Loop Vectorization:
How to vectorize interleave memory access?

Hao Liu, James Molloy and Jiangning Liu

14th April 2015

ARM®

Background: Interleave Access

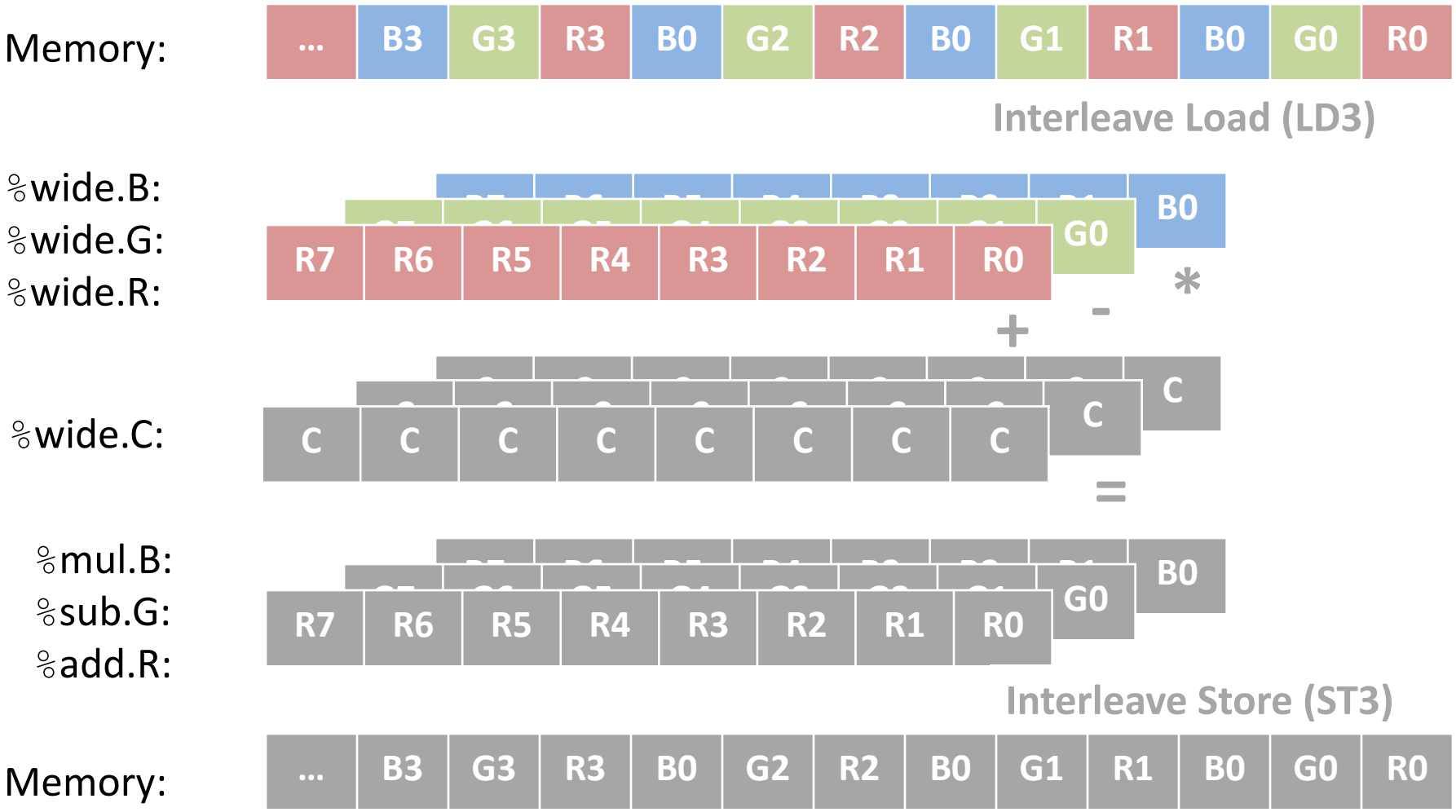
- Case: visit 24-bit RGB image

Memory:


```
for (i = 0; i < N; i += 3) {
    R = RGB[i];
    G = RGB[i+1];
    B = RGB[i+2];
    R += C;
    G -= C;
    B *= C;
    RGB[i] = R;
    RGB[i + 1] = G;
    RGB[i + 2] = B;
}
```

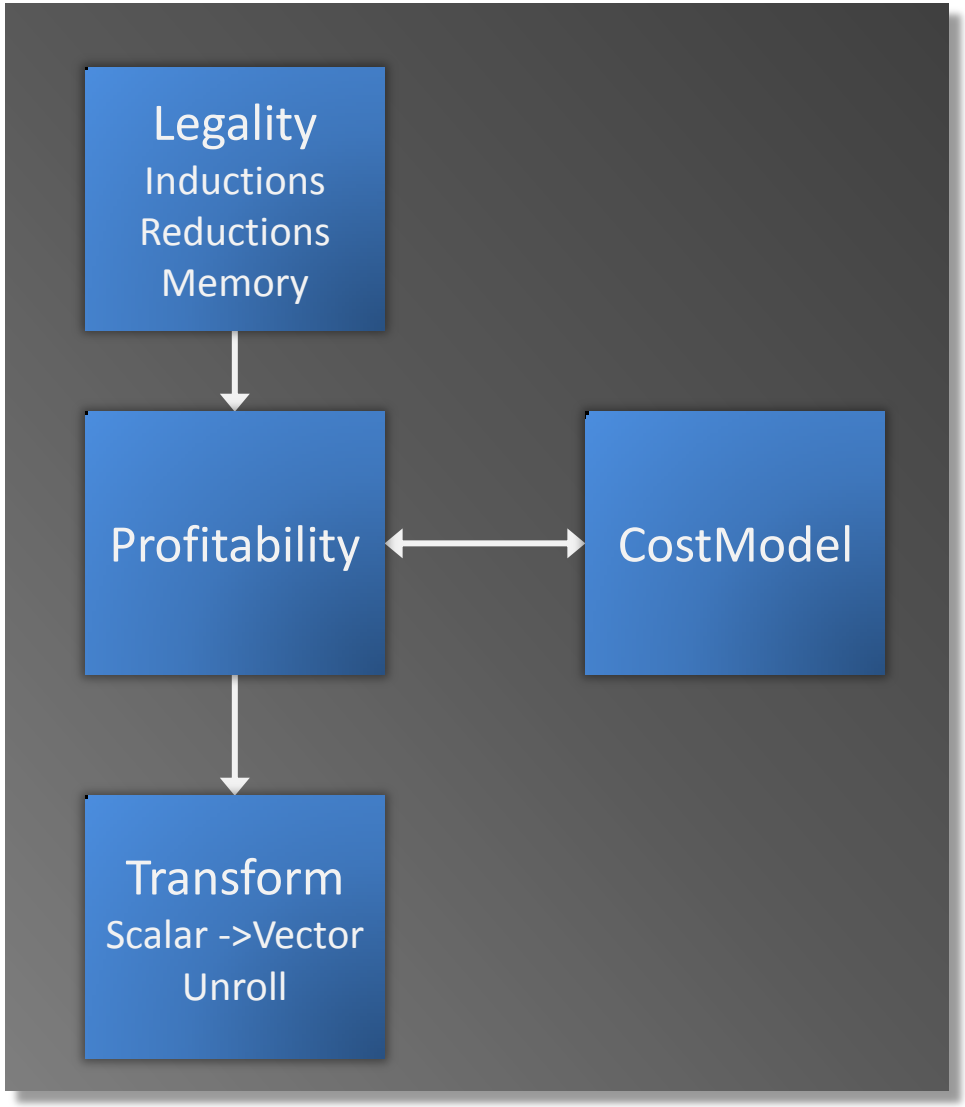
```
for.body:
  ...
  %R = load i8, i8* %idx0
  %G = load i8, i8* %idx1
  %B = load i8, i8* %idx2
  %add = add i8 %R, %C
  %sub = sub i8 %G, %C
  %mul = mul i8 %B, %C
  store i8 %add, i8* %idx0
  store i8 %sub, i8* %idx1
  store i8 %mul, i8* %idx2
  ...
```

Background: Interleave Access



Loop Vectorizer Overview

- 3 phases:
  - Legality
  - Profitability
  - Transform



Teach Loop Vectorizer: Legality

- Identification
  - Collect: Constant strided accesses
  - Sort: Consecutive accesses the same stride
  - Select: Number of accesses equal to the stride

Step1: `StrideList = {<%R, 3>, <%G, 3>, <%B, 3>, ...}`

Step2: `ConsecutiveList = {%R, %G, %B, ...}`

Step3: `InterleaveList = {%R, %G, %B}`

Teach Loop Vectorizer: Legality

- Induction with arbitrary steps (Patch upstreamed)

```
for (unsigned i = 0; i < N; i += 3 ) {
    ...
```

- Memory check

```
for (i = 0; i < N; i += ?) {
    R = RGB[i];
    G = RGB[i+1];
    B = RGB[i+2];
    ...
    RGB[i] = R;
    RGB[i + 1] = G;
    RGB[i + 2] = B;
}
```

```
True Dependence:
  i+=1, i+=2

No Dependence:
  i+=3
```

Teach Loop Vectorizer: Transform

- IRs to intrinsics

```
%R = load i8, i8* %ptr0
%G = load i8, i8* %ptr1
%B = load i8, i8* %ptr2
```

Loop Vectorizer

```
<8 x i8> stride.load(%ptr0, 0, 3)
<8 x i8> stride.load(%ptr0, 1, 3)
<8 x i8> stride.load(%ptr0, 2, 3)
```

```
<24 x i8> index.load(%ptr0, <0,3,6,…,1,...
<8 x i8> shuffle <0,1,2,3,4,5,6,7>
<8 x i8> shuffle <8,9,10,11,12,13,14,15>
<8 x i8> shuffle <16,17,18,19,20,21,22,23>
```

Back End

```
call {<8xi8>, <8xi8>, <8xi8>} llvm.aarch64.ld3(%ptr)
```

Expect Performance Gain

- Expected improvements in specific benchmarks
  - EEMBC.rgbcmy 6x
  - EEMBC.rgbyiq 3x
- Need more testing and tuning
- More Challenges
  - Runtime memory dependence checks
  - Type promotion: i8 is illegal but <8 x i8> is legal

Thank you!