aiXcelerate 2016, Intel optimization report & compiler directives

November, 2016

Rene’ Puttin

NEC Deutschland GmbH

\Orchestrating a brighter world NEC

Intel optimization report

Orchestrating a brighter world NEC

Intel optimization report: introduction

- Intel compilers provide a detailed optimization report

- –qopt-report=<level>   Output detail level
  - 0=disable optimization report, 2=default, 5=maximum detail level

- Different ways to specify output destination:

-qopt-report-file=[stdout|stderr|<file>]	specify destination explicitly
-qopt-report-stdout	print output to stdout
-qopt-report-per-object (DEFAULT)	generate one .optrpt file per object

- Restrictions to specific file and optimization phases are possible:
  - ifort --help reports
  - ifort -qopt-report-help

Intel optimization report: example – source code (1/2)

```fortran
function pow3(x) result(r)

  implicit none

  real(kind=8) :: x, r

  r = x*x*x

end function pow3

```

Intel optimization report: example code (2/2)

```fortran
subroutine report_test(m,n,A,B,ind_inj,ind_noninj)

  implicit none
  integer(kind=8) :: ind_inj(m), ind_noninj(m)
  real(kind=8) :: A(m,n), B(m,n)
  integer :: m, n
  integer :: i, j
  real(kind=8) :: pow3

  do j = 1, n
    do i = 1, n
      B(i,j) = B(i,j) + pow3(A(i,j))
    end do
!dir$ ivdep
    do i = 1, n
      B(ind_inj(i),j) = B(ind_inj(i),j) + pow3(A(i,j))
    end do
    do i = 1, n
      B(ind_noninj(i),j) = B(ind_noninj(i),j) + pow3(A(i,j))
    end do
  end do

end subroutine report_test
```

Intel optimization report: example opt-report (1/4)

```
Begin optimization report for: REPORT_TEST

    Report from: Interprocedural optimizations [ipo]

INLINE REPORT: (REPORT_TEST) [2] test.f90(11,12)
  -> INLINE: (22,25) POW3
  -> INLINE: (26,43) POW3
  -> INLINE: (29,49) POW3

    Report from: Loop nest, Vector & Auto-parallelization optimizations
[loop, vec, par]

LOOP BEGIN at test.f90(20,3)
<Distributed chunk1>
   remark #25426: Loop Distributed (3 way)
   remark #15542: loop was not vectorized: inner loop was already
vectorized

   LOOP BEGIN at test.f90(21,5)
   <Peeled loop for vectorization>
      remark #25456: Number of Array Refs Scalar Replaced In Loop: 2
   LOOP END
```

Intel optimization report: example opt-report (2/4)

```
  LOOP BEGIN at test.f90(21,5)
 remark #15300: LOOP WAS VECTORIZED
    remark #15442: entire loop may be executed in remainder
    remark #15450: unmasked unaligned unit stride loads: 2
    remark #15451: unmasked unaligned unit stride stores: 1
    remark #15475: --- begin vector loop cost summary ---
    remark #15476: scalar loop cost: 12
    remark #15477: vector loop cost: 3.000
    remark #15478: estimated potential speedup: 3.510
    remark #15488: --- end vector loop cost summary ---
    remark #25456: Number of Array Refs Scalar Replaced In Loop: 8
  LOOP END

  LOOP BEGIN at test.f90(21,5)
  <Remainder loop for vectorization>
    remark #15301: REMAINDER LOOP WAS VECTORIZED
    remark #25456: Number of Array Refs Scalar Replaced In Loop: 2
  LOOP END

  LOOP BEGIN at test.f90(21,5)
  <Remainder loop for vectorization>
  LOOP END
LOOP END
```

Intel optimization report: example opt-report (3/4)

```
LOOP BEGIN at test.f90(20,3)
<Distributed chunk2>
  remark #15542: loop was not vectorized: inner loop was already
vectorized
  LOOP BEGIN at test.f90(25,5)
  <Peeled loop for vectorization>
    remark #25456: Number of Array Refs Scalar Replaced In Loop: 3
  LOOP END
  LOOP BEGIN at test.f90(25,5)
    remark #15300: LOOP WAS VECTORIZED
    remark #15442: entire loop may be executed in remainder
    remark #15448: unmasked aligned unit stride loads: 1
    remark #15450: unmasked unaligned unit stride loads: 2
    remark #15458: masked indexed (or gather) loads: 1
    remark #15459: masked indexed (or scatter) stores: 1
    remark #15475: --- begin vector loop cost summary ---
    remark #15476: scalar loop cost: 15
    remark #15477: vector loop cost: 11.500
    remark #15478: estimated potential speedup: 1.290
    remark #15488: --- end vector loop cost summary ---
    remark #25456: Number of Array Refs Scalar Replaced In Loop: 16
  LOOP END
  …
```

Intel compiler directives

\Orchestrating a brighter world NEC

Intel compiler directives: overview

General syntax

- Fortran: **!dir$ <directive>**
- C / C++: **#pragma <directive>**

Most important directives

● **ivdep**	hint to compiler that loop does not include dependencies
● **[no]vector**	hint to compiler to (not) vectorize a loop
● **simd**	forces the compiler to vectorize a loop (if possible)
● **[no]block_loop**	tells compiler to (not) cache-block loop
● **unroll**	tells compiler to unroll loop
● **unroll_and_jam**	tells the compiler to unroll outer loops and jam them
● **[no]fusion**	tells the compiler to (not) fuse loops
● **distribute point**	tells the compiler to divide loop
● **[no/force]inline**	tells / forces the compiler to (not) inline a subroutine

Intel compiler directives: ivdep

ivdep

```
  do j = 1, n
!dir$ ivdep
    do i = 1, m
      B(indi(i),indj(j)) = B(indi(i),indj(j)) + exp(A(i,j))
    end do
  end do
```

User knowledge:
ind is injective in example =>
set ivdep

```
Runtime (original):          5.5 sec
Runtime with directives:     2.2 sec
```

```
 LOOP BEGIN at sub.f90(16,5)
      remark #15344: loop was not vectorized: vector dependence
prevents vectorization. First dependence is shown below. Use
level 5 report for details
      remark #15346: vector dependence: assumed FLOW dependence
between b line 17 and b line 17
```

```
 LOOP BEGIN at sub.f90(16,5)
      remark #15300: LOOP WAS VECTORIZED
…
      remark #15478: estimated potential speedup: 3.620
```

if ind is not injective
ivdep will lead to
wrong results!!!

Intel compiler directives: novector

novector

```
!dir$ novector
  do j = 1, n
!dir$ novector
    do i = 1, m
      if (A(i,j) >= 0.0d0) then
        B(i,j) = B(i,j) + A(i,j)**2.4 + A(i,j)**3.7
      else
        B(i,j) = B(i,j) + 1.0d0
      end if
    end do
  end do
```

User knowledge: most A(i,j) are negative => set novector

Set novector also for outer loop, otherwise compiler might vectorize outer loop

```
Runtime:                      5.4 sec
Runtime with directives:      4.4 sec
```

```
 LOOP BEGIN at sub.f90(17,5)
      remark #15300: LOOP WAS VECTORIZED
      …
      remark #15478: estimated potential speedup: 1.470
```

```
 LOOP BEGIN at sub.f90(17,5)
      remark #15319: loop was not vectorized: novector
directive used
```

Intel compiler directives: block_loop

block_loop

```
!dir$ block_loop
  do k = 3, o
    do j = 1, n
      do i = 1, m-1
        B(i,j) = B(i,j) + A(i,j,k) + A(i,j-1,k) + A(i,j,k-1)
      end do
    end do
  end do
```

User knowledge:

User knowledge: inner loop is long, therefore j-1, k-1 elements cannot be accessed cache-friendly
=> Set block_loop

```
Runtime:                     17.3 sec
Runtime with directives:     10.0 sec
```

```
LOOP BEGIN at sub.f90(14,3)
   LOOP BEGIN at sub.f90(14,3)
      LOOP BEGIN at sub.f90(14,3)
          LOOP BEGIN at sub.f90(14,3)
            remark #25442: blocked by 4   (pre-vector)
            LOOP BEGIN at sub.f90(15,5)
               remark #25442: blocked by 10   (pre-vector)
               LOOP BEGIN at sub.f90(16,7)
                  remark #25442: blocked by 128   (pre-vector)
```

Intel compiler directives: distribute point

distribute point

```
 do j = 1, n
    do i = 1, m
      call sub2(A(i,j))
!dir$ distribute point
      B(i,j) = B(i,j) + exp(A(i,j))
    end do
  end do
```

User knowledge:
No dependencies between sub2 call and remaining loops
=> set distribute point

```
Runtime:                    19.8 sec
Runtime with directives:    14.4 sec
```

```
 LOOP BEGIN at sub.f90(15,5)
      remark #15382: vectorization support: call to function
sub2_ cannot be vectorized   [ sub.f90(16,12) ]
      remark #15344: loop was not vectorized: vector dependence
prevents vectorization
```

```
 LOOP BEGIN at sub.f90(15,5)
   <Distributed chunk2>
…
 remark #15301: PARTIAL LOOP WAS VECTORIZED
…
 remark #15478: estimated potential speedup: 5.460
```

Intel compiler directives: vector nontemporal

vector nontemporal

```
 do j = 1, n
!dir$ vector nontemporal
    do i = 1, m
      B(i,j) = A(i,j)
    end do
  end do
```

User knowledge:
Arrays do not fit into cache, or arrays are not needed in near future => set vector nontemporal

```
Runtime:                    20.0 sec
Runtime with directives:    13.7 sec
```

vector nontemporal

- indicates compiler to use streaming-stores (skip cache)
- same behaviour as –qopt-streaming-stores=always
- speedup for STREAM benchmark and loops working on huge datasets

vector temporal

- same behaviour as –qopt-streaming-stores=never
- indicates compiler to use non-streaming-stores (write data into cache)
- speedup for small amounts of data, that are used again soon

In most cases compiler does a good job on decisions