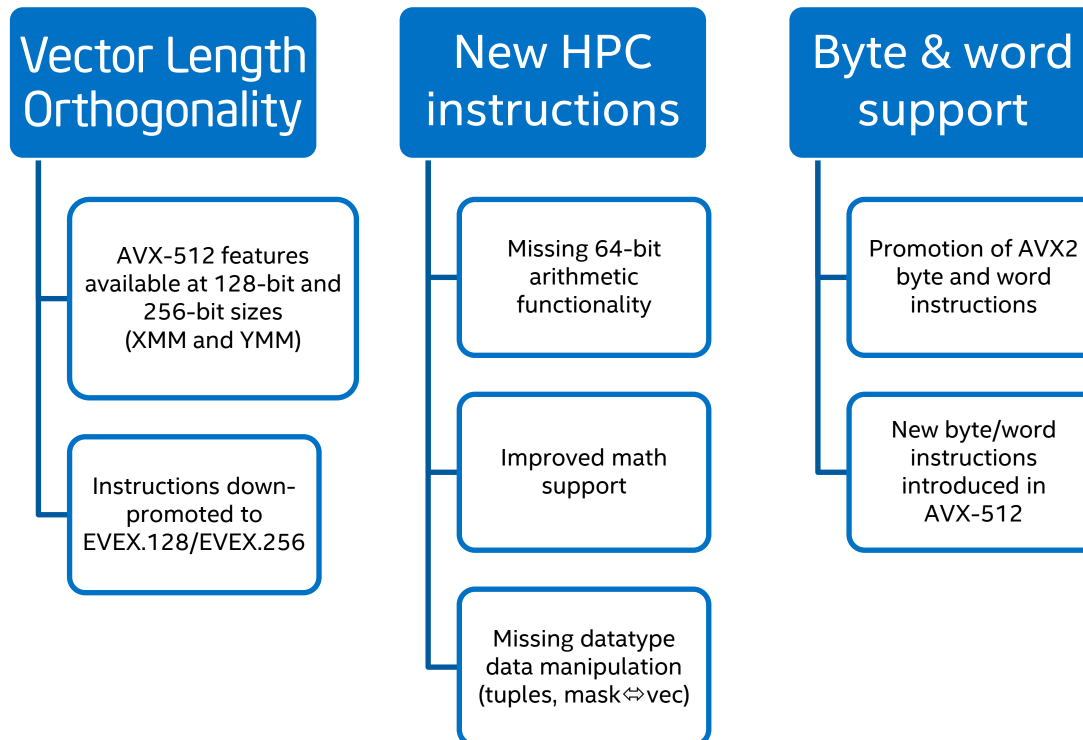
- Promotions of many AVX and AVX2 instructions to AVX-512
  - 32-bit and 64-bit floating-point instructions from AVX
    - Scalar and 512-bit
  - 32-bit and 64-bit integer instructions from AVX2
- Many new instructions to speedup HPC workloads



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

AVX-512{VL,DQ,BW}: Complements AVX-512F

Complete vector ISA extension shows up in Skylake Xeon
- *Main focus on simplifying the task of auto-vectorization for *any* compiler*
- Support for all data types: including 8-bit (byte) and 16-bit (word) integers
  - Useful for media and other workloads
- Support for all vector lengths
- Some instructions to speedup HPC workloads: closing KNL's AVX-512 gaps



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

AVX-512VL: Vector Length Orthogonality

Some algorithms are “natural” at certain element counts

- Scalar = 1 element count
- float4 = 128-bit
- word 4x4 (media) = 256-bit
- *32 registers / broadcast / masking cannot be retroactively added to AVX*

Auto-vectorization of loops with mixed datatypes

- Choose target for number of elements per iteration
  - 16 Single Precisions is one ZMM register, but…
  - 16 Words is a half a ZMM register aka YMM

But… why not just use the mask?

- potential mask bookkeeping overhead
- potential performance pitfalls now and in the future

Solution: Add vector length support for all AVX-512 packed instructions

- Every instruction is supported at 128-bit, 256-bit and 512-bit vector length
  - Ex: VADDPS xmm1 {k1}{z}, xmm2, xmm3 {1toN}

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

AVX-512DQ: New HPC ISA (vs AVX512F)

AVX-512 HPC (64)
VBROADCAST{F32X8,F64X2,I32X8,I64X2}
VBROADCAST{I32X2}
VEXTRACT{F32X8,F64X2,I32X8,I64X2}
VINSERT{F32X8,F64X2,I32X8,I64X2}
VCVT{,T}{PS,PD}2{QQ,UQQ}
VCVT{QQ,UQQ}2{PS,PD}
VCVT{,T}{PS,PD}2{QQ,UQQ}
VFPCLASS{PS,PD}
VRANGE{PS,PD}
VREDUCE{PS,PD}
VPMULLQ
K{AND,ANDN,OR,XNOR,XOR,NOT}B
K{MOV,ORTEST,SHIFR,SHIFTL}B
K{ADD,TEST}{B,W}
VPMOV{D2M,Q2M}, VPMOV{M2D,M2Q}

- VBROADCAST, VEXTRACT, VINSERT → Extended Tuple support: 32X8, 64X2, 32X2
- VCVT → Int64 ⇔ FP conversions
- VFPCLASS, VRANGE, VREDUCE → Transcendental package enhancements
- VPMULLQ → INT64 arithmetic support
- K{AND,…}B, K{MOV,…}B → Byte support for mask instructions
- K{ADD,TEST}{B,W}, VPMOV → Expanded mask functionality

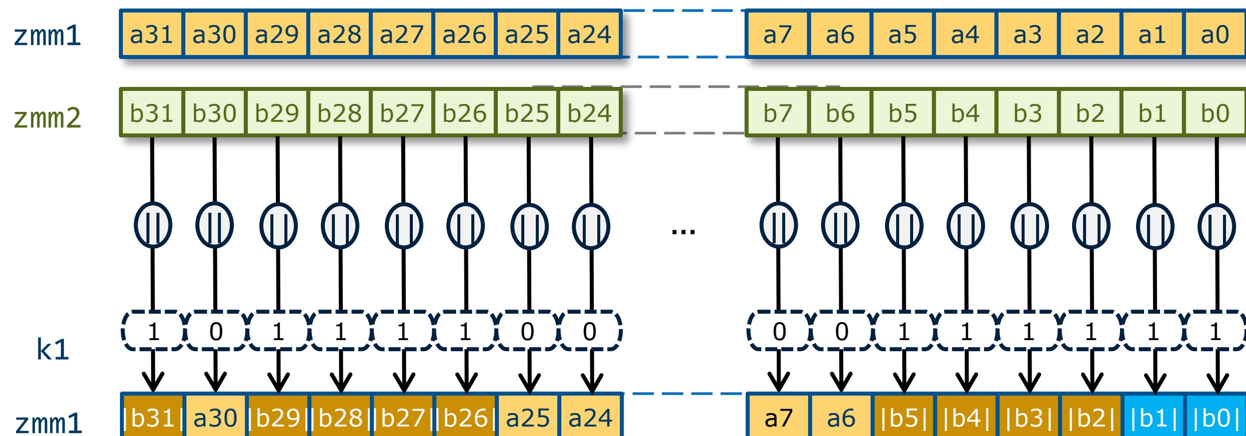
Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
intel

AVX-512BW: Byte and Word Support

131

AVX-512BW	AVX-512BW		
VPBROADCAST{B,W}	VMOVDQU{8,16}	KADD{D,Q}	VPMOV{SX,ZX}BW
VPSRLDQ, VPSLLDQ	VPBLENDM{B,W}	VPMOV{B2M,W2M,M2B,M2W}	VPMUL{HRS,H,L}W
VP{SRL,SRA,SLL}{V}W	{KAND,KANDN}{D,Q}	VPCMP{,EQ,GT}{B,W,UB,UW}	VPSADBW
VPMOV{WB,SWB, USWB}	{KOR,KXNOR,KXOR}{D,Q}	VP{ABS,AVG}{B,W}	VPSHUFB, VPSHUF{H,L}W
VPTESTM{B,W}	KNOT{D,Q}	VP{ADD,SUB}{,S,US}{B,W}	VP{SRA,SRL,SLL}{,V}{B,W}
VPMADW	KORTEST{D,Q}	VPALIGNR	VPUNPCK{H,L}{BW,WD}
VPABSDIFFW	KTEST{D,Q}	VP{EXTR,INSR}{B,W}	
VDBPSADBW	KSHIFT{L,R}{D,Q}	VPMADD{UBSW,WD}	
VPERMW, VPERM{I,T}2W	KUNPACK{WD,DQ}	VP{MAX,MIN}{S,U}{B,W}	

VPABSW zmm1 {k1}, zmm2



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

Enabling of AVX-512 in GNU toolchain

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

KNL support in GNU toolchain overview

Support in binutils (gas/objdump) available from v2.24

glibc tuning not done so far

- memcpy, memset etc.
- Use of transcendental instructions from AVX-512ERI

Basic support in GCC available from GCC 4.9.x (see next slides)

Embedded rounding control autogeneration is not going to be supported in GCC

- fe[get|set]round () is not acting as FP barrier in GCC

Usage of advanced encoding features supported in back-end only

- New meta-pattern called `define_subst' introduced from GCC 4.8.x
- Using `subst' embedded masking, broadcasting and embedded rounding control were easily described in the backend

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

AVX-512: New Patterns

Specific new patterns (est.)

# of instructions: 651 (w/ masking: 500, w/ rounding: 114, w/ msk and rnd: 100)

Total

(400 × 2) + (100 × 3) + (14 × 2) + (651 – 514) ≈ **1300**

~ 5000 new intrinsics

Solution: introduce `define_subst'

Generate new pattern from existing

E.g. add masking and rounding

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

Example (original pattern)

```
(define_insn "*<plusminus_insn><mode>3"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
    (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
```

Optimization Notice

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

Example (+mask)

```
(define_insn "*<plusminus_insn><mode>3"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
    (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
```

```
(define_insn "*<plusminus_insn><mode>3_mask"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
    (vec_merge:VF_AVX512
      (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                  (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
      (match_operand:VF_AVX512 3 "nonimmediate_or_const0_operand" "0C,0C")
      (match_operand:DI 4 "register_operand" "k,k")
```


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

Example(+rounding)

```
(define_insn "*<plusminus_insn><mode>3"
 [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
  (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
             (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
```

```
(define_insn "*<plusminus_insn><mode>3_mask"
 [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
  (vec_merge:VF_AVX512
   (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
              (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
   (match_operand:VF_AVX512 3 "nonimmediate_or_const0_operand" "0C,0C")
   (match_operand:DI 4 "register_operand" "k,k")
```

```
(define_insn "*<plusminus_insn><mode>3_round"
  [(parallel [(set (match_operand:VF_AVX512 0 "register_operand" "=x,x")
      (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                           (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
      (match_operand:SI 3 "const_4_to_8_operand" "n,n")
```


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

Example (+both)

```
(define_insn "*<plusminus_insn><mode>3"
 [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
  (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
            (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
```

```
(define_insn "*<plusminus_insn><mode>3_mask"
 [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
  (vec_merge:VF_AVX512
   (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
              (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
   (match_operand:VF_AVX512 3 "nonimmediate_or_const0_operand" "0C,0C")
   (match_operand:DI 4 "register_operand" "k,k")
```

```
(define_insn "*<plusminus_insn><mode>3_round"
  [(parallel [(set (match_operand:VF_AVX512 0 "register_operand" "=x,x")
      (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                           (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
      (match_operand:SI 3 "const_4_to_8_operand" "n,n")
```

```
(define_insn "*<plusminus_insn><mode>3_mask_round"
 [(parallel (set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
      (vec_merge:VF_AVX512
         (plusminus:VF_AVX512 (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                              (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
      (match_operand:VF_AVX512 3 "nonimmediate_or_const0_operand" "0C,0C")
      (match_operand:DI 4 "register_operand" "k,k")
  (match_operand:SI 3 "const_4_to_8_operand" "n,n")
```

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

define_subst (example)

```
(define_subst "mask"
  [(set (match_operand 0)
        (match_operand 1))]
  "TARGET_MASK"
  [(set (match_dup 0)
     (vec_merge:VF_512
      (match_dup 1)
      (match_operand:VF_512 2 "register_operand" "0C")
      (match_operand:<at> 3 "register_operand" "Yk"))))])
```

Parts of original pattern

Added to condition of new pattern

Constraints (duplicated for each alternative)

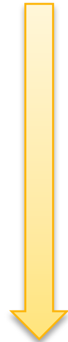
iterators

iterator_attribute

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

Example (using subst)

```
(define_insn "*<plusminus_insn><mode>3<mask_name>"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
        (plusminus:VF_AVX512
                (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
```



```
(define_insn "*<plusminus_insn><mode>3"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
        (plusminus:VF_AVX512
                (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm")))]
```

```
(define_insn "*<plusminus_insn><mode>3_mask"
  [(set (match_operand:VF_AVX512 0 "register_operand" "=x,v")
     (vec_merge:VF_AVX512
        (plusminus:VF_AVX512
                (match_operand:VF_AVX512 1 "nonimmediate_operand" "%0,v")
                (match_operand:VF_AVX512 2 "nonimmediate_operand" "xm,vm"))
        (match_operand:VF_AVX512 3 "nonimmediate_or_const0_operand" "0C,0C")
        (match_operand:DI 4 "register_operand" "Yk,Yk")
```

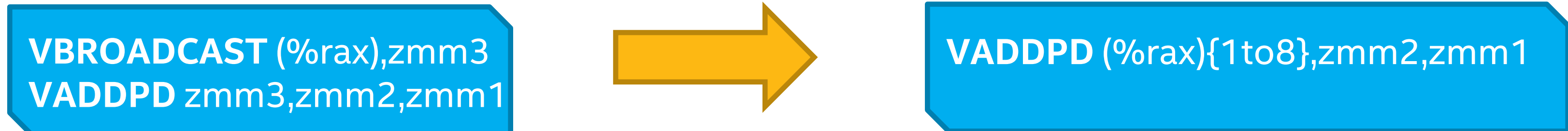
Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512: Embedded broadcasting

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

Embedded broadcasting support in GCC

GOAL



Implementation

Use substs to generate rtx patterns and rely on combiner

```
(define_subst "emb_bcst2"
  [(set (match_operand:BCST_V 0)
        (any_operator2:BCST_V
          (match_operand:BCST_V 1)
          (match_operand:BCST_V 2)))]
  "TARGET_AVX512F"
  [(set (match_dup 0)
       (any_operator2:BCST_V
           (vec_duplicate:BCST_V
               (match_operand:<ssescalarmode> 2 "memory_operand" "m"))
       (match_dup 1)))])
```


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

Results

- Internal implementation done
- Performance gain is 0%
- Combiner can't eliminate broadcasts that are
  - Have multiple destinations
  - Reside in different BBs
- State of the art embedded broadcasting (icc) shows little icount gain
- Impact on icache can't be measured without hardware

Conclusion

- Patch not submitted – no performance gain (for now?)

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

Support of new `scatter’ instruction family

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

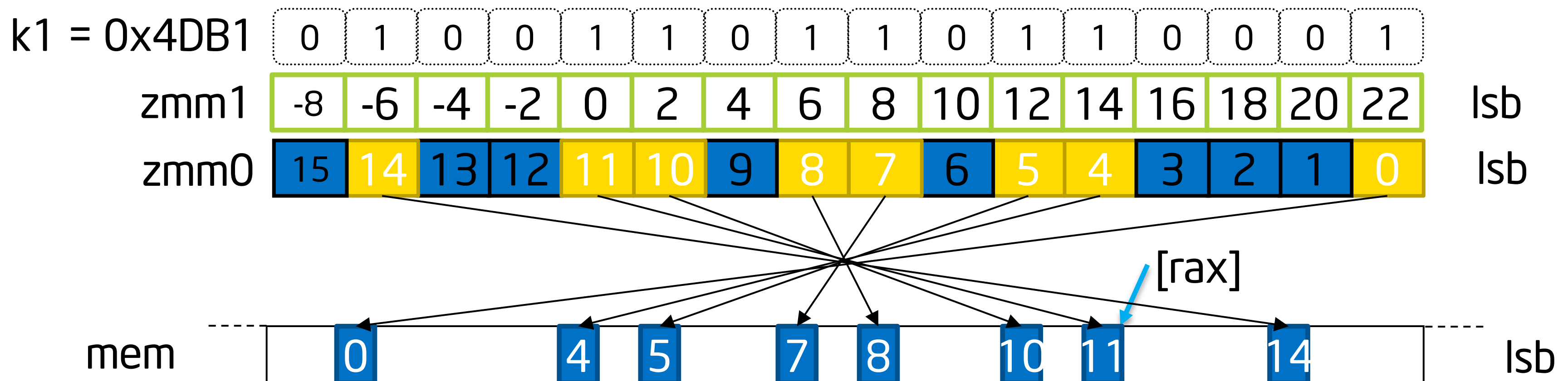
AVX-512: Scatters overview

Allows vectorization of loops with stores which addresses can be represented as:
Address [i] = BaseAddress + Index[i] * Scale

```
for(i=0; i<N; i++)
{
    A[B[i]+3] = C[i];
}
```

VPSCATTERDD zmm0, ([rax], zmm1, 4) {k1}

Stores up to 16 elements (controlled by mask) to the memory location pointed by base address, index vector and scale. For successfully stored elements corresponding mask bits will be set to zero.



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

Scatter Support in GCC

Patch which adds autogeneration of scatter instructions in case when one array is indexed by another array is ready

- Need to add autogeneration of scatter instructions for strided stores
- Expectations of performance improve from strided stores using scatters based on ICC 14:
  - SPEC 2006
    - 434.zeusmp – >1.5%
  - NPB 3.3.1-SER.ClassW – more than 1% of all executed instructions are scatters, so few percent of performance improve can be expected

Array indexing another array:

```
for(i=0; i<N; i++)
{
    A[B[i]+3] = C[i];
}
```

Strided store:

```
for(i=0; i<N; i++)
{
    A[5*i+3] = B[i];
}
```


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

512-bit auto-vectorization in GCC

7% icounts decrease (vs. AVX2 on SPECfp2006, -Ofast, ref data)

- New registers ZMM16–ZMM31 — 0.6% on SPECfp2006
- Extended standard patterns for the vectorizer:
  - Arithmetic and logic — 4.0% on SPECfp2006
  - expand_vector_init — 1.0% on SPECfp2006
  - FP division — 0.6% on SPECfp2006
  - FMA — 0.4% on SPECfp2006
  - copysign — 0.4% on SPECfp2006
  - cmp, vec_perm, unpack, extract, sqrt, rcp, floor, ceil, round, gather, reduction, etc.
- (No hotspots in SPEC CPU2006 benchmarks, where GCC can vectorize with VL=256, but can't vectorize with VL=512)

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

Embedded masking autogeneration

Most promising feature of new encoding (EVEX)

- Preliminary investigation performed
  - Not-for-trunk proof-of-concept patch implemented, which shows about 1.5% of icount decrease on average in SpecFP2006
  - Vectorized loop tails
  - Vectorization of loop heads looks promising as well
- Applicable for if-conv optimization
- Masking of operation, not result, hence no redundant side effects, exceptions, memory accesses etc.

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

Enabling of SKX in GNU toolchain

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

Enabling of SKX in GNU toolchain

New ISA were published on July'18 2014

- Patch set for support new ISA in binutils (gas/objdump) was submitted
- Branch with support for GCC was created (avx512-skx)
  - Extended existing patterns (i386/sse.md) to support AVX-512VL,BW,DQ
  - Set of intrinsics covering new ISA was implemented
  - … Covered by corresponding testsuite
  - Target for GCC 4.10.x
- No performance work was done so far
- glibc work was not performed

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

Backup

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

AVX-512 features (I): More & Bigger Registers

AVX: VADDPS YMM0, YMM3, [mem]

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

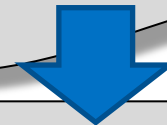
AVX-512: VADDPS ZMM0, ZMM24, [mem]

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

But you need many more features
to use all that real estate effectively…

```
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];
}
```

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

Why Separate Mask Registers?

Don’t waste away real vector registers for vector of booleans

Separate control flow from data flow

Boolean operations on logical predicates consume less energy (separate functional unit)

Tight encoding allows orthogonal operand

- Every instruction now has an extra mask operand

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*

MXCSR flag updates and fault handlers

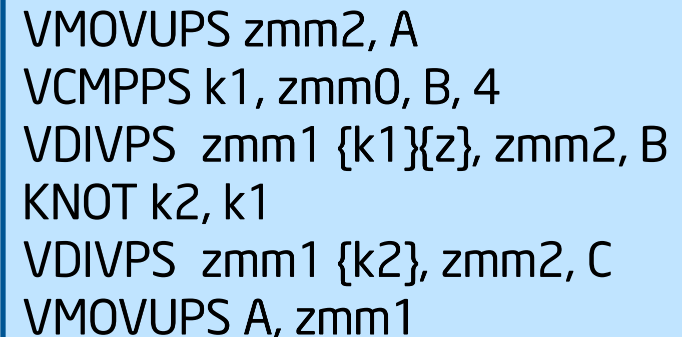
- Avoid spurious floating-point exceptions without having to inject neutral data

Zeroing/merging

- Use zeroing to avoid false dependencies in OOO architecture
- Use merging to avoid extra blends in if-then-else clauses (predication) for greater 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];
    }
}
```



Optimization Notice

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

AVX-512 Features: 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}, zmm2, [rax] {1toN} memory size operand is 64 bits
- Assembler/compiler reverts to 32-bit displacement when the real displacement is not a multiple

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

AVX-512 F: Common Xeon Phi (KNL) and Skylake Xeon Vector ISA Extension

AVX-512 Foundation is the common SIMD foundation for HPC software development
First on KNL
Planned on SKX (Skylake Xeon)

Optimization Notice

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

intel

Quadword Integer Arithmetic

Useful for pointer manipulation
64-bit becomes a first class citizen
Removes the need for expensive SW emulation sequences
Note: VPMULQ and int64 <-> FP converts not in AVX-512 F

Instruction	**Description**
`VPADDQ zmm1 {k1}, zmm2, zmm3`	INT64 addition
`VPSUBQ zmm1 {k1}, zmm2, zmm3`	INT64 subtraction
`VP{SRA,SRL,SLL}Q zmm1 {k1}, zmm2, imm8`	INT64 shift (imm8)
`VP{SRA,SRL,SLL}VQ zmm1 {k1}, zmm2, zmm3`	INT64 shift (variable)
`VP{MAX,MIN}Q zmm1 {k1}, zmm2, zmm3`	INT64 max, min
`VP{MAX,MIN}UQ zmm1 {k1}, zmm2, zmm3`	UINT64 max, min
`VPABSQ zmm1 {k1}, zmm2, zmm3`	INT64 absolute value
`VPMUL{DQ,UDQ} zmm1 {k1}, zmm2, zmm3`	32x32 = 64 integer multiply

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

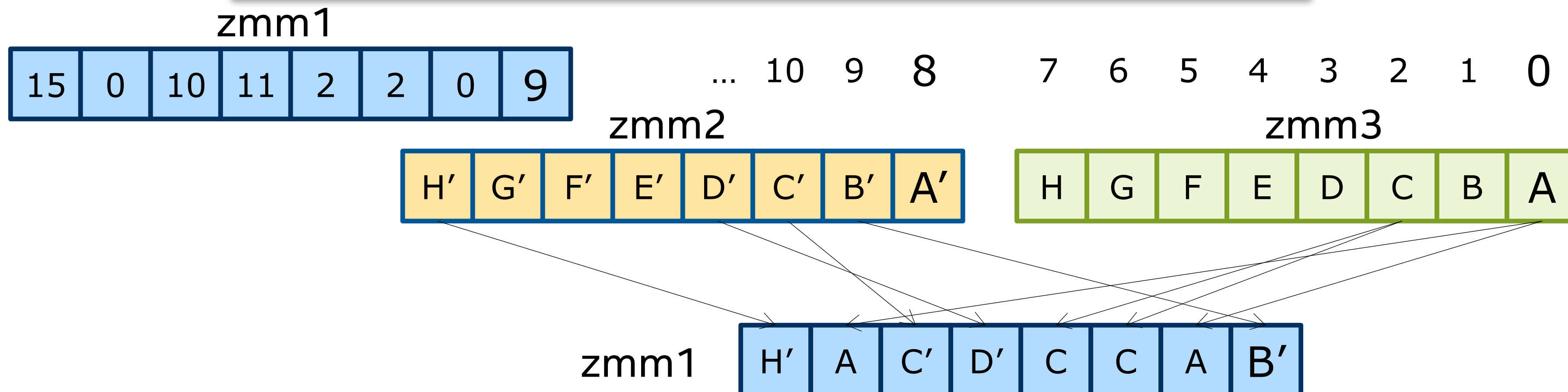
Instruction		
$VGETXEXP_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain exponent in FP format
$VGETMANT_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2	Obtain normalized mantissa
$VRNDSCALE_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, imm8	Round to scaled integral number
$VSCALEF_{\{PS,PD,SS,SD\}}$	zmm1 {k1}, zmm2, zmm3	$X*2^{y}$, 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.

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



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

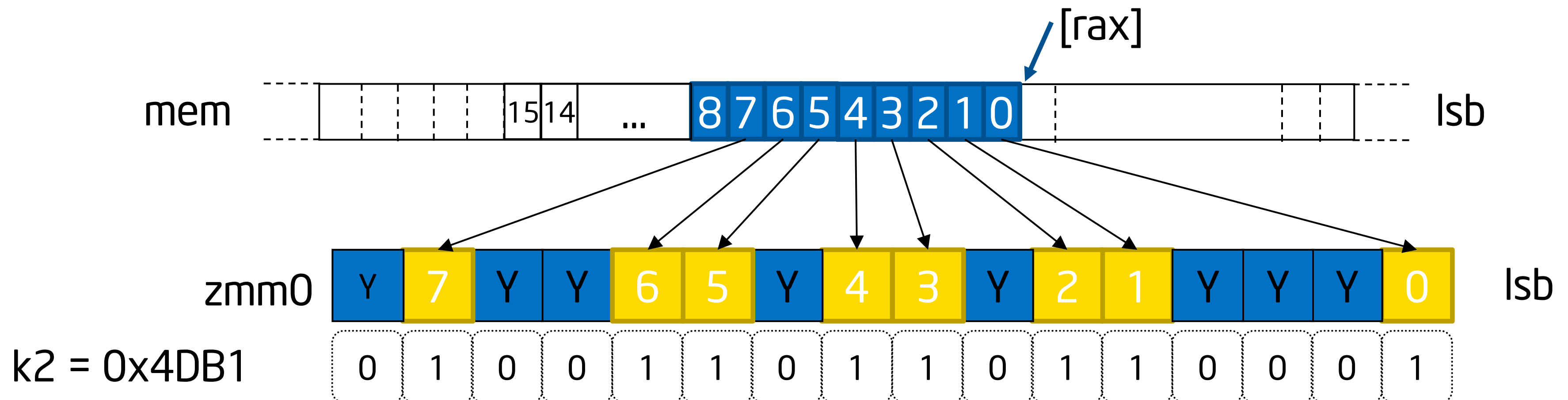
Expand & Compress

Allows vectorization of conditional loops
- Opposite operation (compress) in AXV-512
- 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



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.

VPTERNLOG – Ternary Logic Instruction

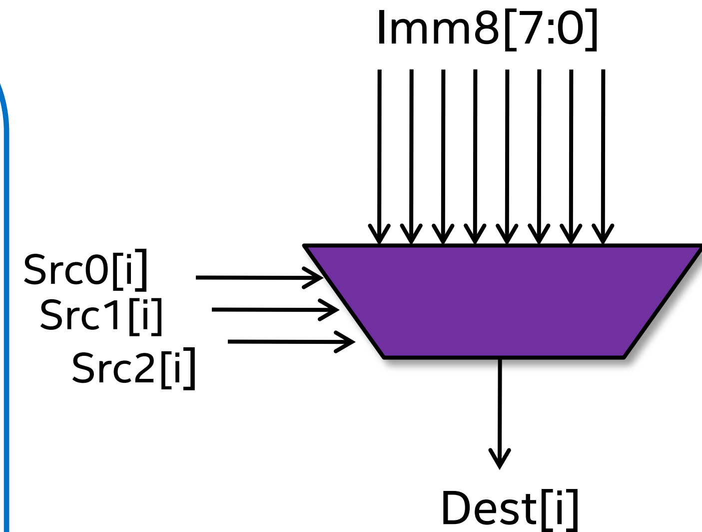
Mimics a FPGA cell

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



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

AVX-512 CDI: Conflict Detection Instructions

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.

Conflict Detection Instructions in AVX512

VPCONFLICT instruction detects elements with previous conflicts in a vector of indexes

- Allows to generate a mask with a subset of elements that are guaranteed to be conflict free
- The computation loop can be re-executed with the remaining elements until all the indexes have been operated upon

CDI instr.
VPCONFLICT{D,Q} zmm1{k1}, zmm2/mem
VPBROADCASTM{W2D,B2Q} zmm1, k2
VPTESTNM{D,Q} k2{k1}, zmm2, zmm3/mem
VPLZCNT{D,Q} zmm1 {k1}, zmm2/mem

```
index = vload &B[i]                              // Load 16 B[i]
pending_elem = 0xFFFF;                           // all still remaining
do {
   curr_elem = get_conflict_free_subset(index, pending_elem)
   old_val = vgather {curr_elem} A, index        // Grab A[B[i]]
   new_val = vadd old_val, +1.0                  // Compute new values
   vscatter A {curr_elem}, index, new_val        // Update A[B[i]]
   pending_elem = pending_elem ^ curr_elem       // remove done idx
} while (pending_elem)
```

This not even the fastest version: see backup for details

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

AVX-512 ERI & AVX-512 PRI: Xeon Phi Only

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

Xeon Phi Only Instructions

Set of segment-specific instruction extensions

- First appear on KNL
- Will be supported in all future Xeon Phi processors
- May or may not show up on a later Xeon processor

Address two HPC customer requests

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

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

AVX-512 ERI & PRI Description

CPUID	Instructions	Description
AVX-512 PRI	PREFETCHWT1	Prefetch cache line into the L2 cache with intent to write (RFO ring request)
	VGATHERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache
	VSCATTERPF{D,Q}{0,1}PS	Prefetch vector of D/Qword indexes into the L1/L2 cache with intent to write
AVX-512 ERI	VEXP2{PS,PD}	Computes approximation of 2^x with maximum relative error of 2^{-23}
	VRCP28{PS,PD}	Computes approximation of reciprocal with max relative error of 2^{-28}
	VRSQRT28{PS,PD}	Computes approximation of reciprocal square root with max relative error of 2^{-28}

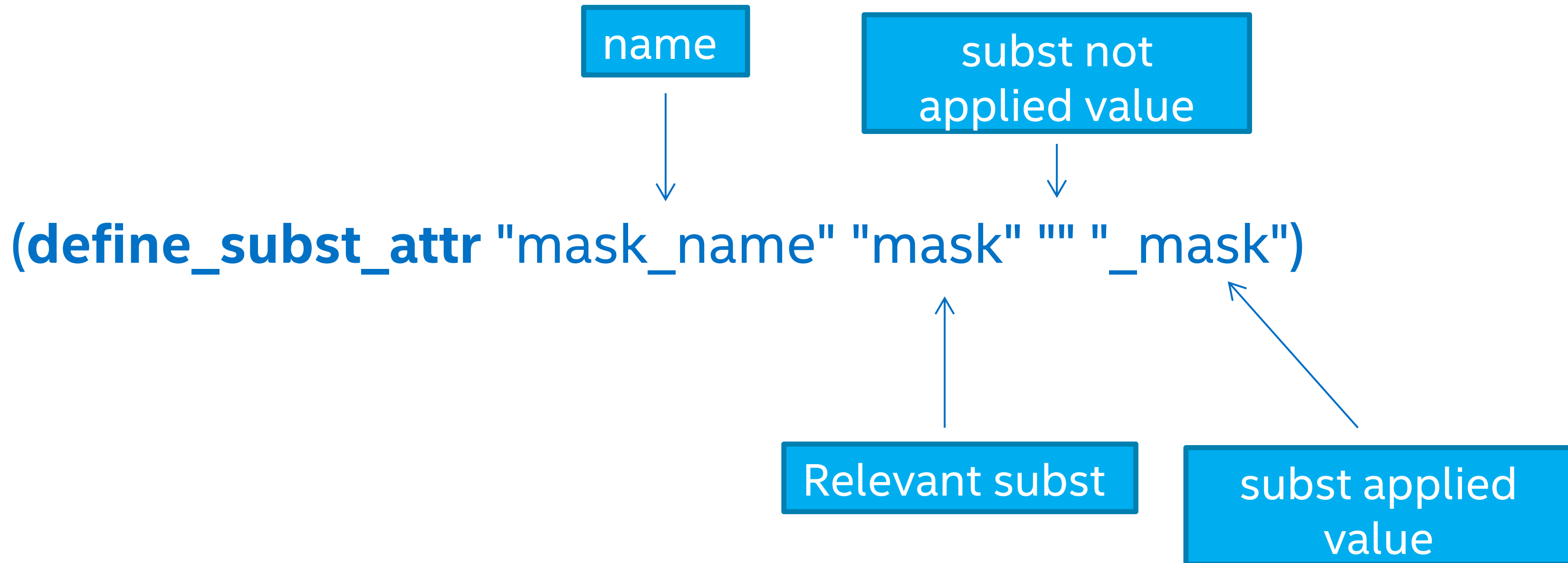
Optimization Notice
Copyright © 2014, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.

AVX-512 ERI & PRI Motivation

CPUID	Instructions	Motivation
AVX-512 PRI	PREFETCHWT1	Reduce ring traffic in core-to-core data communication
	VGATHERPF{D,Q}{0,1}PS	Reduce overhead of software prefetching: *dedicate side engine to prefetch sparse structures while devoting the main CPU to pure raw flops*
	VSCATTERPF{D,Q}{0,1}PS	
AVX-512 ERI	VEXP2{PS,PD}	Speed-up key FSI workloads: Black-Scholes, Montecarlo
	VRCP28{PS,PD}	Key building block to speed up most transcendental sequences (in particular, division and square root): Increasing precision from 14=>28 allows to reduce one complete Newton-Raphson iteration
	VRSQRT28{PS,PD}	

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

define_subst_attr



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

Summary

AVX512: new 512-bit vector ISA extension

- Common between Xeon (SKL) and Xeon Phi (KNL)

AVX512VL, AVX512DQ, AVX512BW: complements AVX512

- Shows up first on Skylake Xeon
- Provides support for all data types and vector lengths

Conflict detection new instructions

- Improves autovectorization
- Common to Xeon and Xeon Phi

TVX new instructions

- 28-bit transcendentals and new prefetch instructions
- On Xeon Phi only

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

Authors

- Mark Charney
- Jesus Corbal
- Roger Espasa
- Milind Girkar
- Moustapha Ould-ahmed-vall
- Ilya Tocar
- Bret Toll
- Bob Valentine
- Ilya Verbin
- Kirill Yukhin

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.

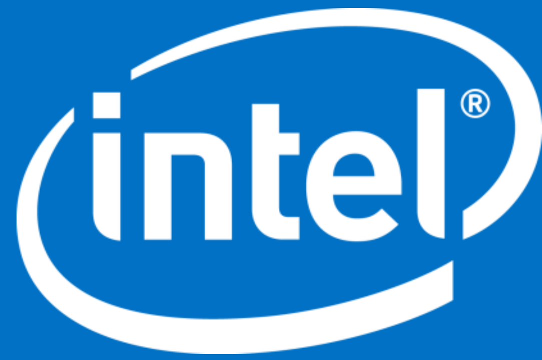
Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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

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


intel