Compiler-assisted Performance Analysis

Adam Nemet
Apple
anemet@apple.com
Some Optimizations?

Compiler Optimization

X, Y

User

Hotspot

Bottleneck

Compiler

Hotspot

Legality

Cost Model

Some Optimizations?
Some Optimizations?

Compiler Optimization $X, Y$

User

Hotspot

Bottleneck

Hotspot

Compiler

Legality

Cost Model

Some Optimizations?
Some Optimizations?

Compiler Optimization X, Y

Disassemble

Hotspot

User

Bottleneck

Compiler

Hotspot

Legality

Cost Model

Some Optimizations?
Some Optimizations?

Compiler Optimization

User

Bottleneck

Hotspot

-comp-only

Compiler

Legality

Cost Model

Some Optimizations?
Some Optimizations?

Compiler Optimization $X, Y$

Optimization Diagnostics

User

Bottleneck

Compiler

Cost Model

Legality

Hotspot

Hotspot

Optimization Diagnostics in LLVM

- Supported in LLVM
- Only a small number of passes emit them
- -Rpass options to enable them in the compiler output

```c
foo.c:8:5: remark: accumulate inlined into compute_sum[-Rpass=inline]
    accumulate(arr[i], sum);
^{
```
Optimization Diagnostics in LLVM

- Supported in LLVM
- Only a small number of passes emit them
- `-Rpass` options to enable them in the compiler output
- For large programs, the output of `-Rpass` is noisy and unstructured
Messages appear in no particular order

Remarks for hot and cold code are intermixed

How can we make this information accessible and actionable?

Messages from successful and failed optimizations are dumped together
Wish List

• **All in one place**: Optimizations Dashboard

• **At a glance**: See high-level interaction between optimizations for targeted low-level debugging

• **Filtering**: Noise-level should be minimized by focusing on the hot code

• **Integration**: Display hot code and the optimizations side-by-side
Approach

• Extend existing optimization remark infrastructure
• Add the new optimizations
• Add ability to output remarks to a data file
• Visualize data in HTML
• Targeting compiler developers initially
Example

```c
void accumulate(int x, int *a) {
    *a += x;
}

int compute_sum(int arr[], int n) {
    int sum = 0;
    for (int i = 0; i < n; ++i)
        accumulate(arr[i], &sum);
    return sum;
}
```
Work Flow

$ clang -O3 -fsave-optimization-record -c foo.c
$ utils/opt-viewer/opt-viewer.py foo.opt.yaml html
$ open html/foo.c.html
### Successful Optimizations

<table>
<thead>
<tr>
<th>Line Optimization</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>void accumulate(int x, int *a) {</code></td>
</tr>
<tr>
<td>2</td>
<td><code>*a += x;</code></td>
</tr>
<tr>
<td>3</td>
<td><code>}</code></td>
</tr>
<tr>
<td>4</td>
<td><code>int compute_sum(int arr[], int n) {</code></td>
</tr>
<tr>
<td>5</td>
<td><code>int sum = 0;</code></td>
</tr>
<tr>
<td>6</td>
<td><code>for (int i = 0; i &lt; n; ++i)</code></td>
</tr>
<tr>
<td>7</td>
<td><strong>loop-vectorize</strong> vectorized loop (vectorization width: 4, interleaved count: 2)</td>
</tr>
<tr>
<td>8</td>
<td><code>accumulate(arr[i], &amp;sum);</code></td>
</tr>
<tr>
<td>9</td>
<td><code>accumulate</code> can be inlined into <code>compute_sum</code> with cost=-5 (threshold=487)</td>
</tr>
<tr>
<td>10</td>
<td><code>}</code></td>
</tr>
</tbody>
</table>

Remarks appear inline under the referenced line.

Further details about the optimization.

Name of the pass: Green for successful optimization.
Successful Optimizations

<table>
<thead>
<tr>
<th>Line Optimization</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>void accumulate(int x, int *a) {</td>
</tr>
<tr>
<td>2</td>
<td>*a += x;</td>
</tr>
<tr>
<td>3</td>
<td>}</td>
</tr>
<tr>
<td>4</td>
<td>int compute_sum(int arr[], int n) {</td>
</tr>
<tr>
<td>5</td>
<td>int sum = 0;</td>
</tr>
<tr>
<td>6</td>
<td>for (int i = 0; i &lt; n; ++i)</td>
</tr>
<tr>
<td>7</td>
<td>accumulate(arr[i], &amp;sum);</td>
</tr>
<tr>
<td>8</td>
<td>accumulate inlined into compute_sum</td>
</tr>
<tr>
<td>9</td>
<td>return sum;</td>
</tr>
</tbody>
</table>

vectorized loop (vectorization width: 4, interleaved count: 2)

accumulate can be inlined into compute_sum with cost=-5 (threshold=487)

Column aligned with the expression

HTML link to facilitate further analysis
Successful Optimizations

Remarks in white are Analysis remarks

vectorized loop (vectorization width: 4, interleaved count: 2)
accumulate(arr[i], &sum);

inline accumulate can be inlined into compute_sum with cost=-5 (threshold=487)

Optimizations can expose interesting analyses
Missed Optimizations

```
void accumulate(int x, int *a);

int compute_sum(int arr[], int n) {
    int sum = 0;
    for (int i = 0; i < n; ++i)
        accumulate(arr[i], &sum);
    return sum;
}
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Optimization</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td><code>accumulate(int x, int *a);</code></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td><code>compute_sum(int arr[], int n) {</code></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td><code>    int sum = 0;</code></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td><code>    for (int i = 0; i &lt; n; ++i)</code></td>
</tr>
<tr>
<td>5</td>
<td>loop-vectorize</td>
<td><code>accumulate(arr[i], &amp;sum);</code></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td><code>loop not vectorized</code></td>
</tr>
<tr>
<td>7</td>
<td>inline</td>
<td><code>accumulate will not be inlined into compute_sum because its definition is unavailable</code></td>
</tr>
<tr>
<td>8</td>
<td>loop-vectorize</td>
<td><code>loop not vectorized: call instruction cannot be vectorized</code></td>
</tr>
</tbody>
</table>

Red means failed optimization.
LLVM Changes

Pass pipeline

IR → Inliner → LoopVectorizer → IR

```
ORE.emit(OptimizationRemarkAnalysis("inline", "CanBeInlined", Call)
  << NV("Callee", Callee) << " can be inlined into " << NV("Caller", Caller)
  << " with cost=" << NV("Cost", IC.getCost())
  << " threshold=" << NV("Threshold", Threshold));
```

```
-llvm-pass-analysis=inline
```

```
foo.c:8:5: remark: accumulate can be inlined into compute_sum with cost=-5 (threshold=487) [-Rpass-analysis=inline]
  accumulate(arr[i], sum);
```
ORE.emit(OptimizationRemarkAnalysis("inline", "CanBeInlined", Call)
  << NV("Callee", Callee) << " can be inlined into " << NV("Caller", Caller)
  << " with cost=" << NV("Cost", IC.getCost())
  << " threshold=" << NV("Threshold", Threshold));

Pass pipeline

IR → Inliner → LoopVectorizer → IR

- fsave-optimization-record
  enables source line debug info
  (-gline-tables-only)

YAML

LLVM Changes
--- !Analysis
Pass: inline
Name: CanBeInlined
DebugLoc: { File: s.cc, Line: 8, Column: 5 }
Function: compute_sum
Args:
  - Callee: accumulate
    DebugLoc: { File: s.cc, Line: 1, Column: 0 }
  - String: ' can be inlined into '
  - Caller: compute_sum
    DebugLoc: { File: s.cc, Line: 5, Column: 0 }
  - String: ' with cost='
  - Cost: '-5'
  - String: ' (threshold='
  - Threshold: '487'
  - String: ')'
...
opt-viewer

YAML

utils/opt-viewer/opt-viewer.py

index.html

foo.o.html
## Index

<table>
<thead>
<tr>
<th>Source Location</th>
<th>Function</th>
<th>Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:485:7</td>
<td>Func2</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7</td>
<td>Func2</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:474:7</td>
<td>Func2</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:221:13</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:245:27</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:246:23</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:251:2</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:261:14</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:271:3</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:292:3</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:5</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:288:6</td>
<td>Proc0</td>
<td>inline</td>
</tr>
<tr>
<td>/org/test-suite/SingleSource/Benchmarks/Dhrystone/dry.c:287:19</td>
<td>Proc0</td>
<td>inline</td>
</tr>
</tbody>
</table>
Noisy: Most of this code not hot

Sort by hotness
Use PGO for Hotness

--- !Analysis
Pass: inline
Name: CanBeInlined
DebugLoc: { File: s.cc, Line: 8, Column: 5 }
Function: compute_sum
Hotness: 3
Args:
- Callee: accumulate
  DebugLoc: { File: s.cc, Line: 1, Column: 0 }
- String: ' can be inlined into '
- Caller: compute_sum
  DebugLoc: { File: s.cc, Line: 5, Column: 0 }
- String: ' with cost='
- Cost: '-5'
- String: '(threshold='
- Threshold: '487'
- String: ')
...
Hotness

Relative to maximum hotness, NOT total time %
Optimizations Recorded

- Function Inliner
- Loop Vectorizer
- Loop Unroller
- Loop Data Prefetch
- LICM
- GVN
- Loop Idiom
- Loop Deletion
- SLP Vectorizer

... more to follow
Test Drive on LLVM test suite
Improve & Evaluate

1. Does the information presented in this high-level view contain sufficient detail to reconstruct what happened?

2. Can we discover the interactions between optimizations?

3. With the improved visibility, can we quickly find real performance opportunities?
DhryStone
(SingleSource/Benchmark)

Interaction of Optimizations
DhryStone

430 Proc8(Array1Par, Array2Par, IntPar1l, IntPar12)
431    Array1Dim    Array1Par;
432    Array2Dim    Array2Par;
433    OneToFifty   IntPar1l;
434    OneToFifty   IntPar12;
435 {
436     REG OneToFifty IntLoc;
437     REG OneToFifty IntIndex;
438
439     IntLoc = IntPar1l + 5;
440     Array1Par[IntLoc] = IntPar12;
441     Array1Par[IntLoc+1] = Array1Par[IntLoc];
442     Array1Par[IntLoc+30] = IntLoc;
443     for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
444     {
445      loop-delete
446      loop-vectorize
447      vectorized loop (vectorization width: 4, interleaved count: 2)
448      Array2Par[IntLoc][IntIndex] = IntLoc;
449      ++Array2Par[IntLoc][IntLoc-1];
450      load of type i32 not eliminated in favor of store because it is clobbered by store
451      load of type i32 not eliminated in favor of store because it is clobbered by call
452      Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
453      load of type i32 not eliminated in favor of store because it is clobbered by store
454      load of type i32 eliminated
455      IntGlob = 5;
456     }
457     }
DhryStone

Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
    Array1Dim = Array1Par;
    Array2Dim = Array2Par;
    OneToFifty = IntParI1;
    OneToFifty = IntParI2;
    {
        REG OneToFifty = IntLoc;
        REG OneToFifty = IntIndex;
        IntLoc = IntParI1 + 5;
        Array1Par[IntLoc] = IntParI2;
        Array1Par[IntLoc+1] = Array1Par[IntLoc];
        Array1Par[IntLoc+30] = IntLoc;
        for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
            vectorized loop (vectorization width: 4, interleaved count: 2)
            Array2Par[IntLoc][IntIndex] = IntLoc;
            ++Array2Par[IntLoc][IntLoc-1];
            Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
            load of type i32 not eliminated in favor of store because it is clobbered by store
        IntGlob = 5;
    }
DhryStone

```c
Proc8(Array1Par, Array2Par, IntPar1, IntPar2)
Array1Dim      Array1Par;
Array2Dim      Array2Par;
OneToFifty     IntPar1;
OneToFifty     IntPar2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;

    IntLoc = IntPar1 + 5;
    Array1Par[IntLoc] = IntPar2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        Array2Par[IntLoc][IntIndex] = IntLoc;
        ++Array2Par[IntLoc][IntLoc-1];

    Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
    load of type i32 not eliminated in favor of store because it is clobbered by store
    IntGlob = 5;
}
```
DhryStone

```c
Proc8(Array1Par, Array2Par, IntPar1l, IntPar12)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntPar1l;
OneToFifty IntPar12;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;
    IntLoc = IntPar1l + 5;
    Array1Par[IntLoc] = IntPar12;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        loop deleted
    Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    load of type i32 not eliminated in favor of store because it is clobbered by store
    load of type i32 not eliminated in favor of store because it is clobbered by call
    Arrav2Par[IntLoc+201fIntLoc1] = ArravlPar[IntLoc1];
    load of type i32 eliminated
    IntGlob = 5;
}
```
DhryStone

Proc0(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim = Array1Par;
Array2Dim = Array2Par;
OneTo Fifty = IntParI1;
OneTo Fifty = IntParI2;
{
    REG OneTo Fifty = IntLoc;
    REG OneTo Fifty = IntIndex;

    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc + 1] = Array1Par[IntLoc];
    Array1Par[IntLoc + 30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc + 1); ++IntIndex)
        loop deleted

Array2Par[IntLoc][IntIndex] = IntLoc;

++Array2Par[IntLoc][IntLoc-1];
load of type i32 not eliminated in favor of store because it is clobbered by store
load of type i32 not eliminated in favor of store because it is clobbered by call

Arrav2Par[IntLoc+20][IntLoc] = Arrav1Par[IntLoc];
load of type i32 eliminated

IntGlob = 5;
DhryStone

```c
Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntParI1;
OneToFifty IntParI2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;
    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        loop deleted
    Array2Par[IntLoc][IntIndex] = IntLoc;
}
```
DhryStone

<table>
<thead>
<tr>
<th>Line</th>
<th>Percentage</th>
<th>Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>278</td>
<td>34%</td>
<td><code>while (IntLoc1 &lt; IntLoc2)</code> completely unrolled loop with 2 iterations</td>
<td>Proc0</td>
</tr>
<tr>
<td>279</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>281</td>
<td>18%</td>
<td>IntLoc3 = 5 * IntLoc1 - IntLoc2; Proc7(IntLoc1, IntLoc2, &amp;IntLoc3);</td>
<td>Proc0</td>
</tr>
<tr>
<td>282</td>
<td>18%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>283</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>284</td>
<td></td>
<td>Proc8 can be inlined into Proc0 with cost=125 (threshold=225)</td>
<td>Proc0</td>
</tr>
<tr>
<td>285</td>
<td>18%</td>
<td>Proc8 inlined into Proc0</td>
<td>Proc0</td>
</tr>
<tr>
<td>286</td>
<td>18%</td>
<td>Proc1 can be inlined into Proc0 with cost=15 (threshold=337)</td>
<td>Proc0</td>
</tr>
<tr>
<td>287</td>
<td></td>
<td>Proc1 inlined into Proc0</td>
<td>Proc0</td>
</tr>
<tr>
<td>288</td>
<td>34%</td>
<td>for (CharIndex = 'A'; CharIndex &lt;= Char2Glob; ++CharIndex) load hoisted</td>
<td>Proc0</td>
</tr>
<tr>
<td>289</td>
<td>17%</td>
<td>load of type i8 not eliminated in favor of store because it is clobbered by store</td>
<td>Proc0</td>
</tr>
<tr>
<td>290</td>
<td>11%</td>
<td>load of type i8 eliminated</td>
<td>Proc0</td>
</tr>
</tbody>
</table>
DhryStone

```c
Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim Array1Par;
Array2Dim Array2Par;
OneToFifty IntParI1;
OneToFifty IntParI2;
{
    REG OneToFifty IntLoc;
    REG OneToFifty IntIndex;
    IntLoc = IntParI1 + 5;
    Array1Par[IntLoc] = IntParI2;
    Array1Par[IntLoc+1] = Array1Par[IntLoc];
    Array1Par[IntLoc+30] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        loop deleted
        Array2Par[IntLoc][IntIndex] = IntLoc;
    ++Array2Par[IntLoc][IntLoc-1];
    load of type i32 not eliminated in favor of store because it is clobbered by store
    load of type i32 not eliminated in favor of store because it is clobbered by call
    Array2Par[IntLoc+201f][IntLoc1] = Array1Par[IntLoc];
    load of type i32 eliminated
    IntGlob = 5;
}
```
DhryStone: Summary

- Without low-level debugging, quickly reconstructed what happened
- Even though it involved interaction between multiple optimizations
  - Inlining and Alias Analysis/GVN
- Missed optimizations: Extra analysis to manage with false positives
  1. Filter trivially false positives
  2. Expose enough information for quick detection by user
Freebench/distray
(MultiSource/Benchmarks)

Finding Performance Opportunity
static double IntersectObjs( VECTOR *LinP, VECTOR *LinD, 
VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
{
for( objn = 0; objn < NUMOBS; objn++ )

loop not vectorized: vectorization is not beneficial and is not explicitly forced

Pos = objn[objn].pos;
Pos.x = LinP->x; /* Translate object into "line-space" */

failed to hoist load with loop-invariant address because the loop may invalidate its value

Pos.y = LinP->y;

failed to hoist load with loop-invariant address because the loop may invalidate its value

Pos.z = LinP->z;

failed to hoist load with loop-invariant address because the loop may invalidate its value

gtelementptr eliminated by PRE

A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);

failed to hoist load with loop-invariant address because the loop may invalidate its value

failed to hoist load with loop-invariant address because the loop may invalidate its value

failed to hoist load with loop-invariant address because the loop may invalidate its value

load of type double not eliminated in favor of load because it is clobbered by store

load of type double not eliminated in favor of load because it is clobbered by store

B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;

failed to hoist load with loop-invariant address because the loop may invalidate its value

C = (objn[objn].z - objn[objn].z - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;

if ( A = C + B*B ) > 0.0 ) /*...else no hit */

A = sqrt(A);

sqrt will not be inlined into IntersectObjs because its definition is unavailable

if ( ttemp = B - A ) < EPSILON ) ttemp = B + A;

if ( EPSILON < ttemp ) & & ( (tttemp) ||(t<0.0) )

Pnt->x = LinD->x*t; /* Calculate intersection point */

Stores SLP vectorized

Pnt->y = LinD->y*t;

load of type double eliminated in favor of load

load of type double eliminated in favor of load

load of type double not eliminated in favor of load because it is clobbered by store

load of type double not eliminated in favor of load because it is clobbered by store

load of type double not eliminated in favor of load because it is clobbered by store

Norm->x = Pnt->x-Pos.x; /* Calculate surface normal */

failed to hoist load with loop-invariant address because the loop may invalidate its value

load of type double eliminated in favor of fmul
static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
{
for( objn = 0; objn < NUMOBSJS; objn++ )
{
    Pos = obs[objn].pos;
    Pos.x == LinP->x;  /* Translate object into "line-space" */
    Pos.y == LinP->y;
    Pos.z == LinP->z;
    A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
    B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
    C = (objs[objn].z-objn[objn].z - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
    if( A == C + B*B ) > 0.0 ) { /*...else no hit*/
        A = sqrt(A);
        if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
        if( (EPSILON < ttmp) && (ttmp-1) )
            t = ttmp;

        Pnt->x = LinD->x+tt;
        /* Calculate intersection point */
        Pnt->y = LinD->y+tt;
        Pnt->z = LinD->z+tt;
        /* Calculate surface normal */
        Norm->x = Pnt->x-Pos.x;
        Norm->y = Pnt->y-Pos.y;
        Norm->z = Pnt->z-Pos.z;
    }
}
static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
    VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
{
    for( objn = 0; objn < NUMOBJJS; objn++ )
    {
        /* Translate object into "line-space" */
        Pos = objs[objn].pos;
        Pos.x = LinP->x;
        Pos.y = LinP->y;
        Pos.z = LinD->z;
        A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
        C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
        if( (A = C + B*B) > 0.0 ) { /* ...else no hit */
            A = sqrt(A);
            if( (ttmp = B - A) < EPSILON ) ttmp = B + A;
            if( EPSILON<ttmp ) & ( (ttmp-t)|| (t<0.0) )
                t = ttmp;
            Pnt->x = LinD->x*t;
            /* Calculate intersection point */
            Stores SLP vectorized
            Pnt->y = LinD->y*t;
            Pnt->z = LinD->z*t;
            Norm->x = Pnt->x-Pos.x; /* Calculate surface normal */
            Norm->y = Pnt->y-Pos.y;
            Norm->z = Pnt->z-Pos.z;
        }
    }
}
Not modified via LinP, maybe writes through other pointers
static double IntersectObjs( VECTOR *LinD, VECTOR *LinD, VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
{
    for( objn = 0; objn < NUMOBJS; objn++ ) {
        Pos = objn[linf];
        Pos.x = LinD->x;
        /* Translate object into "line-space" */
        failed to hoist load with loop-invariant address because the loop may invalidate its value
        Pos.y = LinD->y;
        failed to hoist load with loop-invariant address because the loop may invalidate its value
        Pos.z = LinD->z;
        failed to hoist load with loop-invariant address because the loop may invalidate its value
        getelementptr eliminated by PRE
        A = 1.0 / ( (LinD->x*LinD->x) + (LinD->y*LinD->y) + (LinD->z*LinD->z) );
        failed to hoist load with loop-invariant address because the loop may invalidate its value
        B = (LinD->x*Pos.x + LinD->y*Pos.y + LinD->z*Pos.z) * A;
        failed to hoist load with loop-invariant address because the loop may invalidate its value
        C = (objn[linf].r-objn[linf].r - Pos.x*Pos.x - Pos.y*Pos.y); //...else no hit*/
        if ( (C + B*B) > 0.0 ) {
            A = sqrt(A);
            sqrt will not be inlined into IntersectObjs because its definition is unavailable
            if (Epsilon < ttmp) ;
            if (EPSILON*ttmp && (tmp-t)<0.0 ) {
                t = ttmp;
                Pnt->x = LinD->x*t;
                /* Calculate intersection point */
                Stores SLP vectorized
                Pnt->y = LinD->y*t;
                failed to hoist load with loop-invariant address because the loop may invalidate its value
                load of type double eliminated in favor of load
                Pnt->z = LinD->z*t;
                failed to hoist load with loop-invariant address because the loop may invalidate its value
                load of type double eliminated in favor of load
                load of type double not eliminated in favor of load because it is clobbered by store
                load of type double not eliminated in favor of load because it is clobbered by store
                Norm->x = Pnt->x-Pos.x;
                /* Calculate surface normal */
                load of type double eliminated in favor of fmul
                failed to hoist load with loop-invariant address because the loop may invalidate its value
                load of type double eliminated in favor of fmul
            }
        }
    }
}
static double IntersectObjs( VECTOR *LinP, VECTOR *LinD,
    VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt )
{
    for( objn = 0; objn < NUMOBS; objn++ ) {
        loop not vectorized: vectorization is not beneficial and is not explicitly forced
        Pos = obj[objn].pos;
        Pos.x = LinP->x;  /* Translate object into "line-space" */
        if( Pos.y > LinP->y; )
            failed to hoist load with loop-invariant address because the loop may invalidate its value
            Pos.x = LinP->z;
        if( Pos.y > LinP->y; )
            failed to hoist load with loop-invariant address because the loop may invalidate its value
            Pos.x = LinP->z;
        if( Pos.y > LinP->y; )
            failed to hoist load with loop-invariant address because the loop may invalidate its value
            load of type double not eliminated in favor of load because it is clobbered by store
        if( Pos.y > LinP->y; )
            failed to hoist load with loop-invariant address because the loop may invalidate its value
            load of type double not eliminated in favor of load because it is clobbered by store
        A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);
        if( (A = C + B*B) > 0.0 ) {
            /* else no hit */
            A = sqrt(A);
        }
        sqnt will not be inlined into IntersectObjs because its definition is unavailable
        if( (tmp = B - A) < EPSILON ) tmp = B + A;
        if( (EPSILON<tmp) && (tmp<0) ) {
            t = tmp;
            Pnt->x = LinD->x*t;  /* Calculate intersection point */
            Pnt->y = LinD->y*t;
            load of type double not eliminated in favor of load
            Pnt->z = LinD->z*t;
            load of type double not eliminated in favor of load
            load of type double not eliminated in favor of load because it is clobbered by store
            load of type double not eliminated in favor of load because it is clobbered by store
            Norm->x = Pnt->x-Pos.x;  /* Calculate surface normal */
            Norm->y = Pnt->y-Pos.y;
            load of type double not eliminated in favor of load
            load of type double not eliminated in favor of load
            load of type double not eliminated in favor of load
            load of type double not eliminated in favor of load
    }
}
reads and writes don't alias
Reads and writes don't alias

Loop versioning with array overlap checks?
LICM-based LoopVersioning
(-enable-loop-versioning-licm)
Performance opportunity if we can improve this pass
Approximate the opportunity by manually modifying the source.
static double IntersectObjs(VECTOR *restrict LinP, VECTOR *restrict LinD, VECTOR *Pt, VECTOR *Norm, TEXTURE ***txt) {

    for (objn = 0; objn < NUMOBS; objn++) {
        Pos = objs[objn].pos;
        Pos.x = LinP->x;       /* Translate object into "line-space" */
        Pos.y = LinP->y;
        Pos.z = LinP->z;

        /* load hoisted */
        /* load hoisted */
        /* load hoisted */
        /* load eliminated by PRE */
        getelementptr eliminated by PRE
        A = 1.0 / (LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z);

        /* load hoisted */
        /* load hoisted */
        /* load hoisted */
        /* load hoisted */
        /* load hoisted */

        B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
        C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;

        if((A = C + B*B) > 0.0) { /* ...else no hit */
            A = sqrt(A);
        }

        if((ttmp = B - A) < EPSILON) ttmp = B + A;
        if((EPSILON<ttmp) && ((ttmp>t)||(t<0.0))) {
            t = ttmp;
            Pt->x = LinD->x*t;
            /* Calculate intersection point */
            Pt->y = LinD->y*t;

        /* failed to hoist load with loop-invariant address because load is conditionally executed */

        /* load of type double eliminated in favor of phi */
        Pnt->z = LinD->z*t;

        /* failed to hoist load with loop-invariant address because load is conditionally executed */

        /* load of type double eliminated in favor of phi */
        Norm->z = Pnt->z - Pos.x;   /* Calculate surface normal */
static double IntersectObjs( VECTOR *restrict LinP, VECTOR *restrict LinD, VECTOR *Pnt, VECTOR *Norm, TEXTURE **txt)
{
    for (objn = 0; objn < NUMOBSJS; objn++) {
        Pos = objs[objn].Pos;
        Pos.x = LinP->x; /* Translate object into "line-space" */
        Pos.y = LinP->y;
        Pos.z = LinP->z;
        A = 1.0 / ((LinD->x*LinD->x + LinD->y*LinD->y + LinD->z*LinD->z) * A);
        B = (Pos.x*LinD->x + Pos.y*LinD->y + Pos.z*LinD->z) * A;
        C = (objs[objn].r*objs[objn].r - Pos.x*Pos.x - Pos.y*Pos.y - Pos.z*Pos.z) * A;
        if ((A = C + B*B) > 0.0) { /* else no hit */
            A = sqrt(A);
            if (ttmp = B - A < EPSILON) ttmp = B + A;
            if (EPSILON<ttmp & & (ttmp=t|||(t<0.0))) {
                t = ttmp;
                Pnt->x = LinD->x*t; /* Calculate intersection point */
                Pnt->y = LinD->y*t;
                Pnt->z = LinD->z*t;
            } else {
                failed to hoist load with loop-invariant address because load is conditionally executed
                load of type double eliminated in favor of phi
                Pnt->x = Pnt->x - Pos.x; /* Calculate surface normal */
Dynamic Instruction Count Reduced by 11%
Performance headroom: 11%
Freebench/distray: Summary

• Found optimization opportunity while staying in the high-level view
• Reconstructed the reason for missed optimization
• High-level view exposed that the gain may be substantial
• Got immediate feedback of the desired effect on the prototype
• Identified the pass for low-level debugging
Check Out More Examples

http://lab.llvm.org:8080/artifacts/opt-view_test-suite
Development Timeline

- Initial version on LLVM trunk
- Now
- Compiler Developer Tool
- Code Author Tool
- New tools using Optimization Records
Compiler Developer Tool: Status

- Written in Python
- Hook up new passes
- Improve diagnostics quality for existing passes
  - Perform extra analysis for insightful messages
- Improve UI
Compiler Developer Tool: Status

- Written in Python
- Hook up new passes
- Improve diagnostics quality for existing passes
- Perform extra analysis for insightful messages
- Improve UI
Code Author Tool: Wishlist

• Suggest specific actions

• E.g. for the LICM case: if the two pointers can never point to the same object consider using ‘restrict’

• Add new “recommendation” analysis passes to detect opportunity and suggest:
  • Source annotation to enable off-by-default passes (aggressive loop transformations, non-temporal stores)
  • Refactoring: data transformations
Code Author Tool: Wishlist

• Suggest specific actions

• E.g. for the LICM case: if the two pointers can never point to the same object consider using ‘restrict’

• Add new “recommendations” analysis passes to detect opportunity and suggest:
  • Source annotation to enable off-by-default passes (aggressive loop transformations, non-temporal stores)
  • Refactoring: data transformations
Optimization Records: New Tools

- `llvm-opt-report`
- Performance regression analysis
- Optimization statistics with the ability to zoom into the particular optimization
- Bottom-up search for performance opportunities
  - See all the LICM opportunities like in Freebench/distray
Optimization Records: New Tools

- **llvm-opt-report**
- Performance regression analysis
- Optimization statistics with the ability to zoom into the particular optimization
- Bottom-up search for performance opportunities
  
```
SELECT benchmark, hotspot, hotness
FROM optimizations
WHERE pass = 'licm' AND type = 'missed' AND name = ‘LoadWithLoopInvariantAddressInvalidated'
ORDER BY hotness
```  
- See all the LICM opportunities like in Freebench/distray
Optimization Records: New Tools

- `llvm-opt-report`
- Performance regression analysis
- Optimization statistics with the ability to zoom into the particular optimization
- Bottom-up search for performance opportunities
  - See all the LICM opportunities like in Freebench/distray
  - Allows finding opportunities that occur with high frequency but not in the hottest code
Acknowledgement

• Tyler Nowicki
• John McCall
• Hal Finkel
Q&A
SIBsim4
(MultiSource/Applications)

Finding Performance Opportunity
for (; i = 1; j < stop; j++) {
    loop not vectorized: loop control flow is not understood by vectorizer
    loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
    exon_p_t n = mCol->e.exon[j];
    failed to hoist load with loop-invariant address
    load eliminated by PRE
    failed to hoist load with loop-invariant address
    if (lies_after_p(m, n) && m->Score >= n->Score) {
        lies_after_p can be inlined into link_msp with cost=-14785 (threshold=325)
        lies_after_p inlined into link_msp
        failed to hoist load with loop-invariant address
        failed to hoist load with loop-invariant address
        unsigned int penalty;
        penalty = abs(n->from1 - m->from1) >> 15;
        getelementptr hoisted
        failed to hoist load with loop-invariant address
        load of type i32 eliminated
        failed to hoist load with loop-invariant address
        penalty += abs(n->from2 - m->from2) >> 15;
        getelementptr hoisted
        failed to hoist load with loop-invariant address
        load of type i32 eliminated
        failed to hoist load with loop-invariant address
        if (penalty < m->Score) {
            n->Score = m->Score - penalty;
            n->prev = i;
            shi hoisted
            and hoisted
        }
    }
}
for (j = 1; j < stop; j++) {
  loop not vectorized: loop control flow is not understood by vectorizer
  loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
  exon_p_t n = mCol->e.exon[j];

  if (lies_after_p(m, n) && m->Score >= n->Score) {
    lies_after_p n can be inlined into link_msp with cost=-14785 (threshold=325)
    lies_after_p n inlined into link_msp

    unsigned int penalty;
    penalty = abs(n->from1 - m->from1) >> 15;

    getelementptr hoisted
    failed to hoist load with loop-invariant address
    load of type i32 eliminated
    failed to hoist load with loop-invariant address
    penalty += abs(n->from2 - m->from2) >> 15;

    getelementptr hoisted
    failed to hoist load with loop-invariant address
    load of type i32 eliminated
    failed to hoist load with loop-invariant address

    if (penalty < m->Score) {
      n->Score = m->Score - penalty;
      n->prev = i;
      shi hoisted
      and hoisted
    }
  }
}
```c
for (j = i + 1; j < stop; j++) {
    100% loop-vectorize
    loop not vectorized: loop control flow is not understood by vectorizer
    100% loop-vectorize
    loop not vectorized: use -Rpass-analysis=loop-vectorize for more info

    exon_p_t n = mCol->e.exon[j];

    99% ilcm
    failed to hoist load with loop-invariant address
    100% gvn
    load eliminated by PRE
    99% ilcm
    failed to hoist load with loop-invariant address

    if (lies_after_p(m, n) && m->Score >= n->Score) {

        97% inline
        lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)

        97% inline
        lies_after_p inlined into link_msps

        55% ilcm
        failed to hoist load with loop-invariant address

        55% ilcm
        failed to hoist load with loop-invariant address

        unsigned int penalty;
        penalty = abs(n->from1 - m->from1) >> 15;

        3% ilcm
        getelementptr hoisted

        3% ilcm
        failed to hoist load with loop-invariant address

        3% gvn
        load of type i32 eliminated

        3% ilcm
        failed to hoist load with loop-invariant address

        penalty += abs(n->from2 - m->from2) >> 15;

        3% ilcm
        getelementptr hoisted

        3% ilcm
        failed to hoist load with loop-invariant address

        3% gvn
        load of type i32 eliminated

        3% ilcm
        failed to hoist load with loop-invariant address

        if (penalty < m->Score) {
            n->Score = m->Score - penalty;
            n->prev = i;

            3% ilcm
            shl hoisted

            3% ilcm
            and hoisted
        }
    }
}
```
```c
for (i = 1; i < stop; i++) {
    // loop not vectorized: loop control flow is not understood by vectorizer
    // loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
    exon_p_t n = mCol->e.exon[i];
    failed to hoist load with loop-invariant address
    load eliminated by PRE
    failed to hoist load with loop-invariant address
    if (lies_after_p(m, n) && m->Score >= n->Score) {
        // lies_after_p can be inlined into link_mspss with cost=-14785 (threshold=325)
        li cm
        failed to hoist load with loop-invariant address
        failed to hoist load with loop-invariant address
        unsigned int penalty;
        penalty = abs(n->from1 - m->from1) >> 15;
        // getelementptr hoisted
        failed to hoist load with loop-invariant address
        load of type int32 eliminated
        failed to hoist load with loop-invariant address
        penalty += abs(n->from2 - m->from2) >> 15;
        // getelementptr hoisted
        failed to hoist load with loop-invariant address
        load of type int32 eliminated
        failed to hoist load with loop-invariant address
        if (penalty < m->Score) {
            n->Score = m->Score - penalty;
            n->prev = i;
            // shi hoisted
            and hoisted
        }
    }
}
```
for (j = 1; j < stop; j++) {
    loop vectorize
    loop not vectorized: loop control flow is not understood by vectorizer
    loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
    exon_p_t n = mCol->e.exon[j];
    if (lies_after_p(m, n) && m->Score >= n->Score) {
        inline
        lies_after_p can be inlined into link_msp with cost=14785 (threshold=325)
        lies_after_p inlined into link_msp
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        load of type i32 eliminated
        gvn
        llicm
        failed to hoist load with loop-invariant address
        llicm
        getelementptr hoisted
        llicm
        load of type i32 eliminated
        gvn
        llicm
        failed to hoist load with loop-invariant address
        llicm
        failed to hoist load with loop-invariant address
        llicm
        load of type i32 eliminated
        gvn
        llicm
        failed to hoist load with loop-invariant address
        llicm
        penalty += abs(n->from2 - m->from2) >> 15;
        if (penalty < m->Score) {
            n->Score = m->Score - penalty;
            n->prev = i;
            llicm
            shi hoisted
            llicm
            and hoisted
        }
    }
}
SIBsim4

```c
for (; i = i + 1; j < stop; j++) {
    loop not vectorized: loop control flow is not understood by vectorizer
    loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
    exon_p_t n = mCol->e.exon[j];
    if (lies_after_p(m, n) && m->Score >= n->Score) {
        lies_after_p can be inlined into link_msps with cost=-14785 (threshold=325)
        lies_after_p inlined into link_msps
    }
    penalty = abs(n->from1 - m->from1) >> 15;
    getelementptr hoisted
    failed to hoist load with loop-invariant address
    load of type i32 eliminated
    failed to hoist load with loop-invariant address
    penalty += abs(n->from2 - m->from2) >> 15;
    getelementptr hoisted
    failed to hoist load with loop-invariant address
    load of type i32 eliminated
    failed to hoist load with loop-invariant address
    if (penalty < m->Score) {
        n->Score = m->Score - penalty;
        n->prev = i;
        shl hoisted
        and hoisted
    }
}
```
for (; i = 1; j < stop; j++) {
  loop not vectorized: loop control flow is not understood by vectorizer
  loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
  exon_p_t n = mCol->e.exon[j];
  llicm
  failed to hoist load with loop-invariant address
  gvn
  load eliminated by PRE
  llicm
  failed to hoist load with loop-invariant address
  if (lies_after_p(m, n) && m->Score = n->Score) {
    llicm
    llicm
    llicm
    llicm
    failed to hoist load with loop-invariant address
    failed to hoist load with loop-invariant address
    unsigned int penalty;
    penalty = abs(n->from1 m->from1 >> 15;
    getelementptr hoisted
    failed to hoist load with loop-invariant address
    gvn
    load of type i32 eliminated
    llicm
    failed to hoist load with loop-invariant address
    penalty += abs(n->from2 m->from2 >> 15;
    getelementptr hoisted
    failed to hoist load with loop-invariant address
    gvn
    load of type i32 eliminated
    llicm
    failed to hoist load with loop-invariant address
    if (penalty < m->Score) {
      n->Score = m->Score - penalty;
      n->prev = i;
      shi hoisted
      and hoisted
    }
  }
}
static inline int
lisa_after_p(int s, int b)
{
    /* When we have some overlap, make sure it is only a small part. */
    
    p1 | p2 | p3

    if (b->from1 != a->to1)
        getelementptr hostsid
    if (b->from1 == a->to1)
        getelementptr hostsid

    return 1;

    if (b->from2 != a->to2)
        getelementptr hostsid
    if (b->from2 == a->to2)
        getelementptr hostsid

    return 1;

    if (b->from3 != a->to3)
        getelementptr hostsid
    if (b->from3 == a->to3)
        getelementptr hostsid

    return 1;
SIBsim4

Look at the stores
for (i = i + 1; i < stop; i++) {
  loop not vectorized: loop control flow is not understood by vectorizer
  loop not vectorized: use -Rpass-analysis=loop-vectorize for more info
  exon_p_t n = mCol->e.exon[i];

  if (lies_after_p(m, n) && m->Score >= n->Score) {
    if (m->Score > n->Score)
      lcc
  }

  if (m->Score < n->Score) {
    n->Score = m->Score - penalty;
    n->prev = i;
  }
}
Can ‘m’ and ‘n’ really alias?
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>846</td>
<td><code>for (; i++ &lt; stop; j++) {</code></td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td><code>exon_p_t m = mCol-&gt;e.exon[i];</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>loop not vectorized; loop control flow is not understood by vectorizer;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>exon_p_t n = mCol-&gt;e.exon[j];</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Probably not!</code></td>
<td></td>
</tr>
<tr>
<td>847</td>
<td><code>for (; i++ &lt; stop; j++) {</code></td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td><code>exon_p_t m = mCol-&gt;e.exon[i];</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>loop not vectorized; loop control flow is not understood by vectorizer;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>exon_p_t n = mCol-&gt;e.exon[j];</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>if (lies_after_p(m, n) &amp;&amp; m-&gt;Score &gt;= n-&gt;Score) {</code></td>
<td>97%</td>
</tr>
<tr>
<td></td>
<td><code>lies_after_p can be inlined into link_msp5 with cost=-14785 (threshold=325)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load eliminated by PRE</code></td>
<td></td>
</tr>
<tr>
<td>848</td>
<td><code>if (lies_after_p(m, n) &amp;&amp; m-&gt;Score &gt;= n-&gt;Score) {</code></td>
<td>97%</td>
</tr>
<tr>
<td></td>
<td><code>lies_after_p can be inlined into link_msp5 with cost=-14785 (threshold=325)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load eliminated by PRE</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>unsigned int penalty;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>penalty = abs(n-&gt;from1 - m-&gt;from1) &gt;&gt; 15;</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>getelementptr hoisted</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load of type i32 eliminated</code></td>
<td></td>
</tr>
<tr>
<td>849</td>
<td><code>if (lies_after_p(m, n) &amp;&amp; m-&gt;Score &gt;= n-&gt;Score) {</code></td>
<td>97%</td>
</tr>
<tr>
<td></td>
<td><code>lies_after_p can be inlined into link_msp5 with cost=-14785 (threshold=325)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load eliminated by PRE</code></td>
<td></td>
</tr>
<tr>
<td>850</td>
<td><code>if (lies_after_p(m, n) &amp;&amp; m-&gt;Score &gt;= n-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>lies_after_p can be inlined into link_msp5 with cost=-14785 (threshold=325)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load of type i32 eliminated</code></td>
<td></td>
</tr>
<tr>
<td>851</td>
<td><code>if (lies_after_p(m, n) &amp;&amp; m-&gt;Score &gt;= n-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>lies_after_p can be inlined into link_msp5 with cost=-14785 (threshold=325)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>failed to hoist load with loop-invariant address</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>load of type i32 eliminated</code></td>
<td></td>
</tr>
<tr>
<td>852</td>
<td><code>if (penalty &lt; m-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;Score = m-&gt;Score - penalty;</code></td>
<td></td>
</tr>
<tr>
<td>853</td>
<td><code>n-&gt;prev = i;</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>shl hoisted</code></td>
<td></td>
</tr>
<tr>
<td>854</td>
<td><code>if (penalty &lt; m-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;Score = m-&gt;Score - penalty;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;prev = i;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>and hoisted</code></td>
<td></td>
</tr>
<tr>
<td>855</td>
<td><code>if (penalty &lt; m-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;Score = m-&gt;Score - penalty;</code></td>
<td></td>
</tr>
<tr>
<td>856</td>
<td><code>n-&gt;prev = i;</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>and hoisted</code></td>
<td></td>
</tr>
<tr>
<td>857</td>
<td><code>if (penalty &lt; m-&gt;Score) {</code></td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;Score = m-&gt;Score - penalty;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>n-&gt;prev = i;</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>and hoisted</code></td>
<td></td>
</tr>
</tbody>
</table>
We need to use ‘restrict’ or hoist manually
exon_t mm = *m;

for (j = i + 1; j < stop; j++) {
    exon_p_t n = mCol->e.exon[j];
    if (lies_after_p(&mm, n) && mm.Score >= n->Score) {
        unsigned int penalty;
        penalty = abs(n->from1 - mm.from1) >> 15;
        penalty += abs(n->from2 - mm.from2) >> 15;
        if (penalty < mm.Score) {
            n->Score = mm.Score - penalty;
            n->prev = i;
        }
    }
}
SIBsim4

```c
exon_t mm = *m;
for (j = i + 1; j < stop; j++) {
    exon_p_t n = mCol->e.exon[j];
    if (lies_after_p(&mm, n) && mm.Score >= n->Score) {
        unsigned int penalty;
        penalty = abs(n->from1 - mm.from1) >> 15;
        penalty += abs(n->from2 - mm.from2) >> 15;
        if (penalty < mm.Score) {
            n->Score = mm.Score - penalty;
            n->prev = i;
        }
    }
}
```
exn_t mm = m;

for (j = i + 1; j < stop; j++) {

loop not vectorized: loop control flow is not understood by vectorizer
loop not vectorized: use -Rpass-analysis=loop-vectorize for more info

exon_p_t n = nCol->exon[j];

failed to hoist load with loop-invariant address because the loop may invalidate its value

load of type %struct.exon_t** not eliminated in favor of load because it is clobbered by store
load eliminated by PRE
load eliminated by store
failed to hoist load with loop-invariant address because the loop may invalidate its value

if (lie_after_p(mm, n) && mm.Score >= n->Score) {

unsigned int penalty;
penalty = abs(n->from1 - mm.from1) >> 15;
load of type i32 eliminated
penalty += abs(n->from2 - mm.from2) >> 15;
load of type i32 eliminated

if (penalty < mm.Score) {

n->Score = mm.Score - penalty;
}

ahl hoisted
and hoisted
}
static inline int
lies_after_p(exon_p_t a, exon_p_t b)
{
    /* When we have some overlap, make sure it is only a small part. */
    /* ------------------------------
   | p1 | p2 | p3 |
   ------------------------------
    */

    if (b->from1 > a->to1) {
        unsigned int p1;
        unsigned int p2;
        unsigned int p3;
        if (b->from2 > a->to2)
            return 1;
        if (b->from2 < a->from2 || b->to2 < a->to2)
            return 1;
        p1 = b->from2 - a->from2;
        p2 = a->to2 - b->from2;
        p3 = b->to2 - a->to2;
        if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
            return 1;
    } else if (b->from2 > a->to2) {
        unsigned int p1;
        unsigned int p2;
        unsigned int p3;
        if (b->from1 < a->from1 || b->to1 < a->to1)
            return 0;
        p1 = b->from1 - a->from1;
        p2 = a->to1 - b->from1;
        p3 = b->to1 - a->to1;
        if (p1 > p2 && p3 > p2 && p1 > options.K && p3 > options.K)
            return 1;
    }
    return 0;
}