

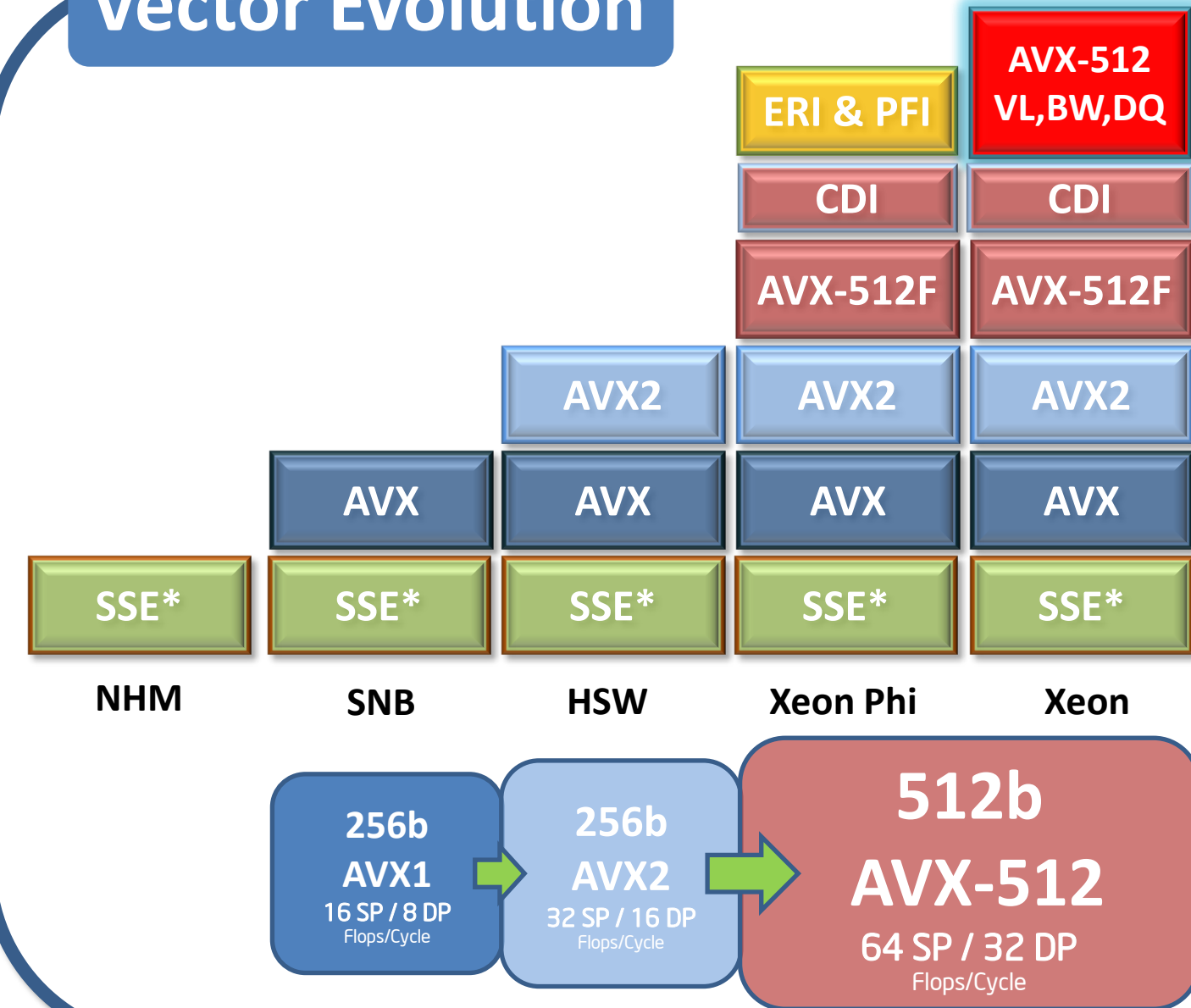


Intel® AVX-512 architecture evolution and support in Clang/LLVM

Robert.Khasanov
Zinovy.Y.Nis @intel.com

2014 LLVM Developers' Meeting, Oct 28-29

Vector Evolution



AVX-512 Features

- 512b-wide vectors(%ZMM0-31)
- Masked instructions, 64b-mask registers (%K0-7)
- Gatherers/Scatters
- Permutations
- Embedded broadcast
- Embedded rounding control
- Compressed displacement
- Embedded suppression of all exceptions



AVX-512 in Clang/LLVM

- Total: 651 instructions, 4000+ intrinsics
- 30% of these instructions implemented
- Encodings, lowering and intrinsics covered with tests
- 100+ patches, 9000+ LOCs
- Work in progress!

Available in trunk since July 2014 !

clang -march=knl...

clang -march=skx ...

New features

AVX-512VL: Vector Length Orthogonality

Apply **AVX-512F** instructions to **128b (%XMM)** and **256b (%YMM)** registers

- AVX-512F (starting with Xeon Phi)
 - VADDPD(%rcx), %zmm2, %zmm3
- AVX-512{F,VL} (starting with Skylake Xeon)
 - VADDPD(%rcx), %ymm2, %ymm3
 - VADDPD(%rcx), %xmm2, %xmm3

AVX-512BW: Byte & word support ¹³¹

AVX-512F packed instructions work on **double-** and **quad**words



AVX-512BW packed instructions work on **byte** and **words**



AVX-512BW also introduced 32b and 64b mask instructions

Perfect for handling graphics **R G B A**

AVX-512DQ: New HPC instructions ⁶⁴

Extended Tuple support:
32x8, 64x2, 32x2

INT64 arithmetic support

Int64 ⇔ FP conversions

Byte support for mask instructions

Transcendental package enhancements

Expanded mask functionality

Enabling compiler optimizations

If-Conversion with memory accesses

```
float A[N], B[N], C[N];
for(i=0; i<16; i++) {
  if (B[i] != 0) {
    A[i] = A[i] * B[i];
  }
}
```

With AVX-512 we can generate this smart code!

```
VCMPPNEQPS k1, zmm0, B
VMOVUPS zmm2 {k1}{z}, A
VMULPS zmm1 {k1}, zmm2, B
VMOVUPS A{k1}, zmm1
```

Masking instructions semantics

```
Zero-masking:
VMULPS zmm1 {k1}{z}, zmm2, B
destj ← maskj==1 ? src1j * src2j : 0

Merge-masking:
VMULPS zmm1 {k1}, zmm2, B
destj ← maskj==1 ? src1j * src2j : destj
```

Currently, this loop can't be vectorized in LLVM IR:

LV: Found a loop: for.body
LV: Can't if-convert the loop.
LV: Not vectorizing: Cannot prove legality.

Potential LLVM IR extended with special intrinsics for masking

```
%a= call <16 x float> @llvm.masked.load(<16 x float>* %a.ptr, <16 x i1> %mask, <16 x float> zeroinitializer)
%mul= call <16 x float> @llvm.masked.fmul(<16 x float> %a, %b, <16 x i1> %mask, <16 x float> %old_mul)
call void @llvm.masked.store(<16 x float> %mul, <16 x float>* %a.ptr, <16 x i1> %mask)
```

Vectorization of Peeled Loops

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

```
i = 0;
V = N - (N % VF); // VF is # of elements in vector
// Vector part
for (; i < V; i += VF) {
  // vectorized!
  C[i:i+VF-1:1] = A[i:i+VF-1:1] + B[i:i+VF-1:1];
}
// Peeled part
// Not vectorized!
for (; i < N; ++i) C[i] = A[i] + B[i];
```

```
// Vectorized
VMOVUPS ZMM1, ZERO_VEC
VADDPD ZMM1, ZMM1, A[0] // 0..7
VADDPD ZMM1, ZMM1, B[0]
VMOVUPS C[0], ZMM1
...
// Peeled part
// Clone of loop body but with masks
KMOVW K1, MASK // Here, MASK is N % VF
VMOVUPS ZMM1, ZERO_VEC
VADDPD ZMM1 {k1}, ZMM1, A[32] // 32..36
VADDPD ZMM1 {k1}, ZMM1, B[32]
VMOVUPS C[32] {k1}, ZMM1
```

More samples



Elena Demikhovskiy's
Intel® AVX-512
Architecture
review poster
@ 2013 LLVM DevMtg



Kirill Yukhin's
Intel® Advanced Vector
Extensions 2015/16
Support in GNU Compiler
Collection
@ GNU Tools Cauldron 2014

Legal Disclaimer

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

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. 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.



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