Compiler Prefetching for the Intel® Xeon Phi™ coprocessor

Rakesh Krishnaiyer
Intel Compiler Lab
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:

Use the compiler reporting options to see detailed diagnostics of prefetching per loop:
- –qopt-report3

Use option –no-opt-prefetch to turn off compiler 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 $\leq 8$ 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), then try compiler prefetching for indirect references (described more in later slides)

- Use appropriate pragma for each such loop OR
- Add option –mP2OPT_hlo_pref_indirect_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;
}
C Prefetch Directives - Contd

```c
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

```c
#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) ]
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-report3 -qopt-report-phase=loop,vec star_pf7.cpp - std=c++0x -mmic -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

```fortran
sum = 0.d0
do j=1, lastrow-firstrow+1
   i = rowstr(j)
iresult = mod( rowstr(j+1)-i, 8 )
   sum = 0.d0
   CDEC$ NOPREFETCH a, p, colidx
   do k=i, i+iresult
      sum = sum + a(k)*p(colidx(k))
   enddo
   CDEC$ NOPREFETCH p
   CDEC$ PREFETCH a:1:16
   CDEC$ PREFETCH colidx:0:8
   do k=i+iresult, 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)**
Prefetch Directives in Fortran (2)

```fortran
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
```

Copyright © 2012, Intel Corporation. All rights reserved.
*Other brands and names are the property of their respective owners.*
#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 -mmic -c foo.f -mP2OPT_hlo_prefetch=F
• Compiler auto-prefetching turned off by option here
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

```c
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]) + ...)
    }
}
```

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

remark #15301: LOOP WASVECTORIZED.

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

14.0 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 \( A[B[i+1]], A[B[i+2]], \ldots \) hit in the cache
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
  – 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)
```
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: 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**

- **-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
Indirect Prefetch Option Example

```c
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

Pseudo Code after vectorization (no prefetches):

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

+ **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**: Load B[i:16] into t101

**Use gather to load A[t101] into t103**: Use gather to load A[t101] into t103

**Store t103 into C[i:16]**
Example: After Adding Prefetches (With Bounds Check)

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

**Initial value prefetching:** For C[i] and B[i]

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

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

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

Initially:
- prefetch0 ( &t3[0] ); prefetch0 ( &t3[16] ); prefetch0 ( &t3[32] ); prefetch0 ( &t3[48] )
- prefetch0 ( &t1[0] ); prefetch0 ( &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] )
|     prefetch0 ( &[al64]t1[i1-1 + 128] ); prefetch0 ( &[al64]t1[i1-1 + 64] )
|   }
+ END DO
Example: After Adding Prefetches (Without Bounds Check)

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

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

No Bound check, obtained using:
-mP2OPT_hlo_pref_insert_bound_check_for_indirect_refs=F


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

- 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)
- 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
#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

15.0 Product Update 1 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

```c
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

```c
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

Pseudo Code after vectorization (no prefetches):

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

+ 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):

```c
+ 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
```

```c
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 –ansi-alias option for C++ programs
- Not ON by default
- Enables compiler to do type-based disambiguation (pointers and floats don’t overlap)
- Enables DOLOOP recognition that is very important for MIC prefetching
- Without the flag, the compiler may assume that the trip-count is changing inside the loop if the upper-bound is a object-field access

Use optimization reports to understand what the compiler is doing:
- -qopt-report-phase=loop,vec -qopt-report3
- 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-report5

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

Use compiler option `-no-opt-prefetch` to turn off all compiler prefetching

Refer to Compiler Documentation and the following links for more performance tips
  • [http://software.intel.com/mic-developer](http://software.intel.com/mic-developer)
Compiler Prefetching for Xeon

Compiler prefetching is less important for Xeon compared to MIC

• Out-of-order vs. MIC in-order core

Compiler prefetch not enabled on Xeon by default

• Requires external option -opt-prefetch=\(<n>\)
• Compiler takes advantage of processor-specific flags to tune (-O3 -xAVX, etc.)
• See Compiler User Guide for more details

Most of this presentation is specific to MIC
Summary

Prefetching is very important for performance on the Intel® Xeon Phi™ coprocessor

Tune your code using compiler prefetch options and pragmas

Report compiler improvement opportunities to us!