IVOPT Implementation & Challenges

Bin Cheng
bin.cheng@arm.com

2015-07-30

The Architecture for the Digital World®
ARM

Outline

- General idea in GCC
- Implementation details
- Challenges

ARM

General idea – Looks good

- Algorithm
  - Find iv uses
  - Add possible iv candidates
  - Compute costs
  - Find optimal set for iv candidates
  - Rewrite program
- Find iv uses

```
int a[1024];
int bar (int);

void
foo (int s, int l, int j)
{
  int i = s;
  for (; i < l; i++)
    a[i] = bar (i + j);
}
```

```
use 1 (address)

in stmt a[s_14] = _9;
type int *
base (int*)(&a + (sizetype)s_3(D)*4)
step 4
```

```
foo (int s, int l, int j)
{
  int _7;
  int _9;

  <bb 2>:
  if (s_3(D) < l_5(D))
    goto <bb 4>;
  else
    goto <bb 3>;

  <bb 3>:
  return;

  <bb 4>:

  <bb 5>:
  # s_14 = PHI <s_3(D)(4), s_11(7)>
  _7 = j_6(D) + s_14;
  _9 = bar (_7);
  a[s_14] = _9;
  s_11 = s_14 + 1;
  if (l_5(D) > s_11)
    goto <bb 7>;
  else
    goto <bb 6>;

  <bb 6>:
  goto <bb 3>;

  <bb 7>:
  goto <bb 5>;
}
```

```
use 0 (generic)

in stmt _7 = j_6(D) + s_14;
type int
base s_3(D) + j_6(D)
step 1
```

```
use 2 (compare)

in stmt if (l_5(D) > s_11)
type int
base s_3(D) + 1
step 1
```


ARM

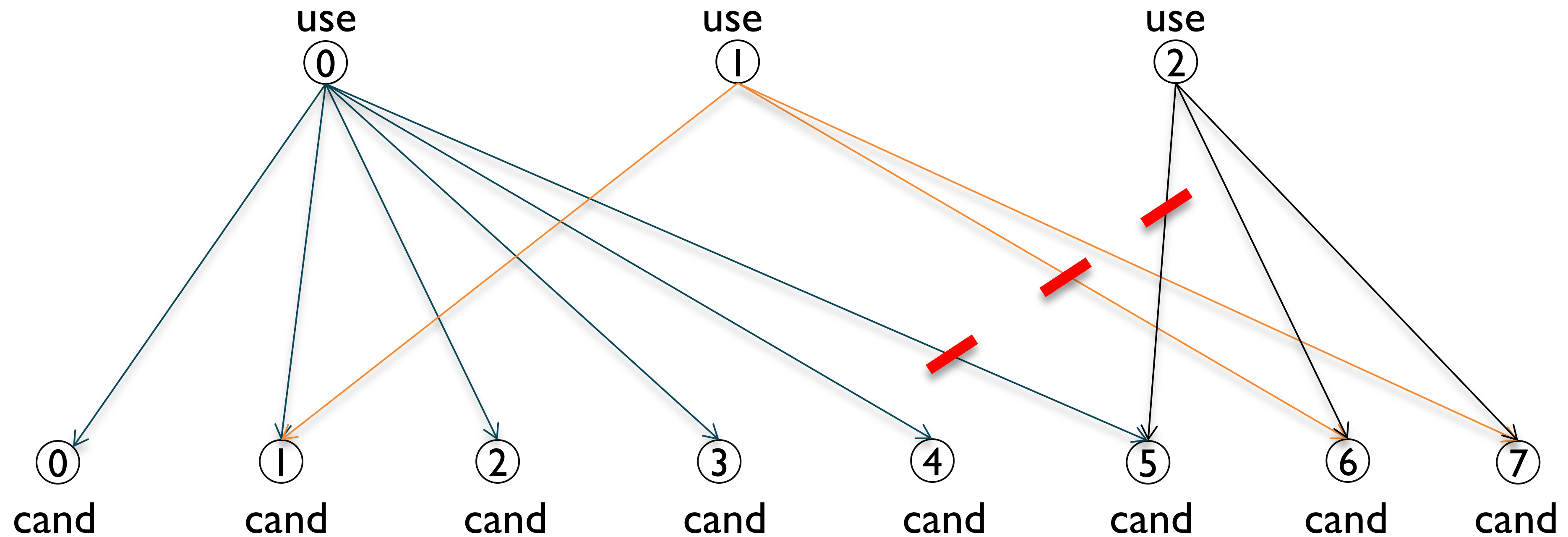
General idea

- Uses
  - `{s_3(D) + j_6(D), 1}                    ; int`
  - `{(int *) (&a + (sizetype) s_3(D) * 4), 4}  ; int*`
  - `{s_3(D) + 1, 1}                         ; int`
- Add iv candidates
  - `{0, 1}                                  ; unsigned int`
  - `{0, 1}                                  ; unsigned long`
  - `{(unsigned int) (s_3(D) + 1), 1}        ; unsigned int`
  - `{(unsigned int) s_3(D), 1}              ; unsigned int`
  - `{s_3(D), 1}                             ; int`
  - `{(unsigned int) (s_3(D) + j_6(D)), 1}   ; unsigned int`
  - `{(unsigned long) (&a + (sizetype) s_3(D) * 4), 4}  ; unsigned long`
  - `{0, 4}                                  ; sizetype`

ARM

General idea

- Compute costs



- Find optimal set for iv cands
  - Cands 5, 6

ARM

General idea

- Rewrite program

```
<bb 4>:

<bb 5>:
# s_14 = PHI <s_3(D)(4), s_11(7)>
_7 = j_6(D) + s_14;
_9 = bar (_7);
a[s_14] = _9;
s_11 = s_14 + 1;
if (l_5(D) > s_11)
  goto <bb 7>;
else
  goto <bb 6>;

<bb 7>:
goto <bb 5>;
}
```

```
<bb 4>:
_17 = s_3(D) + j_6(D);
ivtmp.7_12 = (unsigned int) _17;
_21 = (sizetype) s_3(D);
_22 = _21 * 4;
_23 = &a + _22;
ivtmp.8_20 = (unsigned long) _23;
_25 = (unsigned int) l_5(D);
_26 = (unsigned int) j_6(D);
_27 = _25 + _26;

<bb 5>:
# ivtmp.7_2 = PHI <ivtmp.7_12(4), ivtmp.7_13(7)>
# ivtmp.8_18 = PHI <ivtmp.8_20(4), ivtmp.8_19(7)>
_7 = (int) ivtmp.7_2;
_9 = bar (_7);
_24 = (void *) ivtmp.8_18;
MEM[base: _24, offset: 0B] = _9;
ivtmp.7_13 = ivtmp.7_2 + 1;
ivtmp.8_19 = ivtmp.8_18 + 4;
if (ivtmp.7_13 != _27)
  goto <bb 7>;
else
  goto <bb 6>;

<bb 7>:
goto <bb 5>;
```


ARM

Implementation details – Find iv uses

- Find normal iv uses
- Find comparison iv uses
- Find address type iv uses
  - pr52563, pr62173, pr48052, etc.
- Examples
  - &a[i] is an IV
  - &a[k] is not an IV

```
int a[1000];
void
foo (unsigned char s, unsigned char l){
  int sum;
  unsigned char i;
  for (i = s; i < l; i++)
    a[i] = 0;
}

void
bar (long long s, long long l){
  long long i;
  for (i = s; i < l; i++){
      k = (unsigned char)i;
      a[k] = 0;
    }
}
```


ARM

Implementation details – Add iv candidates

- As a matter of fact, it's quite primitive how gcc adds iv candidates
  - By guessing from single iv use
- Defects
  - Useful candidates are not added – PR52272
  - Useless candidates are added
    - PR65447
    - Auto-increment addressing mode support
  - Increase compilation time
  - Mess up candidate selecting algorithm
- Challenge
  - Add candidates by studying (address) iv uses
  - Mostly about exercising CSE opportunities for address type iv uses
  - Target dependent issue

```
void foo (double *p) {
  int i;
  for (i = -20000; i < 200000; i+= 40) {
      p[i+0] = 1.0;
      p[i+1] = 1.0;
      p[i+2] = 1.0;
      p[i+3] = 1.0;
      // ...
      p[i+36] = 1.0;
      p[i+37] = 1.0;
      p[i+38] = 1.0;
      p[i+39] = 1.0;
  }
}
Use0: {b_a + ((inv_x+inv_y) + const)*8, 8}
Use1: {b_b + ((inv_x+inv_y) + const)*8, 8}
Use2: {b_c + ((inv_x+inv_y) + const)*8, 8}
Use3: {b_d + ((inv_x+inv_y) + const)*8, 8}
Use4: {b_e + ((inv_x+inv_y) + const)*8, 8}
Use5: {b_f + ((inv_x+inv_y) + const)*8, 8}
```

ARM

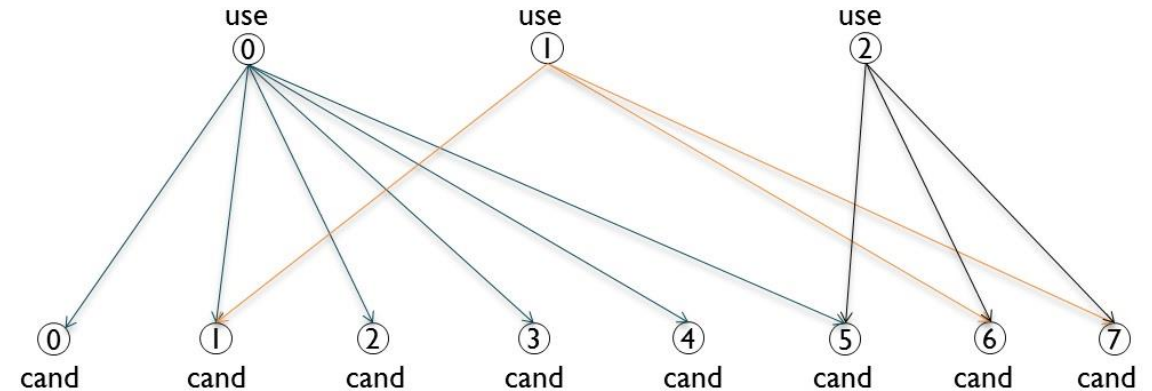
Implementation details – Cost computation

- Compute cost at early compilation stage
  - Compute RTL cost on GIMPLE
  - Now we lower high level IR like
    - {&Arr[inv + IV]. y, 8} → {Arr + (inv + IV) * 8 + 4, 8}
  - Cache RTL pattern/cost for saving compilation time
- Model target features
  - Auto-increment addressing mode support
    - By adding cand which incremented before/after memory access
  - Do-loop structure support
    - Not implemented yet – pr66612
- Model register pressure
  - The only tree optimizer with sophisticated reg-pressure computed
  - It DOES help a lot
  - Yet needs to be improved

ARM

Implementation details – Find optimal set for iv cands

- The heuristic algorithm
  - Start with small iv cand set comprising general ivs
  - Replace expensive use with new iv cand
  - Prune iv cand set to keep it small
  - Try to break local optimal solution
- Challenges
  - Costs are entangled together
    - The use costs. The variable cost. Register pressure cost
  - Complexity issue
    - Parameters bound on number of iv uses and iv cands



ARM

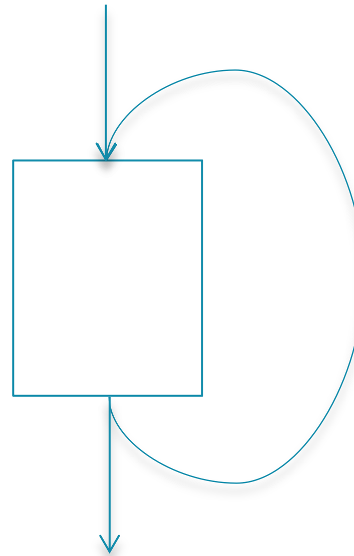
Implementation details – Rewrite program

- Compute iv use with selected iv cand
- Replace old use with computed variable/expression
- Remove dead code

ARM

Implementation details – Reg-Pressure

- Coarse-grained register pressure model
  - Don't model reg-pressure at every program point
  - Don't model reg-pressure for every register class
    - Model reg-pressure for int/fp
  - Model reg-pressure on the basis of LOOP
    - Live in:
      - {s_4, 0(sum), v2, v3, l}
    - Live out:
      - {sum_18}
    - Live through:
      - {v2, v4, v5}
    - Eliminable vars
      - {s_4, l, v2, v3}
    - New vars
      - Introduced by IVopts

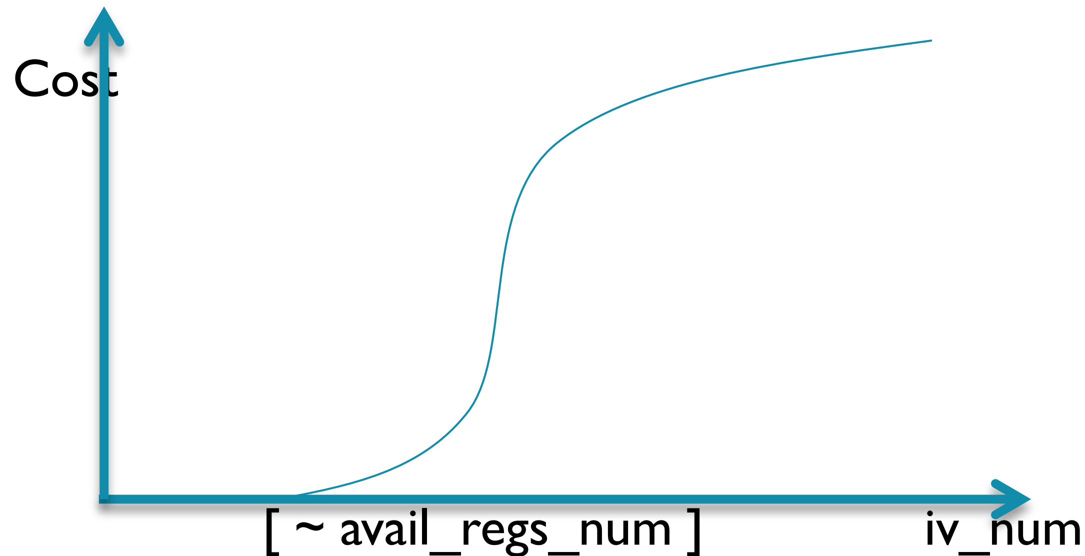


```
f (int s, int l, int *v1, int *v2, int *v3, int *v4, int v5)
{
  <bb 2>:
  if (s_4(D) < l_6(D))
    goto <bb 3>;
  else
    goto <bb 7>;

  <bb 3>:
  <bb 4>:
  # s_28 = PHI <s_4(D)(3), s_20(5)>
  # sum_29 = PHI <0(3), sum_18(5)>
  _7 = (long unsigned int) s_28;
  _8 = _7 * 4;
  _10 = v1_9(D) + _8;
  *_10 = 0;
  _13 = v2_12(D) + _8;
  *_13 = 1;
  _16 = v3_15(D) + _8;
  _17 = *_16;
  sum_18 = _17 + sum_29;
  bar (v3_15(D));
  s_20 = s_28 + 1;
  if (l_6(D) > s_20)
    goto <bb 5>;
  else
    goto <bb 6>;

  <bb 5>:
  goto <bb 4>;

  <bb 6>:
  # sum_27 = PHI <sum_18(4)>

  <bb 7>:
  # sum_30 = PHI <sum_27(6), 0(2)>
  *v2_12(D) = v5_21(D);
  *v4_23(D) = sum_30;
  return;
}
```

ARM

Challenges – Reg-Pressure

- Failed to restrict the number of iv set
  - Reg-pressure is inaccurate
  - Cost function is inaccurate
  - Cost function fails at high reg-pressure




```
# ivtmp.319_3663 = PHI <ivtmp.319_3658(41), ivtmp.319_16(43)>
# ivtmp.321_3654 = PHI <ivtmp.321_3652(41), ivtmp.321_3653(43)>
# ivtmp.323_3648 = PHI <ivtmp.323_3646(41), ivtmp.323_3647(43)>
# ivtmp.325_3642 = PHI <ivtmp.325_3640(41), ivtmp.325_3641(43)>
# ivtmp.327_3636 = PHI <ivtmp.327_3634(41), ivtmp.327_3635(43)>
# ivtmp.329_3630 = PHI <ivtmp.329_3628(41), ivtmp.329_3629(43)>
# ivtmp.331_3624 = PHI <ivtmp.331_3622(41), ivtmp.331_3623(43)>
# ivtmp.333_3618 = PHI <ivtmp.333_3616(41), ivtmp.333_3617(43)>
# ivtmp.335_3612 = PHI <ivtmp.335_3610(41), ivtmp.335_3611(43)>
# ivtmp.337_3606 = PHI <ivtmp.337_3604(41), ivtmp.337_3605(43)>
# ivtmp.339_3600 = PHI <ivtmp.339_3598(41), ivtmp.339_3599(43)>
# ivtmp.341_3594 = PHI <ivtmp.341_3592(41), ivtmp.341_3593(43)>
# ivtmp.435_3312 = PHI <ivtmp.435_3310(41), ivtmp.435_3311(43)>
# ivtmp.437_3306 = PHI <ivtmp.437_3304(41), ivtmp.437_3305(43)>
# ivtmp.439_3300 = PHI <ivtmp.439_3298(41), ivtmp.439_3299(43)>
# ivtmp.441_3294 = PHI <ivtmp.441_3292(41), ivtmp.441_3293(43)>
# ivtmp.443_3288 = PHI <ivtmp.443_3286(41), ivtmp.443_3287(43)>
# ivtmp.445_3282 = PHI <ivtmp.445_3280(41), ivtmp.445_3281(43)>
# ivtmp.447_3276 = PHI <ivtmp.447_3274(41), ivtmp.447_3275(43)>
# ivtmp.449_3270 = PHI <ivtmp.449_3268(41), ivtmp.449_3269(43)>
# ivtmp.451_3264 = PHI <ivtmp.451_3262(41), ivtmp.451_3263(43)>
# ivtmp.453_3258 = PHI <ivtmp.453_3256(41), ivtmp.453_3257(43)>
# ivtmp.455_3252 = PHI <ivtmp.455_3250(41), ivtmp.455_3251(43)>
# ivtmp.457_3246 = PHI <ivtmp.457_3244(41), ivtmp.457_3245(43)>
......// another 49 ivs not shown here.
```

- ?Use limited register budget wisely?
- Expand reg-pressure to other tree optimizers

ARM

Challenges – More precise overflow info

- Can recognize more address type iv uses
- Can't represent iv use with cand of smaller precision
  - {arr + t * 4, 4}_pointertype can't be represented by {0, 1}_(u)int
  - Worse with overflow change
  - Even worse for target support [base + idx << shift], but [base + idx << shift + offset] addressing mode
- Simplify computation of iv use
  - Now in unsigned type
- Conflict between overflow and IV elimination
  - Overflow info is wrto loop niter
  - IV elimination needs to compute one step beyond loop niter

```
int flag;
int arr[144];
void bar (int a, int b);

int foo (int t)
{
  int step = t;
  do
    {
      if (!flag)
        bar (t, 0);
      t += step;
    }
  while (arr[t] != 0);

  return t;
}
```


ARM

Challenges – Rewrite general iv use

- At where it is used – pr18316

```
<bb 2>:
 k_5 = *data_4(D);
 _18 = (unsigned int) k_5;
 ivtmp.7_17 = (unsigned int) k_5;

 <bb 3>:
 # ivtmp.7_20 = PHI <ivtmp.7_17(2), ivtmp.7_19(4)>
 _6 = MEM[(int *)data_4(D) + 8B];
 _7 = (long unsigned int) _6;
 _8 = _7 * 4;
 _9 = data_4(D) + _8;
 *_9 = 2;
 _12 = (int) ivtmp.7_20;
 _13 = MEM[(int *)data_4(D) + 4B];
 ivtmp.7_19 = ivtmp.7_20 + _18;
 if (_12 < _13)
   goto <bb 3>;
 else
   goto <bb 5>;
```

```
<bb 2>:
k_5 = *data_4(D);
_18 = (unsigned int) k_5;

<bb 3>:
# ivtmp.8_20 = PHI <0(2), ivtmp.8_19(4)>
_6 = MEM[(int *)data_4(D) + 8B];
_7 = (long unsigned int) _6;
_8 = _7 * 4;
_9 = data_4(D) + _8;
*_9 = 2;
_13 = MEM[(int *)data_4(D) + 4B];
ivtmp.8_19 = ivtmp.8_20 + _18;
_17 = (int) ivtmp.8_19;
if (_17 < _13)
  goto <bb 3>;
else
  goto <bb 5>;
```


ARM

Challenges – Rewrite general iv use

- Along exit edge – gcc.dg/tree-ssa/scev-4.c

```
f:
        cmp  w0,  999
        bgt  .L1
        sbfiz    x3,  x0,  3,  32
        adrp x1,  a
        add  x1,  x1,  :lo12:a
        add  x4,  x3,  4
        mov  w2,  w0
        mov  w5,  100
        add  x1,  x4,  x1
        .p2align  2
.L3:
        add  w2,  w2,  w0
        str  w5,  [x1]
        mov  x4,  x1
        cmp  w2,  999
        add  x1,  x1,  x3
        ble  .L3
        adrp x0,  a_p
        str  x4,  [x0,  #:lo12:a_p]
.L1:
        ret
```

```
f:
        cmp  w0,  999
        bgt  .L1
        sxtw x6,  w0
        adrp x1,  a
        add  x2,  x1,  :lo12:a
        mov  w3,  w0
        mov  w5,  100
        lsl  x4,  x6,  3
        add  x1,  x4,  4
        add  x1,  x1,  x2
        .p2align  2
.L3:
        add  w3,  w3,  w0
        str  w5,  [x1]
        add  x1,  x1,  x4
        cmp  w3,  999
        ble  .L3
        sub  x1,  x1,  x6,  lsl  3
        adrp x0,  a_p
        str  x1,  [x0,  #:lo12:a_p]
.L1:
        ret
```


ARM

Challenges – Rewrite address iv use

- In a CSE/LIM-oriented approach

```
_3430 = (sizetype) ivtmp.315_3663;
_3429 = ivtmp.531_3286;
_3428 = _3429 + _3430;
_594 = MEM[base: lgxy_102, index: _3428, step: 8, offset: 0B];
_3427 = (sizetype) ivtmp.315_3663;
_3426 = ivtmp.532_3279;
_3425 = _3426 + _3427;
_595 = MEM[base: lgxy_102, index: _3425, step: 8, offset: 0B];
_596 = _594 - _595;
_3424 = (sizetype) ivtmp.315_3663;
_3423 = ivtmp.533_3272;
_3422 = _3423 + _3424;
_597 = MEM[base: lgxy_102, index: _3422, step: 8, offset: 0B];
_598 = _596 - _597;
_3421 = (sizetype) ivtmp.315_3663;
_3420 = ivtmp.534_3266;
_3419 = _3420 + _3421;
_599 = MEM[base: lgxy_102, index: _3419, step: 8, offset: 0B];
```

```
ldr x7, [x29, 936]
add x10, x7, x1
ldr d12, [x12, x10, lsl 3]
ldr x0, [x29, 920]
add x0, x0, x1
ldr d10, [x12, x0, lsl 3]
ldr x2, [x29, 912]
add x2, x2, x1
ldr d11, [x12, x2, lsl 3]
ldr x2, [x29, 904]
add x3, x2, x1
ldr d0, [x12, x3, lsl 3]
```


ARM

Challenges – Rewrite comparison iv use

- Simplify expression using loop bound/initial conds
- Cand : unsigned long
  - {(unsigned long)(a + (sizetype)s_3 * 4), 4}
- Loop niter : unsigned char
  - (l_4 – s_3) – 1
- Cand value after niter iteratations
  - (a + (((sizetype)s_3 + (sizetype) ((l_4 –s_3) - 1)) + 1) * 4)
  - equals to (a + (sizetype)l_4 * 4)
- Prerequisite conditions
  - ((l_4 – s_3) – 1) + 1 == (l_4 – s_3)
  - (sizetype)(l_4 – s_3) == (sizetype)l_4 – (sizetype)s_3

```
<bb 3>:
pretmp_22 = a;
_14 = (long unsigned int) s_3(D);
_13 = _14 * 4;
_2 = pretmp_22 + _13;
ivtmp.9_18 = (unsigned long) _2;
```

```
_23 = (sizetype) s_3(D);
_6 = l_4(D) - s_3(D);
_24 = _6 + 255;
_25 = (sizetype) _24;
_26 = _23 + _25;
_27 = _26 + 1;
_28 = _27 * 4;
_29 = pretmp_22 + _28;
_30 = (unsigned long) _29;
```

```
_23 = (unsigned long) l_4(D);
_6 = _23 * 4;
_24 = pretmp_22 + _6;
_30 = (unsigned long) _24;
```

```
<bb 4>:
#sum_16 = PHI <0(3), sum_11(6)>
#ivtmp.9_20=PHI<ivtmp.9_18(3),ivtmp.9_19(6)>
.....
ivtmp.9_19 = ivtmp.9_20 + 4;
if (ivtmp.9_19 != _30)
  goto <bb 6>;
else
  goto <bb 5>;
```

ARM

Challenges – Conflict with LIM pass

- IVOPT depends on LIM to hoist invariant expr
- Strength reduced by outer loop's IVOPT
- Reduced invariants won't be moved
- Guide LIM with reg-pressure information
  - Model invariant epxrs as a forest of DAG
  - Model reg-pressure change by counting roots/leaves of the forest

```
.L12:
    add   x3, x23, x2
    ldr   d1, [x0]
    ldr   d16, [x3, x1]
    add   x3, x22, x2
    ldr   d17, [x0, x11, lsl 3]
    ldr   d7, [x0, x10, lsl 3]
    fmadd     d16, d16, d1, d0
    ldr   d0, [x3, x1]
    add   x3, x4, x2
    ldr   d3, [x3, x1]
    add   x3, x5, x2
    ldr   d6, [x0, x8, lsl 3]
    fmadd     d0, d0, d17, d16
    ldr   d1, [x3, x1]
    add   x3, x6, x2
    ldr   d4, [x0, x12, lsl 3]
    ldr   d2, [x0, x9, lsl 3]
    fmadd     d0, d3, d7, d0
    ldr   d18, [x0, x13, lsl 3]
    add   x0, x0, 8
    cmp   x0, x15
    fmadd     d6, d1, d6, d0
    ldr   d0, [x3, x1]
    add   x3, x7, x2
    fmadd     d4, d0, d4, d6
    ldr   d0, [x3, x1]
    add   x3, x24, x2
    ldr   d1, [x3, x1]
    add   x1, x1, x14
    fmadd     d0, d0, d2, d4
    fmadd     d0, d1, d18, d0
    bne   .L12
```


ARM

Challenges – Conflict with RTL loop unroller

- PR29256
  - Unroll on GIMPLE level
  - Change how RTL unroller handles iv
    - Don't split live range for iv
    - Factor [base+index] out of memory reference
    - Use [new_base + offset] addressing mode

```
<bb 3>:
 # ivtmp.10_8 = PHI <ivtmp.10_2(4), 0(2)>
 _4 = MEM[symbol: a, index: ivtmp.10_8, offset: 0B];
 MEM[symbol: c, index: ivtmp.10_8, offset: 0B] = _4;
 ivtmp.10_2 = ivtmp.10_8 + 8;
 if (ivtmp.10_2 != 16000000)
   goto <bb 3>;
 else
   goto <bb 5>;
```

```
.L2:
    add x3, x0, 8
    ldr x17, [x0, x1]
    add x9, x0, 16
    add x8, x0, 24
    add x7, x0, 32
    ldr x16, [x1, x3]
    add x6, x0, 40
    add x5, x0, 48
    add x4, x0, 56
    ldr x15, [x9, x1]
    str x17, [x0, x2]
    add x0, x0, 64
    ldr x14, [x8, x1]
    cmp x0, x18
    str x16, [x2, x3]
    ldr x13, [x7, x1]
    str x15, [x9, x2]
    ldr x12, [x6, x1]
    str x14, [x8, x2]
    ldr x11, [x5, x1]
    str x13, [x7, x2]
    ldr x10, [x4, x1]
    str x12, [x6, x2]
    str x11, [x5, x2]
    str x10, [x4, x2]
    bne .L2
```


ARM

Challenges – Compilation time

- IVOPT is expensive
- Bounded by different parameters

- Better algorithm and data structures

ARM

Thank You!