Streaming Store Instructions in the Intel® Xeon Phi™ coprocessor

New “streaming stores” instructions introduced in B0 Si:

Ex: VMOVNRNGOAPS/VMOVNRNGOAPD

- These instructions are intended to speed up performance in the case of vector-aligned unmasked stores in streaming kernels where we want to avoid wasting memory bandwidth by being forced to read the original content of entire cache line from memory when we overwrite their whole content completely

Compiler Behavior

Starting with Composer XE 2013 Update 1 compiler, the compiler **default** has been changed to generate VMOVNRNGO instructions for streaming stores under certain situations

- User can provide hints to the compiler on when to generate these
- See next slide for details

External option to disable generating these instructions:
-opt-streaming-stores *never*

Heuristics for streaming stores

Compiler generates streaming store instructions only when:

- Compiler is able to vectorize the loop and generate an aligned unit-strided vector unmasked store:
  - If the store accesses in the loop are aligned properly, user can convey alignment information using pragmas/clauses
  - Ex: Use #pragma vector aligned OR !DEC$ vector aligned before loop to convey alignment of all memory refs inside loop including the stores
  - In some cases, even when there is no pragma to align the store-access, the compiler may align the store-access at runtime using a dynamic peel-loop based on its own heuristics
  - Based on alignment analysis, compiler could prove that the store accesses are aligned (at 64 bytes)
  - Store has to be aligned and be writing to a full cache line (vstore – 64 bytes, no masks)
    - Note that it is the responsibility of the user to align the data appropriately at allocation time using align clauses, aligned_malloc, "-align array64byte" option on Fortran, etc.
- Vector-stores are classified as nontemporal using one of:
  - User has specified a nontemporal pragma on the loop to mark the vector-stores as streaming
    - #pragma vector nontemporal (in C/C++) OR !DEC$ vector nontemporal (in F) before loop to mark aligned stores
    - Or communicate nontemporal-property of store using "#pragma vector nontemporal A" where "A[i] = …" is the store inside the loop
  - User has specified the compiler option "-opt-streaming-stores always" to force marking ALL aligned vector-stores as nontemporal
    - Has the implicit effect of adding the nontemporal pragma to all loops that are vectorized by the compiler in the compilation scope
    - Using this option on KNC has few negative consequences since the data remains in the L2 cache (just not in the L1 cache) – so this option can be used if most aligned vector-stores are nontemporal
    - Using this option on Xeon for cases where some accesses are temporal can cause significant performance losses since the streaming-store instructions on Xeon bypass the cache altogether
  - Fully automatic heuristic that will kick in when the loop has a constant large trip-count (known to the compiler)
    - Compiler will also generate a memory-fence after the loop in this case

On KNC, compiler generates streaming stores if conditions listed above are satisfied

Study the output of –vec-report6 to check whether store is aligned and whether streaming stores are generated

Optimization Notice

Streaming Stores Code Generation

No compiler-inserted prefetches will be generated for the store cache-line

- Prefetches would cause a read of the cache-line before the store, negating the bandwidth-saving benefits of streaming stores

Compiler also generates a L2 clevict instruction for the store cache-line immediately after the store

- Use option –opt-streaming-cache-evict=0/1/2/3 to control the clevicts for performance tuning
- The option specifies cache eviction level when streaming loads/stores are used:
  - –opt-streaming-cache-evict =0 implies no clevict generated
  - –opt-streaming-cache-evict =1 implies L1 clevict generated after streaming store
  - –opt-streaming-cache-evict =2 (compiler default) implies L2 clevict generated after streaming store
  - –opt-streaming-cache-evict =3 implies L1 and L2 clevict generated after streaming store

Compiler inserts a memory-fence after the loop when:

- The streaming-store ngo version is generated purely based on compiler heuristics
- If nontemporal pragma or –opt-streaming-stores always option is specified, compiler expects user to do the appropriate fences

Optimization Notice

KNC Streaming Store Controls

The compiler behavior can be further controlled via internal optimization parameter “knc_stream_store_controls”, which can be set as follows:

- Use internal option -mGLOB_default_function_attrs="knc_stream_store_controls=value‟
  - Please note that behavior/semantics of internal options may change in future compilers
- Compiler default is: knc_stream_store_controls=0x42
- To pass the internal option to an offload compilation, use -offload-option,mic,compiler,"-mGLOB_default_function_attrs=knc_stream_store_controls=0x2"

Here “value” is a bitmask with the following semantics:

- Bit #0 (value = 1). Generate store.nr for non-temporal stores. The compiler will generate store.nr under the conditions described in slide 3.
- Bit #1 (value = 2). Generate store.nr.ngo for non-temporal stores. The compiler will generate store.nr.ngo under the conditions described in slide 3.
- Bit #2 (value = 4). Generate store.nr for all aligned vector unit-strided unmasked stores.
- Bit #3 (value = 8). Generate store.nr.ngo for all aligned vector unit-strided unmasked stores.
- Bit #4. Reserved for future.
- Bit #5. (value =0x20) If this bit is unset, compiler will skip generating ngo stores any-time there is a dependence between the store and a load inside the loop. If bit is 1, vectorizer goes ahead and marks the streaming-stores even if there are dependences involving the store.
- Bit #6. (value=0x40) If this bit is set, compiler will insert a memory-fence after the loop for the cases where store.nr.ngo stores are generated purely based on completely automatic compiler heuristics. If the user has specified the nontemporal pragma/directive OR the “-opt-streaming-stores always” option, compiler does NOT generate the fence even when this bit is set.

Option is available for both C and Fortran and behaves similarly

Optimization Notice

Source code

```
scellrb5% cat t5.c
void simple_triad(double *restrict a,
 double *b, double *c, double *d, int N)
{
    int i;
#pragma vector aligned nontemporal
    for (i=0; i<N; i++)
        a[i] = b[i] + c[i]*d[i];
}
scellrb5% icc -O2 -mmic -vec-report6 t5.c -
c -restrict -S
t5.c(7): (col. 2) remark: vectorization
support: reference a has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference b has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference c has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference d has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: streaming store was generated for
a.
t5.c(6): (col. 5) remark: LOOP WAS
VECTORIZED.
```

Generated asm for core-loop

```
..B1.4:                         # Preds ..B1.4
    ..B1.3 Latency 45
        vmovapd    (%rdx,%r11,8), %zmm1
        vmovapd    (%rcx,%r11,8), %zmm0
        vfmadd213pd (%rsi,%r11,8), %zmm0, %zmm1
        vmovnrngoaps %zmm1, (%rdi,%r11,8)
        clevict1   (%rdi,%r11,8)
        vprefetch1 512(%rsi,%r11,8)
        movb       %al, %al
        vprefetch0 256(%rsi,%r11,8)
        movb       %al, %al
        vprefetch1 512(%rdx,%r11,8)
        movb       %al, %al
        vprefetch0 256(%rdx,%r11,8)
        movb       %al, %al
        vprefetch1 512(%rcx,%r11,8)
        movb       %al, %al
        vprefetch0 256(%rcx,%r11,8)
        addq       $8, %r11
        cmpq       %r10, %r11
        jb         ..B1.4           # Prob 82%
```

- No fence after the loop, no prefetch for a
- nrngo and clevict1 for a[i] store

Notice

Source code

```
scellrb5% cat t5.c
#define SIZE 500000
void simple_triad(double * restrict a,
double *b, double *c, double *d)
{
    int i;
#pragma vector aligned
    for (i=0; i<SIZE; i++)
              a[i] = b[i] + c[i]*d[i];
}
scellrb5% icc -S -mmic -vec-report6 t5.c
-c -restrict -opt-streaming-cache-evict=0
t5.c(7): (col. 2) remark: vectorization
support: reference a has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference b has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference c has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: reference d has aligned access.
t5.c(7): (col. 2) remark: vectorization
support: streaming store was generated for
a.
t5.c(6): (col. 5) remark: LOOP WAS
VECTORIZED.
```

Generated asm for core-loop

```
..B1.2:..B1.1 Latency 45
        vmovapd    (%rdx,%rax,8), %zmm1
        vmovapd    (%rcx,%rax,8), %zmm0
        vfmadd213pd (%rsi,%rax,8), %zmm0, %zmm1
        vmovnrngoaps %zmm1, (%rdi,%rax,8)
        vprefetch1 512(%rsi,%rax,8)
        movb       %dl, %dl
        vprefetch0 256(%rsi,%rax,8)
        movb       %cl, %cl
        vprefetch1 512(%rdx,%rax,8)
        movb       %cl, %cl
        vprefetch0 256(%rdx,%rax,8)
        movb       %bl, %bl
        vprefetch1 512(%rcx,%rax,8)
        movb       %dl, %dl
        vprefetch0 256(%rcx,%rax,8)
        addq       $8, %rax
        cmpq       $500000, %rax
        jb         ..B1.2
..B1.3:
        lock
        addl       $0, (%rsp)
```

- Fence generated after loop since ngo was generated with no user-help (no nontemporal)
- No prefetch for a, nrngo store generated
- No clevict1/clevict0 based on evict option

Optimization Notice

Fortran Source code

```
scellrb5% cat t7.f90
subroutine sub1(a, b, c, d, len, n1, n2)
      real(8) a(len,len), b(len,len),
c(len,len), d(len,len)
      integer i, j, len
!OMP$ parallel for
      do j = 1,n1
!DEC$ vector aligned nontemporal
      do i = 1,n2
         a(i,j) = 2*b(i,j)
         c(i,j) = d(i,j) * b(i,j)
      enddo
      enddo
      end

scellrb5%  ifort -O2 -vec-report6 t7.f90 -mmic
-openmp -S
t7.f90(11): (col. 10) remark: vectorization
support: reference a has aligned access.
t7.f90(11): (col. 10) remark: vectorization
support: reference b has aligned access.
t7.f90(12): (col. 3) remark: vectorization
support: reference c has aligned access.
t7.f90(12): (col. 3) remark: vectorization
support: reference d has aligned access.
t7.f90(12): (col. 3) remark: vectorization
support: reference b has aligned access.
t7.f90(11): (col. 10) remark: vectorization
support: streaming store was generated for a.
t7.f90(12): (col. 3) remark: vectorization
support: streaming store was generated for c.
t7.f90(10): (col. 7) remark: LOOP WAS
VECTORIZED.
```

Generated asm for core-loop

```
..B1.7:
        vmulpd      (%rbx,%r12), %zmm0, %zmm1
        addq        $8, %r15
        vmovapd     (%rbx,%r12), %zmm2
        vmovnrngoaps %zmm1, (%rbx,%r11)
        clevict1    (%rbx,%r11)
        vmulpd      (%rbx,%r13), %zmm2, %zmm3
        vmovnrngoaps %zmm3, (%rbx,%r14)
        clevict1    (%rbx,%r14)
        vprefetch1  (%rbx,%r10)
        movb        %al, %al
        vprefetch1  (%rbx,%rdi)
        addq        $64, %rbx
        cmpq        %rsi, %r15
        jb          ..B1.7
```

- No fence after the loop, no prefetches for a,c
- nrngo and clevict1 for stores to a,c

Notice

Intrinsics for streaming stores

```
/*
* Store aligned float32/float64 vector with No-Read hint.
*/

extern void __ICL_INTRINCC _mm512_storenr_ps(void*, __m512);
extern void __ICL_INTRINCC _mm512_storenr_pd(void*, __m512d);

/*
* Non-globally ordered store aligned float32/float64 vector with No-Read hint.
*/

extern void __ICL_INTRINCC _mm512_storenrngo_ps(void*, __m512);
extern void __ICL_INTRINCC _mm512_storenrngo_pd(void*, __m512d);
```

Software

Optimization Notice

Optimization Notice

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

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.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, reference www.intel.com/software/products.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Atom, Centrino Atom Inside, Centrino Inside, Centrino logo, Cilk, Core Inside, FlashFile, i960, InstantIP, Intel, the Intel logo, Intel386, Intel486, IntelDX2, IntelDX4, IntelSX2, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skoool, Sound Mark, The Journey Inside, Viiv Inside, vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

http://intel.com/software/products

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