Initial work of SVE vectorization improvement

JDK-8183390: Fix and re-enable post loop vectorization

Pengfei Li @ Arm

Mar. 9, 2022

Agenda

- Background & status
- Our improvement plans
- Fix and re-enable post loop vectorization

Background & status

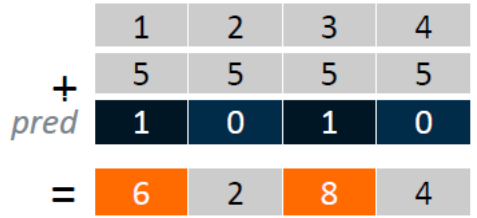
SVE & predicates

- **S**calable **V**ector **E**xtension
  - The latest SIMD instruction set in AArch64

- SVE predicates
  - SVE has 16 new predicate registers (p0 ~ p15) tracking vector lane activity

Per-lane predication

- Operations work on individual lanes under control of a predicate register




```
add     z0.s, p0/m, z1.s, z2.s
```

Predicate-driven loop control

- A predicate is generated by comparing loop's induction variable with its limit value to drive loop operations

```
for (i = 0; i < n; ++i)
```

INDEX i	n-2	n-1	n	n+1
CMPLT n	1	1	0	0

```
whilelt p0.s, x1, x2
```


arm

Scalar/NEON/SVE – a comparison

- Scalar

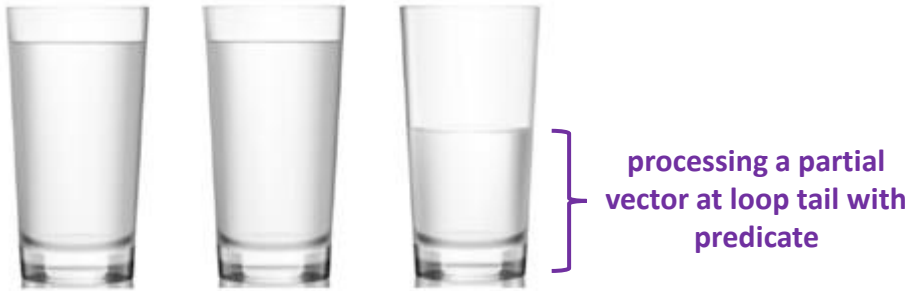
```
...
  ldr     w11, [x11, #16]
  ldr     w13, [x12, #16]
  add     w11, w11, w13
  str     w11, [x10, #16]
...
```

- NEON
  (no predicate)

```
...
  ldr     q16, [x11, #32]
  ldr     q17, [x12, #32]
  add     v16.4s, v16.4s, v17.4s
  str     q16, [x10, #32]
...
  ldr     w11, [x11, #16]
  ldr     w13, [x12, #16]
  add     w11, w11, w13
  str     w11, [x10, #16]
...
```

- SVE
  (predicate-driven)

```
...
  ld1w    {z16.s}, p0/z, [x1]
  ld1w    {z17.s}, p0/z, [x2]
  add     z16.s, z16.s, z17.s
  st1w    {z16.s}, p0, [x3]
...
```



SLP auto-vectorization

Auto-vectorization in C2

- C2 compiler has an existing auto-vectorizer, named **Superword** or **SLP**
- C2's SLP requires some pre-requisite loop transformations (iteration split & unrolling)

```
for (int i = start; i < limit; i++) {
  c[i] = a[i] + b[i];
}
```

iteration split

```
int i = start;
for (; i < mainStart; i++) {     // Pre-loop
  c[i] = a[i] + b[i];
}
for (; i < mainLimit; i++) {     // Main-loop
  c[i] = a[i] + b[i];
}
for (; i < limit; i++) {         // Post-loop
  c[i] = a[i] + b[i];
}
```

unrolling

```
int i = start;
for (; i < mainStart; i++) { ... }  // Pre-loop
for (; i < mainLimit; i += 4) {     // Main-loop
  c[i] = a[i] + b[i];
  c[i + 1] = a[i + 1] + b[i + 1];
  c[i + 2] = a[i + 2] + b[i + 2];
  c[i + 3] = a[i + 3] + b[i + 3];
}
for (; i < limit; i++) { ... }      // Post-loop
```

SLP

```
int i = start;
for (; i < mainStart; i++) { ... }  // Pre-loop
for (; i < mainLimit; i += 4) {     // Main-loop
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < limit; i++) { ... }      // Post-loop
```

some C2 loop transformations in pseudo-code

Current status

- With basic SVE support, C2 can generate SVE instructions for main-loop since JDK 16
- However, there is **no SVE predicate support** (scalar loops still exist!)
  - p7 – a pre-initialized all-true predicate – is used in generated SVE instructions

```
for (int i = 0; i < LENGTH; i++) {
  c[i] = a[i] + b[i];
}
```

```
ldr     w11, [x11, #16]
ldr     w13, [x12, #16]
add     w11, w11, w13
str     w11, [x10, #16]
...
ld1w    {z16.s}, p7/z, [x1]
ld1w    {z17.s}, p7/z, [x2]
add     z16.s, z16.s, z17.s
st1w    {z16.s}, p7, [x3]
...
ldr     w11, [x11, #16]
ldr     w13, [x12, #16]
add     w11, w11, w13
str     w11, [x10, #16]
```

scalar pre-loop

SVE vectorized main-loop

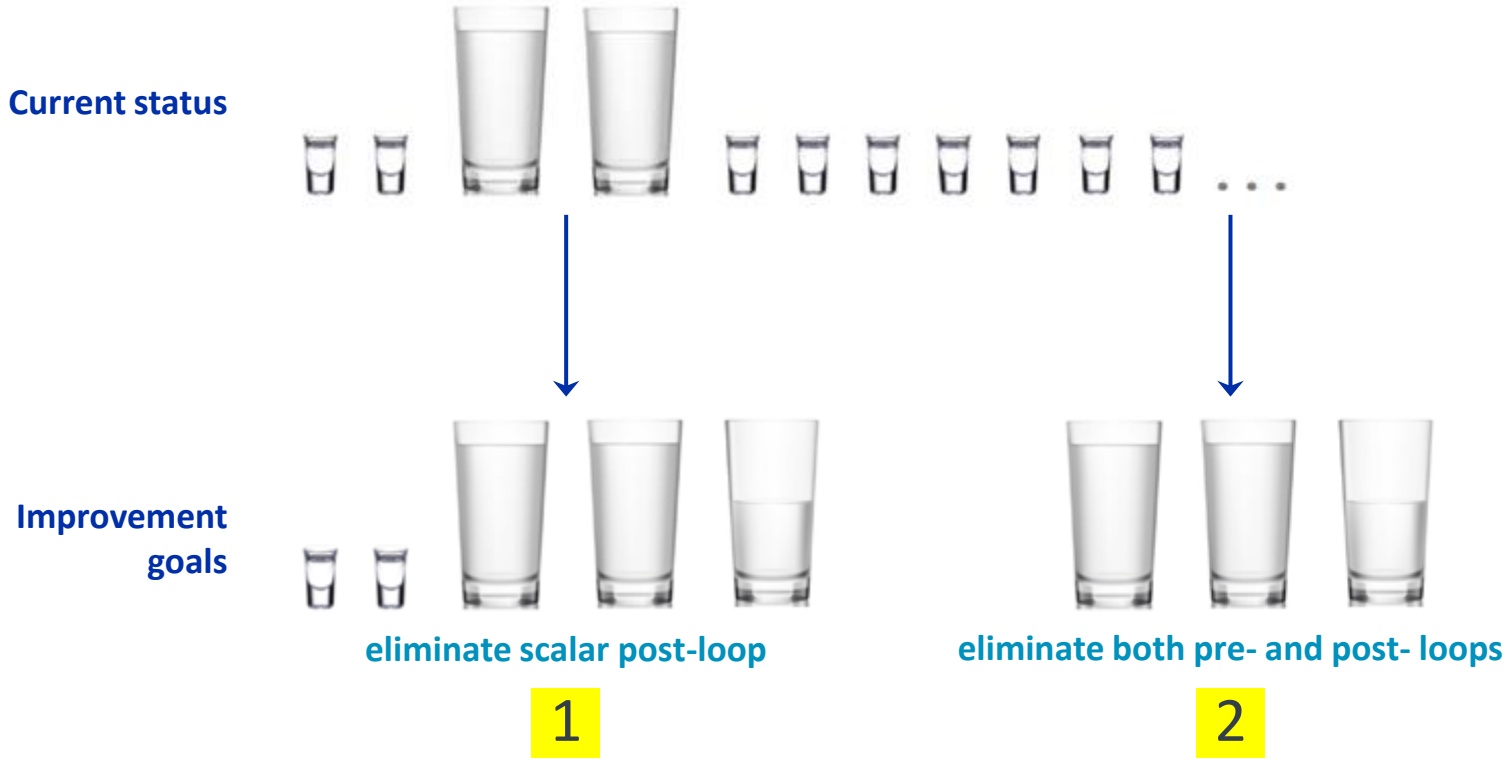
scalar post-loop

always using p7 – an “all-true” predicate

Our improvement plans

Goals

- Better auto-vectorization with SVE predicates
  - Eliminate scalar pre- and post- loops via enabling SVE predicates



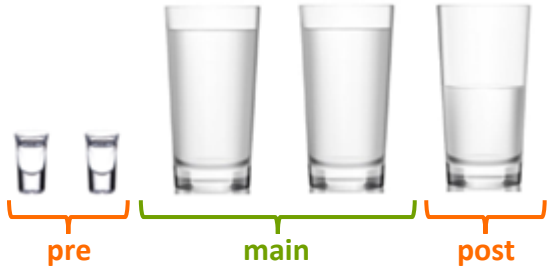
arm

Two solutions/plans

Short-term

- Vectorize only post-loop with vector masks
- Still SLP-based optimization
- Large generated code size
- Small effort

```
int i = start;
for (; i < mainStart; i++) { ... }  // Pre-loop
for (; i < mainLimit; i += 4) {     // Main-loop
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < limit; incr(&i)) {       // Post-loop
  c[i:i+3] = vector_masked(a[i:i+3] + b[i:i+3]);
}
```



Long-term

- Transform full loop iterations
- Implement a new predicate-driven vectorizer
- Small generated code size
- Significant effort

```
for (int i = start; i < limit; incr(&i)) {
  c[i:i+3] = vector_masked(a[i:i+3] + b[i:i+3]);
}
```

Fix and re-enable post loop vectorization

Post loop vectorization

- An existing C2 optimization in experimental VM feature PostLoopMultiversioning
  - Transform one **range-check eliminated post loop** to a **1-iteration vectorized loop** with vector mask
  - History
    - Contributed by Intel in 2016 to support x86 AVX-512 masked vector instructions
    - Due to insufficient maintenance, multiple issues were accumulated inside
  - C2's post loops
    - Usually, C2 generates **2** post loops, one **scalar post loop** and one **vector drain loop**
    - With PostLoopMultiversioning, **3** post loops exist and vector mask applies to the additional one

C2's post loops

W/o PostLoopMultiversioning

- **Scalar post loop** and **vector drain loop**

iteration split, unrolling & SLP

```
int i = start;
for (; i < mainStart; i++) { ... }     // Pre
for (; i < mainLimit; i += 4) {        // Main
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < limit; i++) { ... }         // Scalar post
```

vector loop cloning

```
int i = start;
for (; i < mainStart; i++) { ... }     // Pre
for (; i < newMainLimit; i += 4) {     // Main
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < vPostLimit; i += 4) {       // Vector drain
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < limit; i++) { ... }         // Scalar post
```

super unrolling

```
int i = start;
for (; i < mainStart; i++) { ... }     // Pre
for (; i < newMainLimit; i += 16) {    // Main
  c[i:i+3] = a[i:i+3] + b[i:i+3];
  c[i+4:i+7] = a[i+4:i+7] + b[i+4:i+7];
  c[i+8:i+11] = a[i+8:i+11] + b[i+8:i+11];
  c[i+12:i+15] = a[i+12:i+15] + b[i+12:i+15];
}
for (; i < vPostLimit; i += 4) {       // Vector drain
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < limit; i++) { ... }         // Scalar post
```

This vector drain loop avoids the max iteration count of the following scalar post loop being too large

C2's post loops (multiversioned)

W/ PostLoopMultiversioning

- One more range-check eliminated (RCE'd) post loop is created with specific loop limit value and SLP vectorizes that if it's considered to be vectorizable

```
int i = start;
for (; i < mainStart; i++) { ... }    // Pre
for (; i < newMainLimit; i += 16) {   // Main
  c[i:i+3] = a[i:i+3] + b[i:i+3];
  c[i+4:i+7] = a[i+4:i+7] + b[i+4:i+7];
  c[i+8:i+11] = a[i+8:i+11] + b[i+8:i+11];
  c[i+12:i+15] = a[i+12:i+15] + b[i+12:i+15];
}
for (; i < vPostLimit; i += 4) {      // Vector drain
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < minOfRanges; i++) {        // RCE'd
  c[i] = a[i] + b[i];
}
for (; i < limit; i++) {              // Scalar post
  c[i] = a[i] + b[i];
}
```

SLP (post loop)

```
int i = start;
for (; i < mainStart; i++) { ... }    // Pre
for (; i < newMainLimit; i += 16) {   // Main
  c[i:i+3] = a[i:i+3] + b[i:i+3];
  c[i+4:i+7] = a[i+4:i+7] + b[i+4:i+7];
  c[i+8:i+11] = a[i+8:i+11] + b[i+8:i+11];
  c[i+12:i+15] = a[i+12:i+15] + b[i+12:i+15];
}
for (; i < vPostLimit; i += 4) {      // Vector-drain
  c[i:i+3] = a[i:i+3] + b[i:i+3];
}
for (; i < minOfRanges; i += 4) {     // RCE'd
  c[i:i+3] = opmasked(a[i:i+3] + b[i:i+3]);
}
for (; i < limit; i++) {              // Scalar post
  c[i] = a[i] + b[i];
}
```

With the vector drain loop before, this runs at most 1 iteration

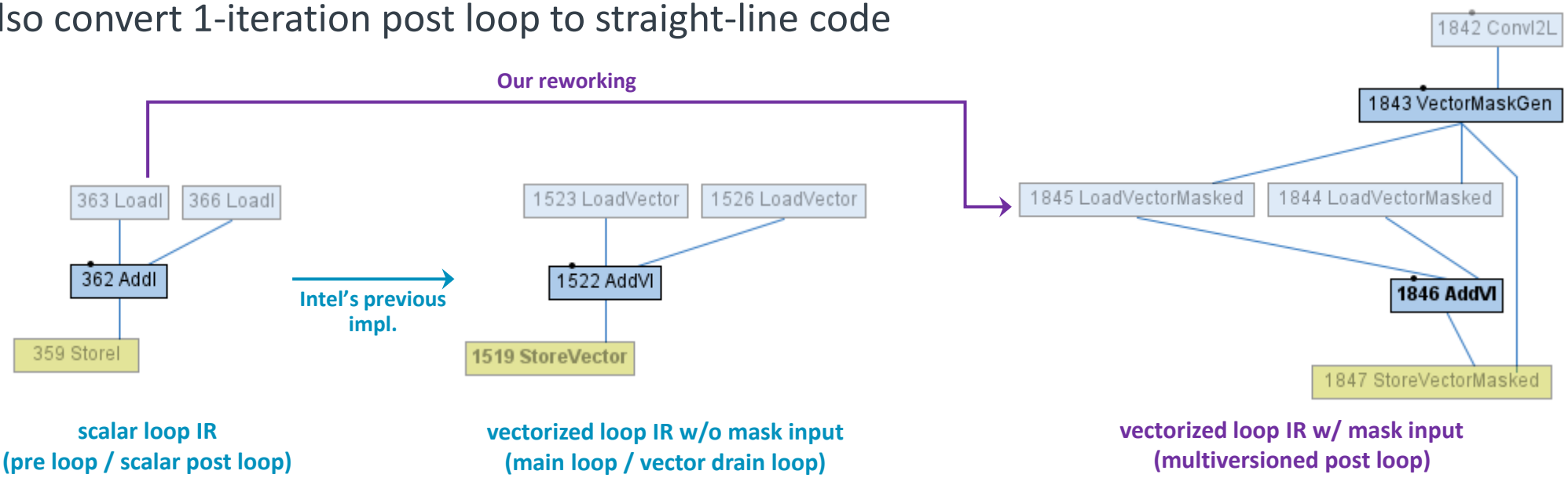
Intel’s previous implementation

- Setting up vector mask in x86 backend
  - Hard-code x86 **k1** register as a reserved AVX-512 opmask register
  - Two routines (setvectmask/restorevectmask) to set and restore the value of k1
  - Directly promote all scalar operations in the loop body to vector operations

- Problem
  - It only works on x86
  - All AVX-512 vector instructions should be encoded with k1 by default
    - k1 needs to be initialized to “all-true”
    - That’s why post loop vectorization is completely broken after JDK-8211251 which encodes AVX-512 instructions as unmasked by default
  - Vector masks only apply to **int** vectors
  - Some incorrect result issues

Our reworking

- Turn to adding vector mask input to C2 mid-end IRs
  - Use a **`VectorMaskGenNode`** to generate a mask
  - Replace all Load/Store nodes in the post loop into **`LoadVectorMasked`**/**`StoreVectorMasked`** nodes with that mask input (it's enough as loops with reductions are not supported)
  - In this way we can enable post loop vectorization for platforms other than x86
  - We also convert 1-iteration post loop to straight-line code



Our further fixups

- Crash fix for strip-mined loops
  - A SIGSEGV issue caused by not taking strip-mined loop into consideration

- Several incorrect result issues fix
  - For partial vectorizable loops (a)
  - For loops with growing-down vectors (b)
  - For manually unrolled loops (c)
  - For loops with mixed vector element sizes (d)
  - For loops with potential data dependence (e)

```
for (int i = 0; i < 10000; i++) {
    c[i] = a[i] * b[i];
    k = 3 * k + 1;
}
```

(a)

```
for (int i = 0; i < 10000; i++) {
    a[MAX - i] = b[MAX - i];
}
```

(b)

```
for (int i = 0; i < 10000; i += 2) {
    c[i] = a[i] + b[i];
    c[i + 1] = a[i + 1] * b[i + 1];
}
```

(c)

```
for (int i = 0; i < 10000; i++) {
    ic[i] = ia[i] + ib[i];  // int
    lc[i] = la[i] + lb[i];  // long
}
```

(d)

```
for (int i = 0; i < 10000; i++) {
    a[i] = x;
    a[i + OFFSET] = y;
}
```

(e)

  - See commit message of the patch for details

Links

- JBS: https://bugs.openjdk.java.net/browse/JDK-8183390
- PR: https://github.com/openjdk/jdk/pull/6828

- Any review comments are welcome!

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה