Compiler Prefetching on KNL

Rakesh Krishnaiyer

Principal Engineer

Intel Compiler Lab

Data Prefetching Support in icc/ifort

Regular array accesses

Pointer accesses similar to array accesses where the address can be predicted in advance

Supports address calculations that involve:

- Affine functions of surrounding loop indices
- More complicated access-patterns that require insertion of additional instructions inside the loop

Prefetch for indirect references (a[b[i]], b[i]->field1, ptr->field3->field4, etc.)

For data-accesses in inner-loops with a surrounding loop-nest, compiler decides whether to prefetch for a future iteration of the inner-loop or the outer-loop. Heuristics use parameters such as:

- Trip-count estimates of inner and outer loops
- Symbolic contiguity-analysis of data-accesses inside the inner loop

Prefetches issued for memory-references in any loop-level, distances calculated taking inner-loops into account

Multi-versioning of loops based on trip-counts for better prefetching

- Use of streaming hints if the data-size accessed will exceed cache-size

Prefetch distance calculation takes TLB pressure into account

Data Prefetching (contd.)

Prefetches may be issued for first few cache-lines accessed in the loop

- These initial-value prefetches are issued before entering the loop

Directives to fine-tune prefetching in loops

- Whether or not prefetches should be issued in the loop
- Ability to specify prefetch hint and distance for individual array accesses

Wide range of prefetch-related internal options to fine-tune the prefetch parameters for different configurations/micro-architectures

Compiler Prefetching in 18.0 Product

Basic compiler prefetching **NOT turned on by default** with option –xmic-avx512

- **Requires explicit options:**
  - **Ex: -O3 –xmic-avx512 –qopt-prefetch=<n>**
  - All three options must be present (no pfing done at –O2 –qopt-prefetch …)
  - You can replace –xmic-avx512 with other targets such as –xCORE-AVX512 or –xcore-avx2
    - Caveat: Indirect prefetches are supported only for AVX512 and above
  - To enable prefetching at any opt-level, one can use the internal option –mP2OPT_hlo_prefetch_level=<n>
    - Ex: -O2 –xmic-avx512 –mP2OPT_hlo_prefetch_level=2

One prefetch instruction (prefetcht0 – hint 0) will be issued per memory reference in loop when icc/ifort decides to issue a prefetch for that memory reference

Notice the difference from what we had for KNC:

- On KNC, -qopt-prefetch=3 was default at opt-level –O2 (with –mmic option)
  - Had to say –qopt-prefetch=0 explicitly to turn off prefetching
  - Two prefetches per memory reference was default
  - In addition, initial value prefetches generated before loop was default

-qopt-prefetch=<n> Levels Explained

n=0 is the default if you omit -qopt-prefetch option

- No prefetches will be issued

n=2 is the default if you just say -qopt-prefetch with no explicit "n" argument

- Insert prefetches only for direct references where the compiler thinks hardware prefetcher may not be able to handle it

n=3 will turn on prefetching for all direct memory references without regard to hardware prefetcher

- n=4 is same as n=3 (currently)

n=5 additional prefetching for all indirect refs (AVX512 & above)

- Indirect prefetches (hint 1) done using AVX512-PF gatherpf instructions on KNL
- On SKX, 8 prefetch instrs issued instead of 1 gatherpf instruction
- Extra prefetches issued for strided vector accesses (hint 0) to cover all cache-lines

Software Prefetching Heuristics

Software PF techniques to minimize overlap with H/W PF (-qopt-prefetch=2 option):

- Loops with lots of memory refs that lead to a large number of streams that the HW prefetcher cannot handle.
- Loops with memory-references that have large (or unknown) strides - HW can handle only accesses within a 4K page
- Loops that have a very large trip-count (known to the compiler statically or based on runtime versioning)
  - Also includes memory accesses in loopnests that are accessed in a contiguous fashion (that compiler can prove) across outer-loops
  - Prefetch references in outer-loops since HW prefetcher is unlikely to handle it well, especially if there are lots of references in inner-loops or if inner-loop trip-counts are high

Other Prefetch Features

Support for option: –qopt-prefetch-distance=n1[,n2]

- Specify custom distance for single level prefetch:
  - -qopt-prefetch-distance=n1
- Two-level prefetching done if both n1 and n2 specified in option

Full support for prefetch pragmas

- #pragma noprefetch
- #pragma prefetch a:1:16 // prefetch var:hint:dist
- #pragma prefetch a:0:6 // hint 0, distance 6 vectorized iters
- #pragma prefetch *:1:24 // all mem-refs inside loop, dist=24

- !dir$ noprefetch
- !dir$ prefetch a:1:16
- !dir$ prefetch a:0:6
- !dir$ prefetch *:1:24

Internal Prefetch Options

-mP2OPT_hlo_prefetch_level=<n> is the internal option that enables prefetching (You can add this at –O2)

- -qopt-prefetch=<n> gets translated to this by the compiler driver

If you want to turn on prefetching for indirect references + prefetching for references where hardware prefetcher won't overlap:

- Use "-O3 –xmic-avx512 –qopt-prefetch=2 –mP2OPT_hlo_pref_indirect_refs=T"

To get the extra prefetching for strided refs, but no indirect prefetches, do:

- "-O3 –xmic-avx512 –qopt-prefetch=3 -mP2OPT_hlo_pref_multiple_pfes_strided_refs=T

By default, compiler issues t0 hint for direct prefetches. To change this to t1 hint, add:

- "-mP2OPT_hlo_pref_hint=1 –mP2OPT_hlo_pref_int_hint=1"

- Internal options should only be used for performance exploration – open compiler feature request to create external option if it really helps your use case

Indirect Prefetch: Internal Options

Internal compiler options to enable and fine-tune insertion of indirect prefetches by the compiler for all loops:

- **-mP2OPT_hlo_pref_indirect_refs=T/F**
  - Enable (T) / disable (F) indirect prefetches, default is F.
  - **In most cases, enabling this option alone is enough**
  - **(-qopt-prefetch=5 enables this)**
- -mP2OPT_hlo_use_const_indirect_pref_dist=<n>
  - Constant prefetch distance used for indirect prefetches, default value of n=2
- -mP2OPT_hlo_pref_indirect_hint=<0/1>
  - Prefetch hint used for indirect prefetches, default hint is 1
- -mP2OPT_hlo_pref_max_indirect_pfes=<n>
  - Maximum number of indirect prefetches that can be issued per loop, value of 0 means no limit. Default is 0
- -mP2OPT_hlo_pref_insert_bound_check_for_indirect_refs=T/F
  - Enable (T) / disable (F) bound check for indirect prefetches. If disabled, compiler assumes that index arrays are sufficiently padded to not cause any access violations. Default is T

Prefetch Reporting

-qopt-report=<n> where n is 2,3,4 or 5

No need to specify explicit phase, default phase is “ALL”

- Prefetch reporting is part of –qopt-report-phase=loop
- No prefetch reporting under –qopt-report-phase=vec

Loop Prefetch Example1(Indirect Ref)

```
scellrb9% cat sub1.c
  #include <stdio.h>
  void product(int jstart, int jend, int jincr, int *start_of_row, double *b,
               double *d, double *matrix, int *column, double *vector,
               double *result)
  {
    int i, row;
    double tmpb;

    for (row = jstart; row <= jend; row += jincr)
    {
      result[row] = 0.0;
      tmpb = b[row];
      for (i=start_of_row[row]; i<start_of_row[row+1]; i++)
      {
        tmpb += matrix[i] * vector[column[i]]; // direct references in innermost-loop
      }
      result[row] = tmpb/d[row];
    }
  }
```

Loop Prefetch Example1 contd

```
icc -O3 -xmic-avx512 -qopt-prefetch=3 -qopt-report=3 -qopt-report-file=stderr -S -unroll0 -restrict -std=c99
sub1.c -qopt-report-phase=loop,vec
LOOP BEGIN at sub1.c(10,3)
   remark #15542: loop was not vectorized: inner loop was already vectorized
   remark #25018: Total number of lines prefetched=4
   remark #25035: Number of pointer data prefetches=4, dist=7
…
   LOOP BEGIN at sub1.c(15,5)
      remark #15300: LOOP WAS VECTORIZED
      remark #15450: unmasked unaligned unit stride loads: 2
      remark #15462: unmasked indexed (or gather) loads: 1
      remark #25018: Total number of lines prefetched=2
      remark #25035: Number of pointer data prefetches=2, dist=8
   LOOP END

scellrb9% grep prefetch sub1.s
        prefetcht0 512(%r10,%r12)                  #17.15 c1
        prefetcht0 256(%r14,%r11)                  #17.34 c11
        prefetcht0 (%r10,%r9,8)                  #12.5 c51 // Outer loop prefetches
        prefetcht0 (%r11,%r9,8)                  #13.12 c55
        prefetcht0 (%r12,%r9,4)                  #15.12 c61 stall 1
        prefetcht0 (%r13,%r9,8)                  #19.24 c65 stall 1
```

- Only the direct references prefetched in innermost loop (and outer loop) at –qopt-prefetch=3
- At level=2, no pfes in innermost loop, but some pfes in outer loop

Loop Prefetch Example contd2

```
icc -O3 -xmic-avx512 -qopt-prefetch=5 -qopt-report=3 -qopt-report-file=stderr -S -unroll0 -restrict -std=c99
sub1.c -qopt-report-phase=loop,vec

LOOP BEGIN at sub1.c(10,3)
   remark #25018: Total number of lines prefetched=4
   …
   LOOP BEGIN at sub1.c(15,5)
      remark #15300: LOOP WAS VECTORIZED
      remark #25033: Number of indirect prefetches=1, dist=2
      remark #25035: Number of pointer data prefetches=2, dist=8
      remark #25540: Using gather/scatter prefetch for indirect memory reference, dist=2   [ sub1.c(17,27) ]
      remark #25143: Inserting bound-check around lfetches for loop
   LOOP END

scellrb9% grep prefetch sub1.s
        prefetcht0 512(%r11,%r13)                  #17.15 c1 // inner loop direct pfes
        prefetcht0 256(%r12,%r14,4)                 #17.34 c3
        prefetcht0 (%r10,%r9,8)                   #12.5 c51 // outer loop prefetches
        prefetcht0 (%r11,%r9,8)                   #13.12 c55
        prefetcht0 (%r12,%r9,4)                   #15.12 c61 stall 1
        prefetcht0 (%r13,%r9,8)                   #19.24 c65 stall 1
scellrb9% grep gatherpf sub1.s
          vgatherpf1dpd (%rbx,%ymm4){%k1}         #17.27 c3 // inner loop indirect prefetch
```

- At level=5, both direct and indirect refs prefetched in innermost loop
- Indirect prefetch done using AVX512 PF gatherpf instruction (hint 1)

Loop Prefetch Example2 (Strided Refs)

scellrb9% **cat t2_restrict.c**

```
void foo(int* data, int * restrict res, int n, int xi, int yi, int zi)
{
  int j, xij, yij, zij;

  for (j = 0; j < n; ++j) {
    xij = xi - data[3 * j];
    yij = yi - data[3 * j + 1];
    zij = zi - data[3 * j + 2];

    res[j] = xij * xij + yij * yij + zij * zij;
  }
}
```

```
scellrb9%:/home/cmplr/proj9/rkrish3/test3/pf_hint_dist_stuff/strided_pf% grep prefetch
   t2_restrict.s
# mark_description "-O3 -xmic-avx512 -qopt-prefetch=5 -qopt-report=4 -qopt-report-
   file=stderr -c -S -restrict";
        prefetcht0 1536(%rdi,%r14,4)                    #6.16 c1
        prefetcht0 1600(%rdi,%r14,4)                    #6.16 c11
        prefetcht0 1664(%rdi,%r14,4)                    #6.16 c15
        prefetcht0 512(%r12,%rsi)                      #10.5 c23
```

Loop Prefetch Example2 contd

```
LOOP BEGIN at t2_restrict.c(5,3)
   remark #15389: vectorization support: reference res[j] has unaligned access
[ t2_restrict.c(10,5) ]
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15415: vectorization support: non-unit strided load was generated for the variable
<data[j*3]>, stride is 3    [ t2_restrict.c(6,16) ]
   remark #15415: vectorization support: non-unit strided load was generated for the variable
<data[j*3+1]>, stride is 3    [ t2_restrict.c(7,16) ]
   remark #15415: vectorization support: non-unit strided load was generated for the variable
<data[j*3+2]>, stride is 3    [ t2_restrict.c(8,16) ]
   remark #15305: vectorization support: vector length 16
   remark #15309: vectorization support: normalized vectorization overhead 0.143
   remark #15300: LOOP WAS VECTORIZED
   remark #15442: entire loop may be executed in remainder
   remark #15451: unmasked unaligned unit stride stores: 1
   remark #15452: unmasked strided loads: 3
   remark #15475: --- begin vector cost summary ---
   remark #15476: scalar cost: 18
   remark #15477: vector cost: 5.680
   remark #15478: estimated potential speedup: 2.800
   remark #15488: --- end vector cost summary ---
   remark #25018: Total number of lines prefetched=4
   remark #25035: Number of pointer data prefetches=2, dist=8
   remark #25467: Number of extra pointer data prefetches for strided references=2
LOOP END
```

Prefetch Pragma Example3

```
scellrb9% cat star_pf8.c
   double * restrict a, * restrict b, * restrict c;
   int main() {
     a = (double *) _mm_malloc(sizeof(double)*(64000000+27),64);
     b = (double *) _mm_malloc(sizeof(double)*(64000000+27),64);
     c = (double *) _mm_malloc(sizeof(double)*(64000000+27),64);
     int j;   double scalar=3.0;
   #pragma omp parallel for
   #pragma noprefetch
     for (j=0; j<64000000; j++) {
       a[j] = 1.0;
       b[j] = 2.0;
       c[j] = 3.0;
     }

   #pragma omp parallel for
   #pragma prefetch *:1:64 // Directives  get obeyed even when prefetch level=2
   #pragma prefetch *:0:8
   #pragma vector nontemporal
     for (j=0; j<64000000; j++)
       a[j] = b[j] + scalar * c[j]; // 2 loads, 4 prefetches, no prefetch for nt store

     printf("\n %f \n", a[1]);
   }
```

Prefetch Pragma Example3 contd

icc -O3 -xmic-avx512 -**qopt-prefetch=2** -qopenmp -c -S -restrict -std=c99 star_pf8.c -unroll0

```
   remark #25018: Total number of lines prefetched=4
   remark #25035: Number of pointer data prefetches=4, dist=8
   remark #25149: Using directive-based hint=1, distance=64 for pointer data reference
[ star_pf8.c(22,12) ]
   remark #25141: Using second-level distance 8 for prefetching pointer data reference
[ star_pf8.c(22,12) ]
   remark #25149: Using directive-based hint=1, distance=64 for pointer data reference
[ star_pf8.c(22,24) ]
   remark #25141: Using second-level distance 8 for prefetching pointer data reference
[ star_pf8.c(22,24) ]

..B1.36:                        # Preds ..B1.36 ..B1.35     # Execution count [5.56e+00]
        prefetcht1 4096(%rsi,%r14,8)                        #22.12 c1
        vmovups   (%r15,%r14,8), %zmm1                      #22.24 c1
        vfmadd213pd (%rsi,%r14,8), %zmm0, %zmm1             #22.24 c7 stall 2
        prefetcht0 512(%rsi,%r14,8)                         #22.12 c7
        vmovntpd  %zmm1, (%rdx,%r14,8)                      #22.5 c13 stall 2
        prefetcht1 4096(%r15,%r14,8)                        #22.24 c13
        prefetcht0 512(%r15,%r14,8)                         #22.24 c15
        addq      $8, %r14                                  #17.1 c15
        cmpq      %rax, %r14                                #17.1 c17
        jb        ..B1.36       # Prob 82%                  #17.1 c19
```

Prefetch Pragma Example3 contd2

icc -O3 -xmic-avx512 **-qopt-prefetch=2** **-qopt-streaming-stores:never** -qopenmp -c -S -restrict -std=c99 star_pf8.c -unroll0

```
   remark #25018: Total number of lines prefetched=6
   remark #25035: Number of pointer data prefetches=6, dist=8
   remark #25149: Using directive-based hint=1, distance=64 for pointer data reference
[ star_pf8.c(22,12) ]
   remark #25141: Using second-level distance 8 for prefetching pointer data reference   [ star_pf8.c(22,12) ]
   remark #25149: Using directive-based hint=1, distance=64 for pointer data reference
[ star_pf8.c(22,24) ]
   remark #25141: Using second-level distance 8 for prefetching pointer data reference   [ star_pf8.c(22,24) ]
   remark #25149: Using directive-based hint=1, distance=64 for pointer data reference    [ star_pf8.c(22,5) ]
   remark #25141: Using second-level distance 8 for prefetching pointer data reference   [ star_pf8.c(22,5) ]
```

```
..B1.37:                        # Preds ..B1.37 ..B1.36         # Execution count [5.56e+00]
        prefetcht1 4096(%r15,%r14,8)                            #22.12 c1
        vmovups   (%rsi,%r14,8), %zmm1                          #22.24 c1
        vfmadd213pd (%r15,%r14,8), %zmm0, %zmm1                 #22.24 c7 stall 2
        prefetcht0 512(%r15,%r14,8)                             #22.12 c7
        vmovupd   %zmm1, (%rdx,%r14,8)                          #22.5 c13 stall 2
        prefetcht1 4096(%rsi,%r14,8)                            #22.24 c13
        prefetcht0 512(%rsi,%r14,8)                             #22.24 c15
        prefetcht1 4096(%rdx,%r14,8)                            #22.5 c17
        prefetcht0 512(%rdx,%r14,8)                             #22.5 c19
        addq      $8, %r14                                      #17.1 c19
        cmpq      %rax, %r14                                    #17.1 c21
        jb        ..B1.37       # Prob 82%                      #17.1 c23
```

C++ Example Using Lambda Function

```
typedef double* __restrict__  __attribute__((align_value (64))) Real_ptr;
typedef int Indx_type;

template <typename LOOP_BODY>
inline  __attribute__((always_inline))
void forall(Indx_type begin, Indx_type end, LOOP_BODY loop_body)
{
#pragma simd
#pragma vector aligned
#pragma prefetch *:1:25
#pragma prefetch *:0:2
  for ( Indx_type ii = begin ; ii < end ; ++ii ) {  loop_body( ii );  }
}
void foo8(Indx_type len, Real_ptr out1, Real_ptr out2, Real_ptr out3,
                Real_ptr in1, Real_ptr in2)
{
  forall(0, len, [&] (Indx_type i) {
    out1[i] = in1[i] * in2[i] ;
    out2[i] = in1[i] + in2[i] ;
    out3[i] = in1[i] - in2[i] ;
  } ) ;
}
```

C++ Ex. Using Lambda - Contd

```
$ icpc -c –qopt-prefetch=3 -qopt-report=3 -qopt-report-phase=loop,vec
      star_pf7.cpp -std=c++0x –xmic-avx512 –O3 -unroll0

LOOP BEGIN at star_pf7.cpp(12,4) inlined into star_pf7.cpp(17,4)
  remark #15301: SIMD LOOP WAS VECTORIZED
…
  remark #25018: Total number of lines prefetched in=10
  remark #25021: Number of initial-value prefetches=6
  remark #25035: Number of pointer data prefetches=10, dist=8
  remark #25149: Using directive-based hint=1, distance=25 for pointer
      data reference    [ star_pf7.cpp(18,21) ]
  remark #25141: Using second-level distance 2 for prefetching pointer
      data reference   [ star_pf7.cpp(18,21) ]
…
```

- Prefetch pragma using the * syntax to control all arrays inside the loop
- Command-line uses –unroll0 option for illustrative purposes only
  - In general, all unrolled cache-lines are prefetched irrespective of the unroll factor chosen by the compiler for the vectorized loop
- 5 arrays, 2 prefetches per array, 10 cache-lines prefetched inside the loop
- First-level prefetch distance =25 vectorized loop-iterations ahead

Prefetch Directives in Fortran

```
          sum = 0.d0
          do j=1,lastrow-firstrow+1
            i = rowstr(j)
            iresidue = mod( rowstr(j+1)-i, 8 )
            sum = 0.d0
CDEC$ NOPREFETCH a,p,colidx
            do k=i,i+iresidue-1
              sum = sum +  a(k)*p(colidx(k))
            enddo
CDEC$ NOPREFETCH p
CDEC$ PREFETCH a:1:16
CDEC$ PREFETCH colidx:0:8
            do k=i+iresidue, rowstr(j+1)-8, 8
              sum = sum + a(k  )*p(colidx(k  ))
     &                  + a(k+1)*p(colidx(k+1)) + a(k+2)*p(colidx(k+2))
     &                  + a(k+3)*p(colidx(k+3)) + a(k+4)*p(colidx(k+4))
     &                  + a(k+5)*p(colidx(k+5)) + a(k+6)*p(colidx(k+6))
     &                  + a(k+7)*p(colidx(k+7))
            enddo
           q(j) = sum
          enddo
```

- CDEC$ prefetch var:hint:distance
- hint value can be 0-3, distance in terms of iterations (possibly vectorized)

C Prefetch Intrinsics

```
#include <stdio.h>
#include <immintrin.h>
#define N 1000
int main(int argc, char **argv)
{
  int i, j, htab[N][2*N];
  for (i=0; i<N; i++) {
#pragma noprefetch  // Turn off compiler prefetches for this loop
    for (j=0; j<2*N; j++) {
      _mm_prefetch((const char *)&htab[i][j+20], _MM_HINT_T1); // vprefetch1
      _mm_prefetch((const char *)&htab[i][j+2],   _MM_HINT_T0); // vprefetch0
      htab[i][j] = -1;
    }
  }
  printf("htab element is %d\n", htab[3][40]);  return 0;
}
/* constants to use with _mm_prefetch  (extracted from *mmintrin.h) */
#define _MM_HINT_T0 1
#define _MM_HINT_T1 2
#define _MM_HINT_T2 3
#define _MM_HINT_NTA    0
#define _MM_HINT_ENTA   4
#define _MM_HINT_ET0    5
#define _MM_HINT_ET1    6
#define _MM_HINT_ET2    7
```

Fortran Prefetch Intrinsics

```
subroutine spread_lf (a, b)
PARAMETER (n = 1028)
real*8 a(n,n), b(n,n), c(n)
do j = 1,n
   do i = 1,n
      a(i, j) = b(i-1, j) + b(i+1, j)
      call mm_prefetch (a(i+2, j), 0)
      call mm_prefetch (a(i+20, j), 1)
      call mm_prefetch (b(i+21, j), 1)
   enddo
enddo
print *, a(2, 567)
stop
end
```

- ifort -O2 –xmic-avx512 -c foo.f
- Compiler auto-prefetching not turned on (no –qopt-prefetch option) here

Prefetch Performance Tuning on KNL

Hardware prefetcher does a reasonable job in many cases - but compiler prefetch tuning may help in extracting the maximal performance

Careful software prefetching of select data structures in loops that are bound by latency (as indicated by Vtune) are good candidates to try prefetching pragmas or options

Try these options for your application:

- –qopt-prefetch=2 (Especially Fortran codes with big loop-nests)
- **-qopt-prefetch=3**
- **-qopt-prefetch=5 (if app has lots of gathers/scatters)**
- Add "–mP2OPT_hlo_pref_hint=1 –mP2OPT_hlo_pref_int_hint=1" with each of the above
- Can also use prefetch pragmas to do this on a per-loop basis
- For applications with lots of loops with short trip-counts:
  - Try –mP2OPT_hpo_pref_initial_vals=100 <or_any_large_value>

Prefetch Perf Tuning on KNL (2)

Even when an app overall doesn't show gains from prefetching, individual loops may be gaining

- Use Vtune to identify such cases
- Try tuning with prefetch distance options or pragmas
- Note that retuning may be required for each MPI/OMP configuration

Observed ~6% improvement in geomean (of 15 benchmarks) on SPEC ACCEL OMP suite with per-benchmark prefetch tuning (only using options)

- Estimated results on KNL 68 core, 2 thread (136 threads) configuration

Legal Disclaimer & Optimization Notice

The estimated results reported above may need to be revised as additional testing is conducted. The results depend on the specific platform configurations and workloads utilized in the testing, and may not be applicable to any particular user's components, computer system or workloads. The results are not necessarily representative of other benchmarks and other benchmark results may show greater or lesser impact from mitigations.

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. For more complete information visit www.intel.com/benchmarks.

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.

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

BACKUP

Prefetching Basics

Compiler prefetching is turned on by default for the Intel® Xeon Phi™ coprocessor

- At option levels –O2 and above
- Prefetches issued for all regular memory accesses inside loops
- Prefetching for memory accesses expressed using load/store intrinsics
- Maximal loop prefetching
- For a detailed discussion, see paper at:
  - http://software.intel.com/sites/default/files/article/326703/mtaap2013-prefetch-streaming-stores.pdf

Use the compiler reporting options to see detailed diagnostics of prefetching per loop

- –qopt-report3

Use option –no-opt-prefetch to turn off cmplr prefetching

Loop-Prefetches

Prefetches issued targeting memory access in a future iteration of the loop

Targeting regular array accesses

Pointer accesses similar to array accesses where the address can be predicted in advance

Supports address calculations that involve:

- Affine functions of surrounding loop indices
- More complicated access-patterns that require additional instructions inside the loop

Prefetch Instructions Generated

Compiler issues two prefetches for each memory-reference inside a loop: one VPREFETCH1 and one VPREFETCH0 (with a shorter distance)

- Exclusive variant (such as VPREFETCHE1) issued for stores
- Compiler heuristics determine prefetch distance to be used for each memory-reference
  - Distance is the number of iterations ahead that a prefetch is issued
  - Prefetching is done after vectorization-phase, so distance is in terms of vectorized iterations if loop is vectorized
- Prefetch distance can be controlled via options and pragmas
  - Use the option to control prefetch distance for all loops in compilation scope
  - Use the loop-level pragma to control prefetch distance per memory reference

Loop-Prefetching Heuristics

Compiler issues prefetches for memory accesses specified using load/store intrinsics

- These are treated similar to regular loads/stores

Prefetches issued for memory-references at any loop-level, distances calculated taking inner-loops into account

Compiler generates initial-value prefetches (using vprefetch0) for first few cache-lines before entering the inner loop

- Useful especially for short-trip-count inner loops
  - Compiler default is to issue a maximum of 6 such prefetch instructions before each loop
  - Use the internal option –mP2OPT_hlo_pref_initial_vals=<n> to increase this limit (say, with n=100)

For data-accesses in inner-loops with a surrounding loop-nest, compiler decides whether to prefetch for a future iteration of the inner-loop or the outer-loop. Heuristics use parameters such as:

- Trip-count estimates of inner and outer loops
- Symbolic contiguity-analysis of data-accesses inside inner loop

Prefetching Using Intrinsics

Prefetch intrinsics supported by the compiler for fine-tuning

- Turn off compiler prefetching (via option or pragma) to minimize overlap with compiler-issued prefetches in such cases

Interactions with the Hardware Prefetcher

Intel® Xeon Phi™ coprocessor has a hardware L2 prefetcher that is enabled by default

If software prefetches are doing a good job, then hardware prefetching does not kick in

- In several workloads (such as stream), maximal software prefetching gives the best performance

Any references not prefetched by compiler may get prefetched by hardware

Prefetch Distance Computation

Distance reported in terms of (potentially vectorized) loop iterations

Compiler starts off assuming an L2 miss

- Distance computed based on memory latency

Distance refined based on compiler estimate of trip counts

- Constant trip counts
- Trip count directives
- Estimate of max trip count based on array dimensions
- Dynamic profiles
- Triangular loops handled recursively

Identifies contiguous access across outer loop iterations based on symbolic analysis

If prefetch distance too high, recalculate distance based on L2 latency, if distance still too high, prefetching turned off

Distance calculation takes TLB pressure into account

- If the prefetch distance value chosen will cause undue pressure on TLB, distance is throttled to prevent TLB thrashing
- Prefetch distance calculated at runtime to account for TLB pressure when data-access stride is unknown at compile-time

Directive Support for Loop Prefetches

Directive to turn off prefetching for a particular loop

- #pragma noprefetch
- CDEC$ noprefetch
- Specify before a loop, affects only that loop, does not affect inner loops

Directive to turn off prefetching for a particular routine

- #pragma noprefetch
- CDEC$ noprefetch
- Specify at the top of the routine as the first executable statement

Directive Support - Contd

Prefetch pragma support for C loops

- Apply uniform distance for all arrays in a loop:
  - #pragma prefetch *:hint:distance
- Fine-grained control for each array:
  - #pragma prefetch var:hint:distance
  - #pragma noprefetch var
- You can combine the two forms for the same loop

```
#pragma prefetch *:1:5
#pragma noprefetch A // prefetch only for B and C arrays
 for(int i=0; i<n; i++)  { C[i] = A[B[i]]; }
```

Prefetch directive support for Fortran loops

- Apply uniform distance for all arrays in a loop:
  - CDEC$ prefetch *:hint:distance
- Fine-grained control for each array:
  - CDEC$ prefetch var:hint:distance
  - CDEC$ noprefetch var

Directive Support – Contd2

When the user inserts (any) prefetch pragma for a variable in a loop, the compiler will explicitly issue only the prefetch specified in the pragma for that variable inside the loop

If the user wants only L2->L1 prefetches, use:

- #pragma prefetch src_arr:0:1
- Only the vprefetch0 will be issued and no vprefetch1 for this variable.

If the user wants both vprefetch1 and vprefetch2, then use:

- #pragma prefetch src_arr:1:8
- #pragma prefetch src_arr:0:1

Prefetch Distance Tuning Option

-opt-prefetch-distance=n1[,n2]

- n1 specifies the distance for first-level prefetches into L2
- n2 specifies prefetch distance for second-level prefetches from L2 to L1 (use n2 <= n1)
- -opt-prefetch-distance=64,32
- -opt-prefetch-distance=24
  - Use first-level distance=24, second-level distance to be determined by compiler
- -opt-prefetch-distance=0,4
  - Turns off all first-level prefetches, second-level uses distance=4 (Use this if you want to rely on hardware prefetching to L2, and compiler prefetching from L2 to L1)
- -opt-prefetch-distance=16,0
  - First-level distance=16, no second-level prefetches issued
- If option not specified, all distances determined by compiler

Prefetch Performance Tuning

If algorithm is well blocked to fit in L2 cache, prefetching is less critical

For data access patterns where L2-cache misses are common , prefetching is critical

- Default compiler heuristics typically use a first-level prefetch distance of <=8 vectorized iterations
- For bandwidth-bound benchmarks (such as stream), using a larger first-level prefetch (vprefetch1) distance sometimes shows performance improvements
- If you see a performance drop when you turn off compiler-prefetching, the app is a likely candidate that will benefit from fine-tuning of compiler prefetches with options/pragmas

Prefetch Performance Tuning - Contd

Use different first-level (vprefetch1) and second-level prefetch (vprefetch0) distances to fine-tune your application performance

- -opt-prefetch-distance=n1[,n2]
- Useful values to try for n1: 0,4,8,16,32,64
- Useful values to try for n2: 0,1,2,4,8
- Can also use prefetch pragmas to do this on a per-loop basis
- Try –mP2OPT_hpo_pref_initial_vals=100 <large_value>

If your application hot-spots use indirect accesses (gather/scatter) or non-unit-strided accesses, then try enhanced compiler prefetching for such references (described more in later slides)

- Use appropriate pragma for each such loop OR
- Add option –mP2OPT_hlo_pref_indirect_refs=T
- Add option –mP2OPT_hlo_pref_multiple_pfes_strided_refs=T

C Prefetch Directives

Src-code snippet:

```
for (i=i0; i!=i1; i+=is) {
  float sum = b[i];
  int ip = srow[i];
  int c = col[ip];
#pragma NOPREFETCH col
#pragma PREFETCH value:1:12
#pragma NOPREFETCH x
  for(; ip<srow[i+1];
c=col[++ip])
    sum -= value[ip] * x[c];
  y[i] = sum;
}
```

Pseudo-code for compiler-generated code:

```
for (i=i0; i!=i1; i+=is) {
  float sum = b[i];  int ip = srow[i];
  int c = col[ip];

  /*pref for refs in outer loop with dist d2/d1*/
  /* No prefetch directive for outer loop, use
     compiler heuristics for prefetching */
  vprefetch1(&b[i+is*d2]);
  vprefetch0(&b[i+is*d1]);
  vprefetch1(&srow[i+is*d2]);
  vprefetch0(&srow[i+is*d1]);
  vprefetch1(&y[i+is*d2]);
  vprefetch0(&y[i+is*d1]);

  for(…) {
    /* vprefetch1 for value with a distance of 12, no
prefetching for others. If loop is vectorized, prefetch
12 vector-iters ahead*/
    vprefetch1(&value[ip+12*VLEN]);
  }
  y[i] = sum;
}
```

- #pragma prefetch var:hint:distance
- hint value can be in the range 0-3, distance in terms of iterations
- hint-0 means vprefetch0, hint-1 means vprefetch1, …

C Prefetch Directives - Contd

```
void foo(int *htab_p, int m1, int N)
{
  int i, j;

  for (i=0; i<N; i++) {

#pragma prefetch htab_p:1:16
#pragma prefetch htab_p:0:6

// Issue vprefetch1 for htab_p with a distance of 16 vectorized iterations ahead
// Issue vprefetch0 for htab_p with a distance of 6 vectorized iterations ahead
// If pragmas are not present, compiler chooses both distance values

     for (j=0; j<2*N; j++) {
            htab_p[i*m1 + j] = -1;
     }
  }
}
```

C Prefetch Directives – Example Using Intrinsics

```
#pragma prefetch a:1:64 // Use distance of 64 vectorized iterations for a - vprefetch1
#pragma prefetch a:0:8   // Use distance of 8 vectorized iterations for a - vprefetch0
#pragma noprefetch b     // No prefetches for b
 for (i = 0; i < nn; i+=16) {
    _val =  _mm512_load_ps ((void*)(&a[i]));

    _yy  = _mm512_add_ps (_val, _val);

    _mm512_extstore_ps ((void*)(&b[i]), _yy, _MM_DOWNCONV_PS_NONE, _MM_HINT_NONE);
 }
```

C Prefetch Directives - Contd

```
#pragma omp parallel for
// Use distance of 64 vectorized iterations for b,c arrays - vprefetch1
#pragma prefetch *:1:64
// Use distance of 8 vectorized iterations for b,c arrays – vprefetch0
#pragma prefetch *:0:8
// array a marked as streaming-store, no prefetches issued for a
#pragma vector aligned nontemporal
  for (j=0; j<N1; j++)
    a[j] = b[j]+scalar*c[j];
```

$ icc -qopt-report-phase=loop -qopt-report3 pf_test.c -mmic -restrict –openmp

```
LOOP BEGIN at pf_test.c(12,4)
 remark #25018: Total number of lines prefetched=4
   …
 remark #25149: Using directive-based hint=1, distance=64 for pointer data reference [ pf_test.c(13,15) ]
 remark #25141: Using second-level distance 8 for prefetching pointer data reference [ pf_test.c(13,15) ]
 remark #25149: Using directive-based hint=1, distance=64 for pointer data reference [ pf_test.c(13,20) ]
 remark #25141: Using second-level distance 8 for prefetching pointer data reference [ pf_test.c(13,20) ]
```

Prefetch Directives in Fortran (2)

```
      subroutine spread(a1, b, n)
      integer n
      real*8 a1(:), b(:)

C Issue vprefetch0 for a1 with a distance of 4 vectorized iterations ahead
C Issue vprefetch1 for b with a distance of 40 vectorized iterations ahead
C Issue vprefetch0 for b with a distance of 8 vectorized iterations ahead

!dir$ prefetch a1:0:4
!dir$ prefetch b:1:40
!dir$ prefetch b:0:8
      do i = 1,N
        a1(i) = b(i-1) + b(i+1)
      enddo

      return
      end
```

Loop Prefetch Example

```
void work(int i,  __m512 *b, __m512 *c);
void f1(__m512 *a, __m512 *b, __m512 *c)
{
 for (i = 0; i < 1024; i++) {
   work(i, b, c);
   a[i] = _mm512_mul_ps(b[i], _mm512_loadd(&c[i],  _MM_FULLUPC_NONE,
            _MM_BROADCAST32_NONE, _MM_HINT_NONE)); // No pref hint
 }
}
```

```
$ icc -O2 –qopt-report3 –qopt-report-phase=loop intrin5_ex.c
…
remark #25018: Total number of lines prefetched=6
remark #25019: Number of spatial prefetches=6, dist=24
…
```

- Loop has normal loads of a[i] and b[i]
- Intrinsic load c[i] treated just like b[i] and a[i]
- Prefetching reported as part of –opt-report output
  - 3 arrays, 2 prefetches per array, 6 cache-lines prefetched
  - First-level prefetch distance =24 loop-iterations ahead

Loop Prefetch Example2

```
for(int y = y0; y < y1; ++y) {
    float div, *restrict A_cur = &A[t & 1][z * Nxy + y * Nx];
    float *restrict A_next = &A[(t + 1) & 1][z * Nxy + y * Nx];
    float *restrict vvv = &vsq[z * Nxy + y * Nx];
    for(int x = x0; x < x1; ++x) { // Typical trip-count is 192, 12 after vectorization
       div = c0 * A_cur[x]  + c1 * ((A_cur[x + 1] + A_cur[x - 1])
                   + (A_cur[x + _Nx] + A_cur[x - _Nx])
                   + (A_cur[x + Nxy] + A_cur[x - Nxy]))
                   + c2 * ((A_cur[x + 2] + A_cur[x - 2]) + ...
       A_next[x] = 2 * A_cur[x] - A_next[x] + vvv[x] * div;
    }
}
```

$ icc -O2 -qopt-report3 -qopt-report-phase=loop,vec p3_orig.cpp

…

remark #15301: LOOP WAS VECTORIZED.

remark #25018: Total number of lines prefetched=38

remark #25035: Number of pointer data prefetches=38, dist=8

…

- Prefetch coverage is low (dist =8) since typical trip-count is only 12
- Use –opt-prefetch-distance=2,1 (Or add pragmas)
- Or use loop-count directive before inner-loop: #pragma loop count (192)

Prefetches for Indirect Accesses

Compiler also supports prefetching for indirect memory accesses

- Not turned on by default
- Requires user to add pragmas OR use internal options

Indirect memory access:

- Most common form: A[B[i]]

Index array: B

- B[i] and B[i+1] can index to distant elements in the A[] array

No spatial locality over A

- A[B[i+1]] need not be in the same or next cache line of A[B[i]]
- Accessing A[B[i]]
  - Cannot be expected to make accesses A[B[i+1]], A[B[i+2]], … hit in the cache

Prefetches for Indirect Refs - Contd

Prefetching for A[B[i]]

- Need to assume:
  - B[i], B[i+1], B[i+2],… index to A[] elements on separate cache lines.
- Must issue a separate prefetch for each element
  - We need to prefetch A[B[i+1]], A[B[i+2]], …

Indirect Prefetch: Non-Vectorized Loop

Case 1: Assume LOOP IS NOT VECTORIZED

At iteration i (assume no unroll)

- A[B[i]] is used
- A[B[i+distance]] is prefetched

Example: Let distance=2

- A[B[i]] is used
- A[B[i+2]] is prefetched

Indirect Prefetch: Vectorized Loop

Case 2: Assume LOOP IS VECTORIZED

At iteration i (assume no unroll, 4B data type integer or float)

- A[B[i:16]] are accessed
- A[B[(i+distance*16):16]] are prefetched

Example: Let distance=2

- A[B[i]], A[B[i+1]],…A[B[i+15]] are accessed (16 elements)
  - For example, using a vgather instruction
- A[B[i+32]], A[B[i+33]],…A[B[i+47]] are prefetched (16 elements, potentially 16 different cache-lines)
  - Prefetches done using regular vprefetch instructions (16 of them)

Indirect Prefetch Pragma Example

```
void foo(int n, int* A, int *B, int *C)
{
#pragma vector aligned
#pragma prefetch A:1:3
#pragma simd
   for(int i=0; i<n; i++)  { C[i] = A[B[i]]; }
}
```

$ icc –c -mmic indirect_p.c –qopt-report3 –opt-report-phase=loop,vec

…

remark #15301: SIMD LOOP WAS VECTORIZED

remark #25018: Total number of lines prefetched=20

remark #25021: Number of initial-value prefetches=6

remark #25033: Number of indirect prefetches=16, dist=2

remark #25035: Number of pointer data prefetches=4, dist=8

remark #25141: Using second-level distance 4 for prefetching pointer data reference [indirect_p.c(7,40)]

remark #25150: Using directive-based hint=1, distance=3 for indirect memory reference [indirect_p.c(7,38)]

remark #25141: Using second-level distance 4 for prefetching pointer data reference [indirect.c(7,31)]

remark #25143: Inserting bound-check around lfetches for loop

…

Indirect Prefetch Option Example

```
void foo(int n, int* A, int *B, int *C)
{
#pragma vector aligned
#pragma simd
   for(int i=0; i<n; i++)  {
      C[i] = A[B[i]];
   }
}
```

$ icc –c -mmic **-mP2OPT_hlo_pref_indirect_refs=T** indirect.c –qopt-report3 –qopt-report-phase=loop,vec

…

remark #15301: SIMD LOOP WAS VECTORIZED
remark #25018: Total number of lines prefetched=20
remark #25018: Total number of lines prefetched=20
remark #25033: Number of indirect prefetches=16, dist=2
remark #25035: Number of pointer data prefetches=4, dist=8
remark #25141: Using second-level distance 4 for prefetching pointer data reference [indirect.c(6,17)]
remark #25141: Using second-level distance 4 for prefetching pointer data reference [indirect.c(6,8)]
remark #25143: Inserting bound-check around lfetches for loop

…

Indirect Prefetch Example: After Vectorization

```
void foo(int n, int* A,
int *B, int *C)
{
#pragma vector aligned
#pragma simd
  for(int i=0; i<n; i++)
    C[i] = A[B[i]];
}
```

Pseudo Code after vectorization (no prefetches):

```
+ DO i1 = 1, t99, 16(SI32)    <DO_LOOP> <VEC>
|    t101 = [al64]t3[i1 - 1];
|    t102 = &t2[0];
|    (M512) t103 = _mm512_setzero{ic=VX512_SETZERO}(  );
|    (M512) t103 = _mm512_mask_gatherd{ic=VX512_MASK_I32GATHERD}
                                  ( t103, 65535(I16), t101, t102, 0(SI32), 4(SI32), 0(SI32) );
|    [al64]t1[i1 - 1] = t103;
+ END DO
```

Load B[i:16] into t101

Use gather to load A[t101] into t103

Store t103 into C[i:16]

Example: After Adding Prefetches (With Bounds Check)

```
void foo(int n, int* A,
int *B, int *C)
{
#pragma vector aligned
#pragma simd
  for(int i=0; i<n; i++)
    C[i] = A[B[i]];
}
```

```
prefetch0 ( &t3[0]); prefetch0 ( &t3[16]); prefetch0 ( &t3[32]); prefetch0 ( &t3[48])
prefetche0 ( &t1[0]); prefetche0 ( &t1[16])

+ DO i1 = 1, t99, 16(SI32)    <DO_LOOP> <VEC>
|    t101 = [al64]t3[i1 - 1];
|    t102 = &t2[0];
|    (M512) t103 = _mm512_setzero{ic=VX512_SETZERO}(  );
|    (M512) t103 = _mm512_mask_gatherd{ic=VX512_MASK_I32GATHERD}(t103, 65535(I16), t101, t102, 0(SI32), 4(SI32), 0(SI32) );
|    [al64]t1[i1 - 1] = t103;
|    if ( i1 + 32(SI32) <= t99 )
|    {
|       prefetch1 ( &t2[ t3[i1-1+32] ] )
|       prefetch1 ( &t2[ t3[i1-1+33] ] )
|       ...
|       prefetch1 ( &t2[ t3[i1-1+46] ] )
|       prefetch1 ( &t2[ t3[i1-1+47] ] )
|       prefetch1 ( &[al64]t3[i1-1 + 128]); prefetch0 ( &[al64]t3[i1-1 + 64])
|       prefetche1 ( &[al64]t1[i1-1 + 128]); prefetche0 ( &[al64]t1[i1-1 + 64])
|    }
+ END DO
```

Initial value prefetching: for C[i] and B[i]

Condition to prevent loads of B beyond array bounds: *Don't prefetch in last 2 iterations.*

Indirect prefetching for A[B[i]] at distance=2: A[B[i+32]], … A[B[i+47]]

Spatial prefetching: B[i] and C[i] at distance=8,4

Example: After Adding Prefetches (Without Bounds Check)

```
void foo(int n, int* A,
int *B, int *C)
{
#pragma vector aligned
#pragma simd
  for(int i=0; i<n; i++)
    C[i] = A[B[i]];
}
```

```
prefetch0 ( &t3[0]); prefetch0 ( &t3[16]); prefetch0 ( &t3[32]); prefetch0 ( &t3[48])
prefetche0 ( &t1[0]); prefetche0 ( &t1[16])

+ DO i1 = 1, t99, 16(SI32)    <DO_LOOP> <VEC>
|    t101 = [al64]t3[i1 - 1];
|    t102 = &t2[0];
|    (M512) t103 = _mm512_setzero{ic=VX512_SETZERO}(  );
|    (M512) t103 = _mm512_mask_gatherd{ic=VX512_MASK_I32GATHERD}(t103, 65535(I16), t101, t102, 0(SI32), 4(SI32), 0(SI32) );
|    [al64]t1[i1 - 1] = t103;
|    prefetch1 ( &t2[ t3[i1-1+32] ] )
|    prefetch1 ( &t2[ t3[i1-1+33] ] )
|    ...
|    prefetch1 ( &t2[ t3[i1-1+46] ] )
|    prefetch1 ( &t2[ t3[i1-1+47] ] )
|    prefetch1 ( &[al64]t3[i1-1 + 128]); prefetch0 ( &[al64]t3[i1-1 + 64])
|    prefetche1 ( &[al64]t1[i1-1 + 128]); prefetche0 ( &[al64]t1[i1-1 + 64])
+ END DO
```

Initial value prefetching:
for B[i] and C[i]

No Bound check, obtained using:
-mP2OPT_hlo_pref_insert_bound_check_for_indirect_refs=F

Indirect prefetching for A[i] at distance=2:
A[B[i+32]] … A[B[i+47]]

Spatial prefetching:
C[i] and B[i] at distance=8,4

Indirect Prefetch: More Details

Internal prefetch is done by the compiler using regular vprefetch instructions for KNC

- Not to be confused with instructions for gather-hints and gather-prefetches
- See article here for details on compiler generation of gather hints (such as VGATHERPF0HINTDPD)
  - http://software.intel.com/en-us/articles/selective-use-of-gatherhintscatterhint-instructions
- Compiler does not automatically generate gather prefetch instructions (such as VGATHERPF0DPS)

Compiler-generated indirect prefetching also enabled for Xeon

- Requires advanced options such as "-O3 –opt-prefetch –xCORE-AVX2"

Fortran Indirect Prefetch Example

```
 Subroutine spmxv( nrows, nelmts, indx, rowp, matvals, invec, outvec )
      Integer :: nrows, nelmts, ncols, rowp(nrows), indx(nelmts), i, j, ii
      Real*8 :: matvals(nelmts), invec(*), outvec(nrows), temp1
!$omp parallel do
      Do i = 1, nrows – 1
!dec$ vector aligned
        Do j = rowp(i), rowp(i+1) - 1
          outvec(i) = outvec(i) + matvals(indx(j))*invec(indx(j))
        End Do
      End Do
 End Subroutine spmxv
```

```
scel2%: ifort -O2 -qopt-report-phase=loop,vec sparse_mv.f -mmic -
      mP2OPT_hlo_pref_indirect_refs=T -openmp -unroll0 -c -opt-report3
...
LOOP BEGIN at sparse_mv.f(7,15)
   remark #15301: LOOP WAS VECTORIZED
   remark #25018: Total number of lines prefetched=18
   remark #25019: Number of spatial prefetches=2, dist=8
   remark #25021: Number of initial-value prefetches=2
   remark #25033: Number of indirect prefetches=16, dist=2
   remark #25143: Inserting bound-check around lfetches for loop
...
```

Indirect Prefetching via Intrinsics

Tips for advanced users who prefer to do the prefetching for indirect accesses via intrinsics in source

- In some such loops, user may be able to insert prefetches for all cache-lines likely to be accessed (via indirect references) inside the loop before entering the loop
- In other cases, user may want to do the indirect-access prefetches inside the loop (for a future access)
  - May require careful consideration to make sure that the address-calculations (that will involve some load-operations) don't result in out-of-bound accesses

Indirect Prefetch via intrinsics example

```
#pragma simd reduction(+:fxtmp,fytmp,fztmp) vectorlengthfor(double)
for (int jj = 0; jj < jnum; jj++) {
    int j,sbindex, jtype; double factor_lj;
    j = jlist[jj];   sbindex = sbmask(j); …
        _mm_prefetch((char *) &xx[jlist[jj+1+16]],  1);
        _mm_prefetch((char *) &xx[jlist[jj+2+16]],  1);
         …
        _mm_prefetch((char *) &xx[jlist[jj+8+16]],  1);
        _mm_prefetch((char *) &ff[jlist[jj+1+16]],  5);
        …
        _mm_prefetch((char *) &ff[jlist[jj+8+16]],  5);
    double delx = xtmp - xx[j].x;    double dely = ytmp - xx[j].y;
    double delz = ztmp - xx[j].z;    double rsq = delx*delx + dely*dely + delz*delz;
    if (rsq < global_cutsq)  {
         double r2inv = 1.0/rsq;         double r6inv = r2inv*r2inv*r2inv;
         double forcelj = r6inv * (global_lj1*r6inv - global_lj2);
         double fpair = factor_lj*forcelj*r2inv;
         fxtmp += delx*fpair;     fytmp += dely*fpair;       fztmp += delz*fpair;
         if (NEWTON_PAIR || j < nlocal) {
              ff[j].x -= delx*fpair;     ff[j].y -= dely*fpair;     ff[j].z -= delz*fpair;  }
     }
}
```

Extra Prefetches for Strided Accesses in Vectorized Loops

Compiler supports inserting extra prefetches for strided memory accesses in vectorized loops to cover multiple cache line accesses

- Not turned on by default
- Requires user to add pragmas OR use internal options

Strided memory access:

- Most common form: A[3*i]

Assuming A[] is an 4B integer type, the vector loop accesses 16 elements in one iteration and the cache line size is 64 bytes the above reference will access three cache lines so the compiler will insert three prefetches (with the same hint), one for each of the three cache lines

If the stride is unknown, the compiler inserts a prefetch for each element assuming that they access different cache lines

Prefetches for Strided Access Internal Option Example

```
void foo(int* data, int* res, int n, int xi, int yi, int zi)
{
   int j, xij, yij, zij;

   for (j = 0; j < n; ++j) {
      xij = xi - data[3 * j];
      yij = yi - data[3 * j + 1];
      zij = zi - data[3 * j + 2];

      res[j] = xij * xij + yij * yij + zij * zij;
   }
}
```

Prefetches for Strided Access Internal Option Example - Contd

$ icpc -c -mmic -opt-report-phase=loop -opt-report3 **-mP2OPT_hlo_pref_multiple_pfes_strided_refs=T** strided.c

```
LOOP BEGIN at strided.c(5,5)
   remark #25018: Total number of lines prefetched=8
   remark #25021: Number of initial-value prefetches=6
   remark #25035: Number of pointer data prefetches=4, dist=8
   remark #25465: Number of extra pointer data prefetches for strided
      references=4
   remark #25141: Using second-level distance 4 for prefetching pointer data
      reference   [ strided.c(6,20) ]
...
```

- Total number of lines prefetched (8) is the sum of regular pointer data prefetches (4) and extra pointer data prefetches for strides references (4)
- The regular prefetches consist of vprefetch1 and vprefetch0 for res[] and data[]
- Extra prefetches consist of vprefetch1 and vprefetch0 for the two extra cache lines accessed by data[]
- Number of cache lines accessed is calculated as (element_stride(12) * number_of_elements_accessed_per_iter(16))/ cache_line_size(64) = 3

Prefetch Pragma Example For Strided Access

```
void foo(int n, int* A, int *B)
{
#pragma prefetch A:1:10
#pragma prefetch A:0
for(int i=0; i<n; i++)  { B[i] = A[2*i]; }
}
```

```
$  icpc -c -mmic -opt-report-phase=loop -opt-report3 strided.c

LOOP BEGIN at strided.c(5,5)
   remark #25018: Total number of lines prefetched=6
   remark #25021: Number of initial-value prefetches=6
   remark #25035: Number of pointer data prefetches=4, dist=8
   remark #25465: Number of extra pointer data prefetches for strided
      references=2
   remark #25149: Using directive-based hint=1, distance=10 for pointer data
      reference    [strided.c(5,38) ]
   remark #25141: Using second-level distance 5 for prefetching pointer data
      reference   [ strided.c(5,38) ]
…
```

Fortran Example For Strided Access

```
subroutine foo(A, B, N, N1, X)
    integer N, N1, X
    integer A(N), B(N)
    !dec$ prefetch A:1
 do i = 1,N1
       B(i) = A(X*i)
    enddo
    return
 end
```

```
$  ifort -c -mmic -opt-report-phase=loop -opt-report3 strided.f

LOOP BEGIN at strided.f(7,7)
   remark #25018: Total number of lines prefetched=18
   remark #25019: Number of spatial prefetches=2, dist=8
   remark #25021: Number of initial-value prefetches=4
   remark #25023: Number of unconditional prefetches=1
   remark #25464: Number of extra unconditional prefetches for strided
      references=15
   remark #25148: Using directive-based hint=1, distance=8 for prefetching
      unconditional memory reference    [ strided1.f(7,10) ]
...
```

Prefetch For Strided Access Example: After Vectorization

```
void foo(int n, int* A,
int *B)
{
for(int i=0; i<n; i++)
    B[i] = A[3*i];
}
```

Pseudo Code after vectorization (no prefetches):

```
+ DO i1 = 1, t99, 16   <DO_LOOP> <VEC>
   |    t101 = *((&t1[3*i1 - 3](M512)));
   |    t2[i1 - 1](M512) = t101;
+ END DO
```

Vector load of A[i:45:3] into t101

Store t101 into B[i:15]

Prefetch For Strided Access Example: After Adding Prefetches

Pseudo Code after vectorization (no prefetches):

```
+ DO i1 = 1, t99, 16   <DO_LOOP> <VEC>
   |    t101 = *((&t1[3*i1 - 3](M512)));
   |    t2[i1 - 1](M512) = t101;
   |    prefetch1 ( &t1[3*i1 + 381]);
   |    prefetch0 ( &t1[3*i1 + 189]);
   |    prefetch1 ( &t1[3*i1 + 397]);
   |    prefetch0 ( &t1[3*i1 + 205]);
   |    prefetch1 ( &t1[3*i1 + 413]);
   |    prefetch0 ( &t1[3*i1 + 221]);
   |    prefetch1 ( &t2[i1 + 127]);
   |    prefetch0 ( &t2[i1 + 63]);
+ END DO
```

```
void foo(int n, int* A,
int *B)
{
for(int i=0; i<n; i++)
    B[i] = A[3*i];
}
```

Usual spatial prefetching for A[i] and B[i] at distance=8,4

Extra prefetches for A[3*i] covering the next 2 cache lines

General Tips and Comments

Use optimization reports to understand what the compiler is doing:

- -qopt-report-phase=loop,vec –qopt-report=3
- Check whether loop of interest is properly vectorized
  - "Loop Vectorized" message is the first step, look at generated asm (add –S option) to study if the loop is vectorized efficiently
  - You can get extra information using –qopt-report=5

Trip-count vs prefetch distance

- Correlate runtime loop trip-counts with prefetch distances (and vectorization) to understand their efficiency
- Turn off compiler prefetching if code already uses intrinsic prefetches
- Loop-prefetching works well when inner-loop trip-count is large compared to distance (coverage high)
  - If not true, try smaller distance (using option, loop count directive, etc.)

General Tips (contd)

Use loop count directives to give hints to compiler

- Affects prefetch distance calculation
- #pragma loop count (200) before a loop

Refer to Compiler Documentation and the following links for more performance tips

- http://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture