Induction Variable Detection Tests Design

Change history:

|  |  |  |
| --- | --- | --- |
| Oct. 28, 2014 | Initial design for Induction Variable Detection tests | Binbin Li |
| Nov. 12, 2014 | Refine the test cases according Xin’s suggestions | Binbin Li |
|  |  |  |

Review history:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

**Optimization Description**

An induction variable is a variable that gets increased/decreased by a constant on every iteration of a loop. It can also be the linear function of another IV. A basic IV is an inter-iteration dependent variable and a dependent IV is a not inter-iteration dependent on itself since it depends on a basic IV.

Details can be found in below link: <https://sp2010.amr.ith.intel.com/sites/SSG-703/VM/_layouts/WordViewer.aspx?id=/sites/SSG-703/VM/Shared%20Documents/Documents/QA%20Documents/Induction%20Variable%20Detection.docx&Source=https%3A%2F%2Fsp2010%2Eamr%2Eith%2Eintel%2Ecom%2Fsites%2FSSG%2D703%2FVM%2FSitePages%2FHome%2Easpx%3FRootFolder%3D%252Fsites%252FSSG%252D703%252FVM%252FShared%2520Documents%252FDocuments%252FQA%2520Documents%26FolderCTID%3D0x012000F7964ECBC709E04989CC4A4545409D1D%26View%3D%257bDB18E63B%2D7111%2D40AF%2D833C%2DFBE6FDE2A5D4%257d&DefaultItemOpen=1>

**Test Goals**

The primary goal is to test for functionality in corner cases. A stretched goal is to detect scenarios where optimization is not applied and can be a possible candidate to generalization.

**Approach**

Suggested by optimization owner, it is first important to understand the scenarios where the optimization is applied and this can be done via reading the document. For the second goal, the limitations of current implementation must be understood; that is done through a combination of the document and using verbosity on optimization pass.

**Limitation of Optimization**

Limitation of Induction Variables Detection

* It cannot detect cases that are commutative.
* It cannot always detect cases when an instruction has no side effect. For example, if there is a i = i + 0; i = i + 1; then “i” is not detected as an IV.

**Test Case Design**

First, the test suite contains a batch of test cases to test the functionality correctness of current optimization implementation. The code in tests can meet all the limitation and will be optimized.

1. Simple BIV

Test code contains a simple and non-nested loop with no branches

The loop only includes one BIV.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

double i = 0;

while ( i < 10 ) {

i = i + 2.234;

}

This test can also be extended to Float add, Int add (Lit 16), Int Substract, Long add, Long Substract.

1. BIV with different kind of increment.

Test code contains a simple and non-nested loop with no branches

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

float i = 0;

float j = 1.332f;

float k = 3.883f;

float m = 1.43f;

float n = 5;

while ( i < 10f ) {

i = i + 2;

j = j + m;

k += n + 23.3552f;

}

This test can also be extended to Double add, Int add, Int Substract, Long add, Long Substract.

1. BIV with changed increment variable.

Test code contains a simple and non-nested loop with no branches

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

float i = 0;

float j = 1.332f;

float k = 3.883f;

float m = 1.43f;

float m1 = m;

while ( i < 10f ) {

float m2 = m + 1;

i = i + m;

j = j + m1;

k = k + m2;

}

This test can also be extended to Double add, Int add, Int Substract, Long add, Long Substract.

1. MultipleBIV with no type convert.

Test code contains a simple and non-nested loop with no branches

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

int i1 = 0;

int i2 = 83442;

int m = 391;

long j1 = -142l;

long j2 = 783l;

float f = 23.4632f;

double d = 0;

while (i1 < 5) {

i1 = i1 + 1;

i2 = i2 - m;

j1 = j1 + 13l;

j2 = j2 - 57l;

f = f + 349f;

d = d + 97;

}

1. MultipleBIV with type convert.

Test code contains a simple non-nested loop with no branches

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop within data type convert

For example,

int i = 0;

long l1 = 332l;

float f1 = 0.234f;

float f2 = 83.982f;

double d1 = -242;

double d2 = 475;

while (i < 5) {

i = i + 1;

l1 = l1 + 16;

f1 = f1 + 749;

f2 = f2 + 39l;

d1 = d1 + 84l;

d2 = d2 + 97.323f;

}

000218: 1208 |0000: const/4 v8, #int 0 // #0

00021a: 1606 4c01 |0001: const-wide/16 v6, #int 332 // #14c

00021e: 1405 b29d 6f3e |0003: const v5, #float 0.234000 // #3e6f9db2

000224: 1404 c9f6 a742 |0006: const v4, #float 83.982002 // #42a7f6c9

00022a: 1802 0000 0000 0040 6ec0 |0009: const-wide v2, #double -242.000000 // #c06e400000000000

000234: 1800 0000 0000 00b0 7d40 |000e: const-wide v0, #double 475.000000 // #407db00000000000

00023e: 6009 0000 |0013: sget v9, LOptimizationTests/InductionVariable/MultiBIV\_02/Main;.iterations:I // field@0000

000242: 3598 1800 |0015: if-ge v8, v9, 002d // +0018

000246: d808 085c |0017: add-int/lit8 v8, v8, #int 92 // #5c

00024a: 1609 1000 |0019: const-wide/16 v9, #int 16 // #10

00024e: bb96 |001b: add-long/2addr v6, v9

000250: 1409 0040 3b44 |001c: const v9, #float 749.000000 // #443b4000

000256: c695 |001f: add-float/2addr v5, v9

000258: 1509 1c42 |0020: const/high16 v9, #int 1109131264 // #421c

00025c: c694 |0022: add-float/2addr v4, v9

00025e: 1909 5540 |0023: const-wide/high16 v9, #long 4635611391447793664 // #4055

000262: cb92 |0025: add-double/2addr v2, v9

000264: 1809 0000 0000 ac54 5840 |0026: const-wide v9, #double 97.322998 // #405854ac00000000

00026e: cb90 |002b: add-double/2addr v0, v9

000270: 28e7 |002c: goto 0013 // -0019

1. CommutateOperands

Test code contains a simple and non-nested loop with no branches

The loop only includes multiple BIVs.

The loop have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

int i0 = 0;

int i1 = 274;

int i2 = 762;

int i3 = 53;

int i4 = 87;

int i5 = 123;

int m = 15;

while ( i0 < 5 ) {

i0 = 1 + i0;

i1 = m + i1;

i2 = m + 3 + i2;

i3 = 74 + m + i3;

i4 = i4 + 74 + m;

i5 = 74 + i5 + m;

}

Test result: i0, i1, i2, i3 are BIV; i4, i5 are not BIV.

1. BranchesInLoop

Test code contains control flow branches and only have BIV on one path

Test code contains a non-nested loop.

The loop only includes one BIV.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

int i = 0;

int j = 2;

while (i < 5) {

if (j < 5) {

i = i + 1;

} else if ( i < 10 ) {

i = 5\*j - 2;

} else if ( i < 20 ) {

i = j\*(i + 3);

} else if ( i < 30 ) {

i = i \* j - 5;

} else {

i = j \* (i + 2);

}

}

1. BIV with supported cast

Test code contains a simple and non-nested loop with no branches

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop contain supported cast.

For example,

int i = 0;

int j = 3;

int m = 45;

int n = 5;

while ( i < 10 ) {

i = i + (short)j;

m = m - (byte)n;

}

000218: 1201 |0000: const/4 v1, #int 0 // #0

00021a: 1232 |0001: const/4 v2, #int 3 // #3

00021c: 1300 2d00 |0002: const/16 v0, #int 45 // #2d

000220: 1253 |0004: const/4 v3, #int 5 // #5

000222: 6004 0000 |0005: sget v4,

000226: 3541 0700 |0007: if-ge v1, v4, 000e // +0007

00022a: 8f24 |0009: int-to-short v4, v2

00022c: b041 |000a: add-int/2addr v1, v4

00022e: 8d34 |000b: int-to-byte v4, v3

000230: b140 |000c: sub-int/2addr v0, v4

000232: 28f8 |000d: goto 0005 // -0008

1. BIV within nested loops

Test code contains nested loop with no branches.

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

int m = 0;

int n = 0;

for ( int i = 0 ; i < 10; i++) {

m = m + 1;

for ( int j = 1; j < 10; j+=2) {

n = n + 1;

}

}

1. BIV within multiple loops

Test code contains multiple non-nested loop with no branches.

The loop only includes multiple BIVs.

The loop doesn’t have commutation of operands.

The loop doesn’t contain dependent induction variable

The loop only includes instructions: Additions, Subtractions, Constant affections

The loop doesn’t contain data type convert

For example,

int m = 0;

for ( int i = 0; i < 10; i++) {

m = m + 2;

}

for ( int j = 0; j < 10; j++) {

m = m + 1;

}