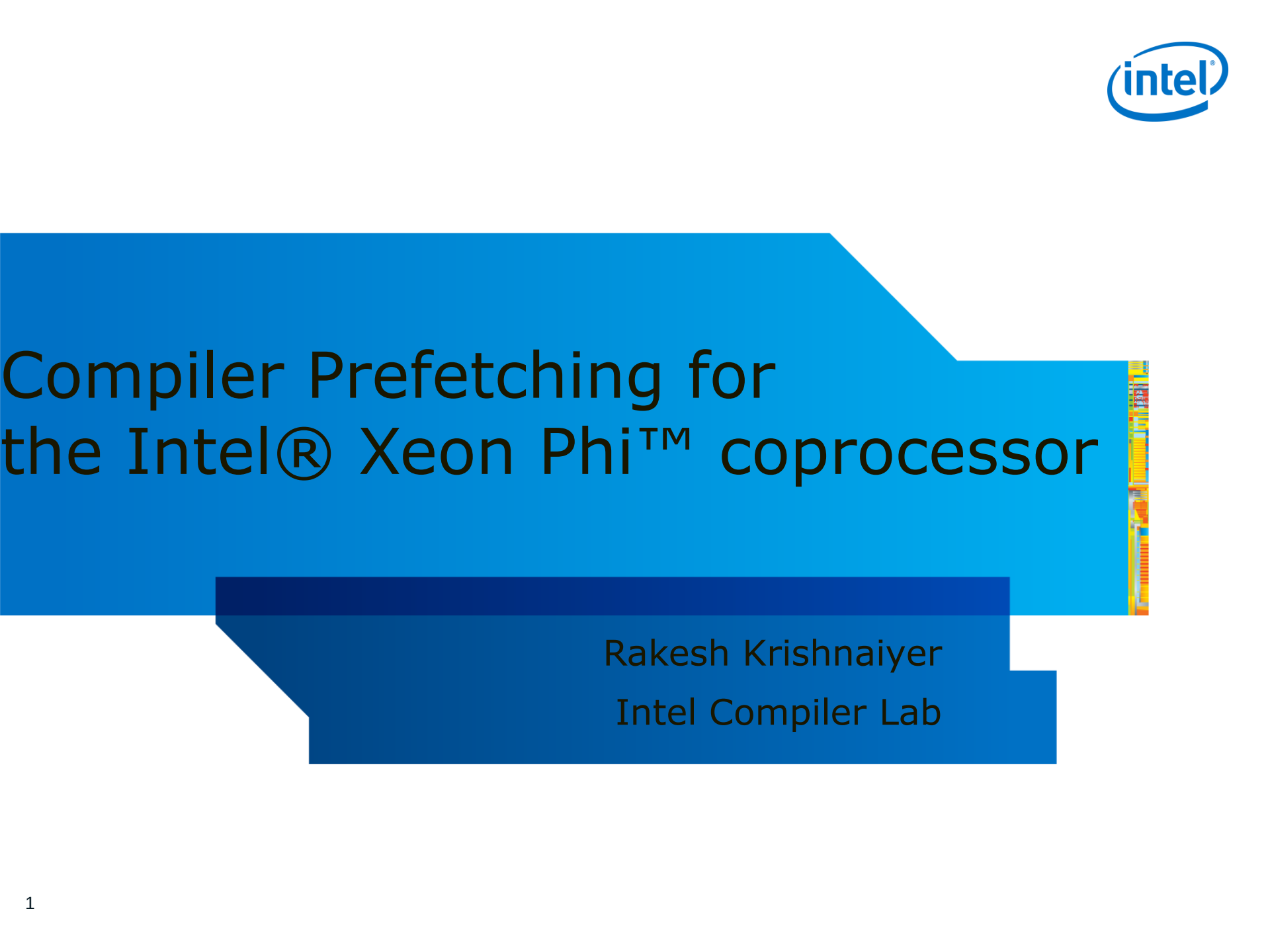


# Compiler Prefetching for the Intel® Xeon Phi™ coprocessor

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

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

- -opt-report-phase hlo –opt-report 3

Use compiler 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

# Handling Indirect Accesses

Compiler does not issue any prefetches for indirect memory accesses of the form a[b[i]]

- For loops where such indirect accesses cause a lot of cache-misses, user may want to do prefetching using compiler intrinsics
- 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

Compiler does issue gatherpf0hint/scatterpf0hint for gathers/scatters (such as VGATHERPF0HINTDPS)

- But it does not generate gather prefetch instructions (such as VGATHERPF0DPS)

# Indirect Prefetch 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;  }
     }
}
```

# 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

Prefetch pragma support for C loops

- #pragma prefetch var:hint:distance

Prefetch directive support for Fortran loops

- CDEC$ prefetch var:hint:distance

# 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

# 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);
 }
```

# 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)

# 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
```

# 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 -mmic -c foo.f -mP2OPT_hlo_prefetch=F
- Compiler prefetches 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
 }
}
 scel2%: icc -O2 –opt-report 3 –opt-report-phase hlo intrin5_ex.c
 High Level Optimizer Report for: f1
 Total #of lines prefetched in f1 for loop at line 6=6
   # of spatial prefetches in f1 for loop at line 6=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;
    }
}
scel2%: icc -O2 -mP2OPT_hlo_report -restrict -vec-report1 p3_orig.cpp
High Level Optimizer Report for: _Z10serial_veciiiiiiii
p3_orig.cpp(53): (col. 17) remark: LOOP WAS VECTORIZED.
```

**Total #of lines prefetched in _Z10serial_veciiiiiiii for loop at line 53=38**
**# of dynamic_mapped_array prefetches in _Z10serial_veciiiiiiii for loop at line 53=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)

# 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 LRB 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:

- -opt-report-phase hlo –opt-report 3 –vec-report 2
- 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 –vec-report6

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 mic-dev site for more performance tips

# 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
- 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!

# BACKUP