Active Lane Consolidation

Rouzbeh Paktinatkeleshteri, M.Sc.
João P. L. de Carvalho, PhD
José Nelson Amaral, PhD

UNIVERSITY OF ALBERTA

SVE

- Vector-Length-Agnostic Extension For Arm Processors
  - Supports variety of vector lengths
  - From 128 bits to 2048 bits
- Powerful Vector Predication support
  - Previous vector extensions (e.g NEON, AVX) have simple predication
  - Supports Per Vector Predication
- Special vector manipulation Instructions
  - Used for moving data between vectors
  - Very fast!

Loop Vectorization Challenges

If Conversion

```
For ( int i = 0 ; i < n ; ++i ) {
        if( cond[i]) {
                A[i] = B[i] * C[i];
        }else{
                A[i] = B[i] + C[i];
        }
}
```

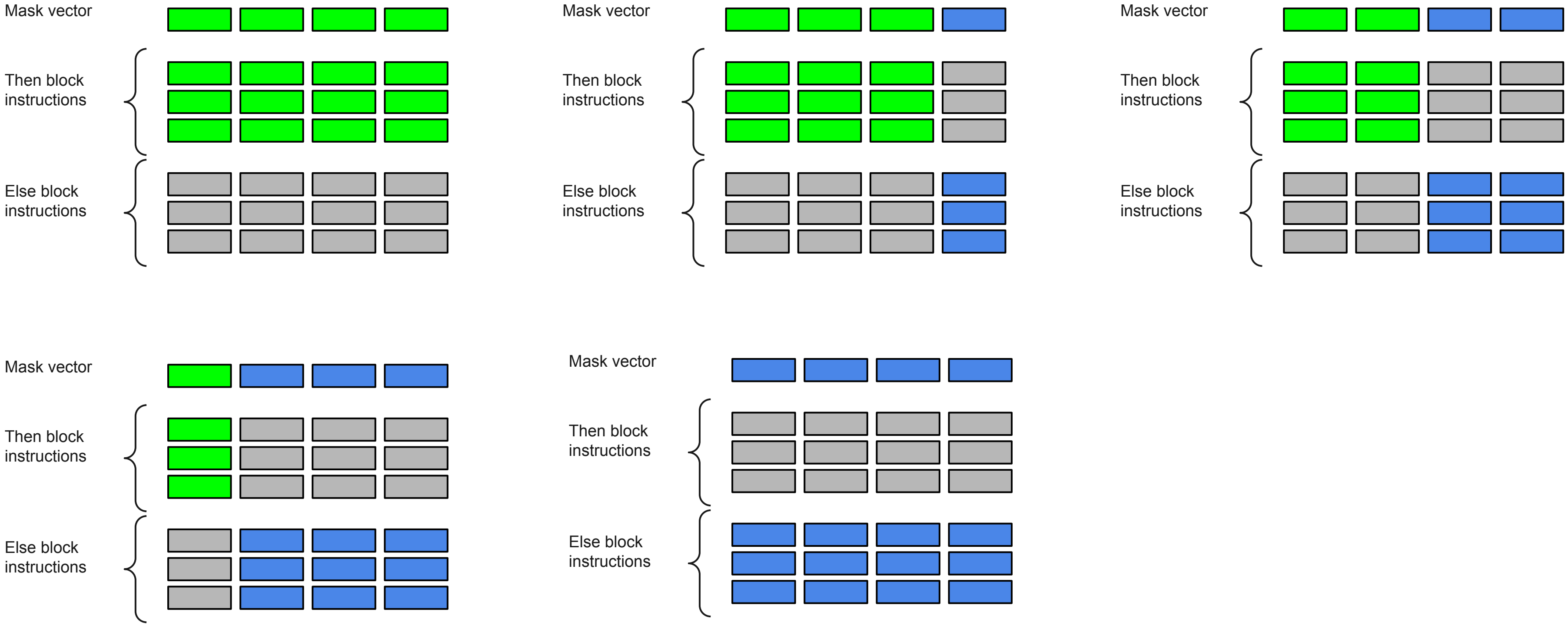
```
For ( int i = 0 ; i < n ; i+=4 ) {
        mask = cond[i:i+4]
        store_p( mask, &A[i], B[i:i+4] * C[i:i+4] );
        store_p( !mask, &A[i], B[i:i+4] + C[i:i+4] );
}
```

What is the Problem?

```
For ( int i = 0 ; i < n ; i+=4 ) {
        mask = cond[i:i+4]
        store_p( mask, &A[i], B[i:i+4] * C[i:i+4] );
        store_p( !mask, &A[i], B[i:i+4] + C[i:i+4] );
}
```

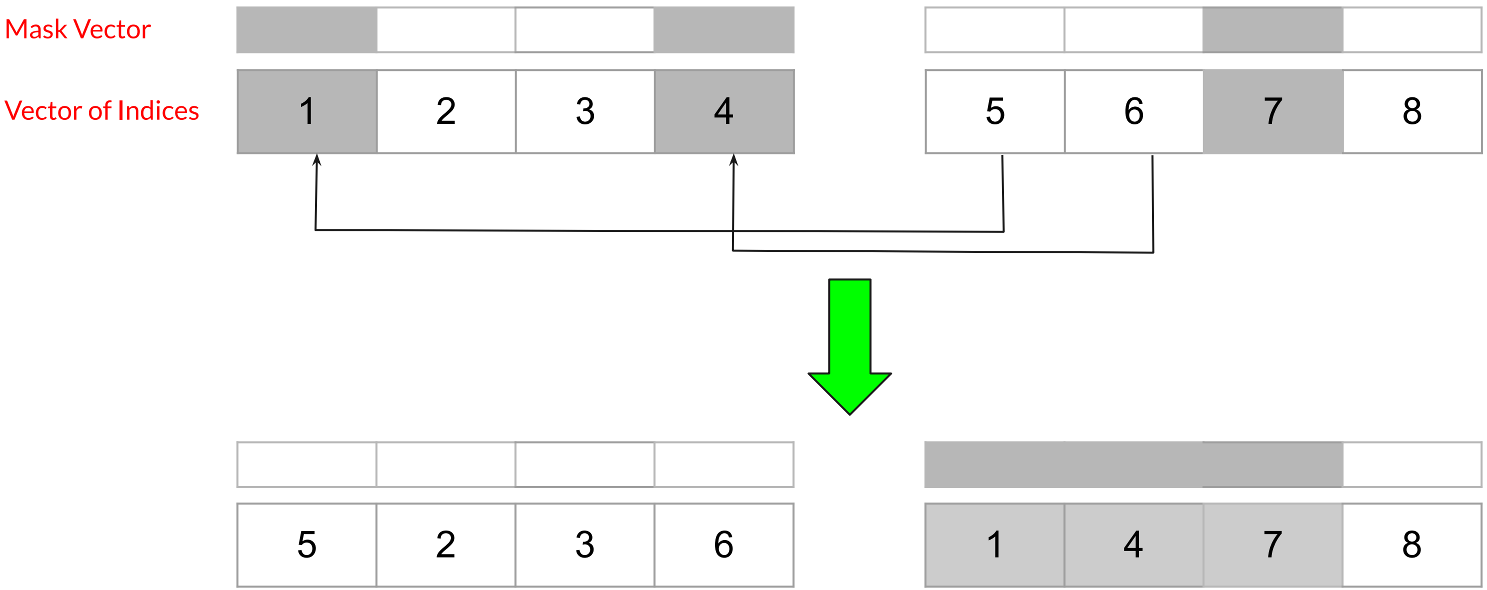
Only one of them will be committed!!

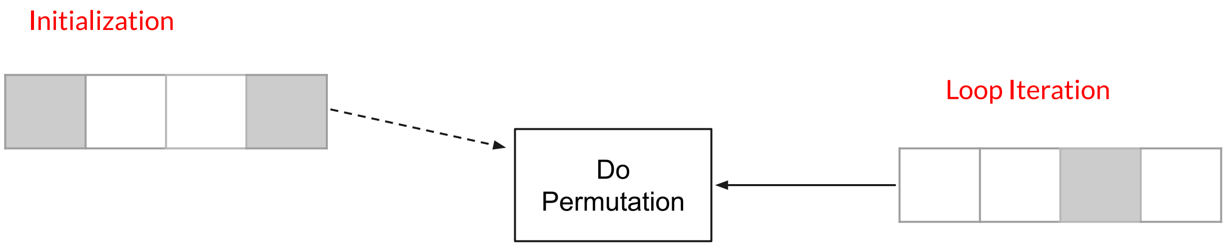
The other one is wasted!!

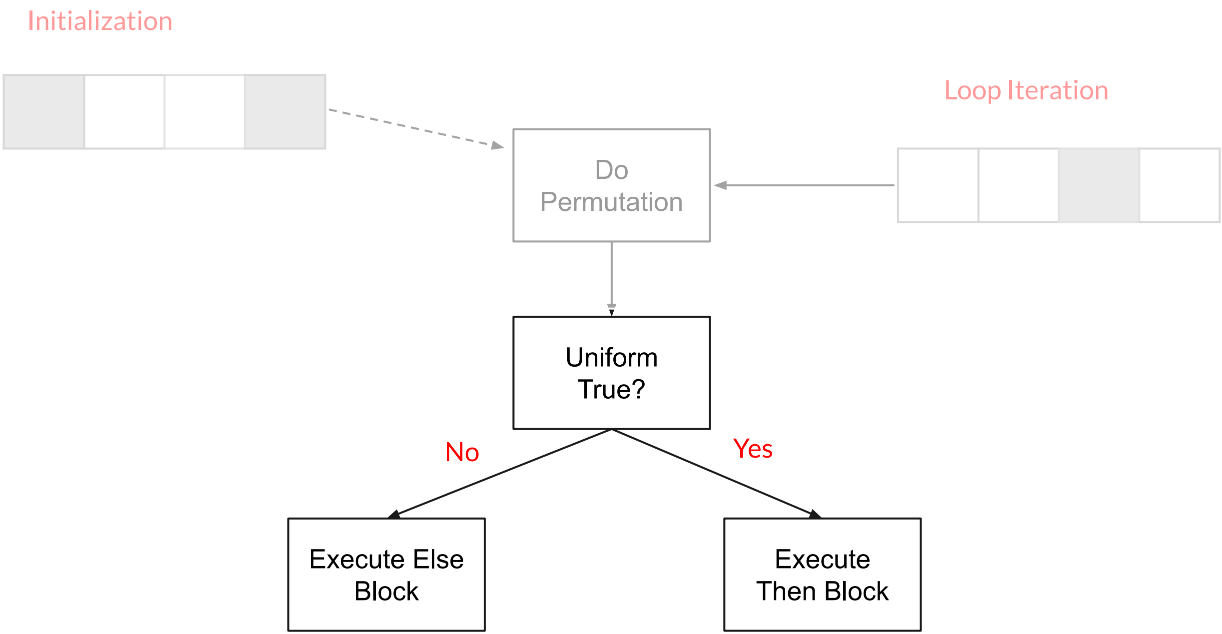

Mask vector
Then block instructions
Else block instructions
Mask vector
Then block instructions
Else block instructions
Mask vector
Then block instructions
Else block instructions
Mask vector
Then block instructions
Else block instructions
Mask vector
Then block instructions
Else block instructions

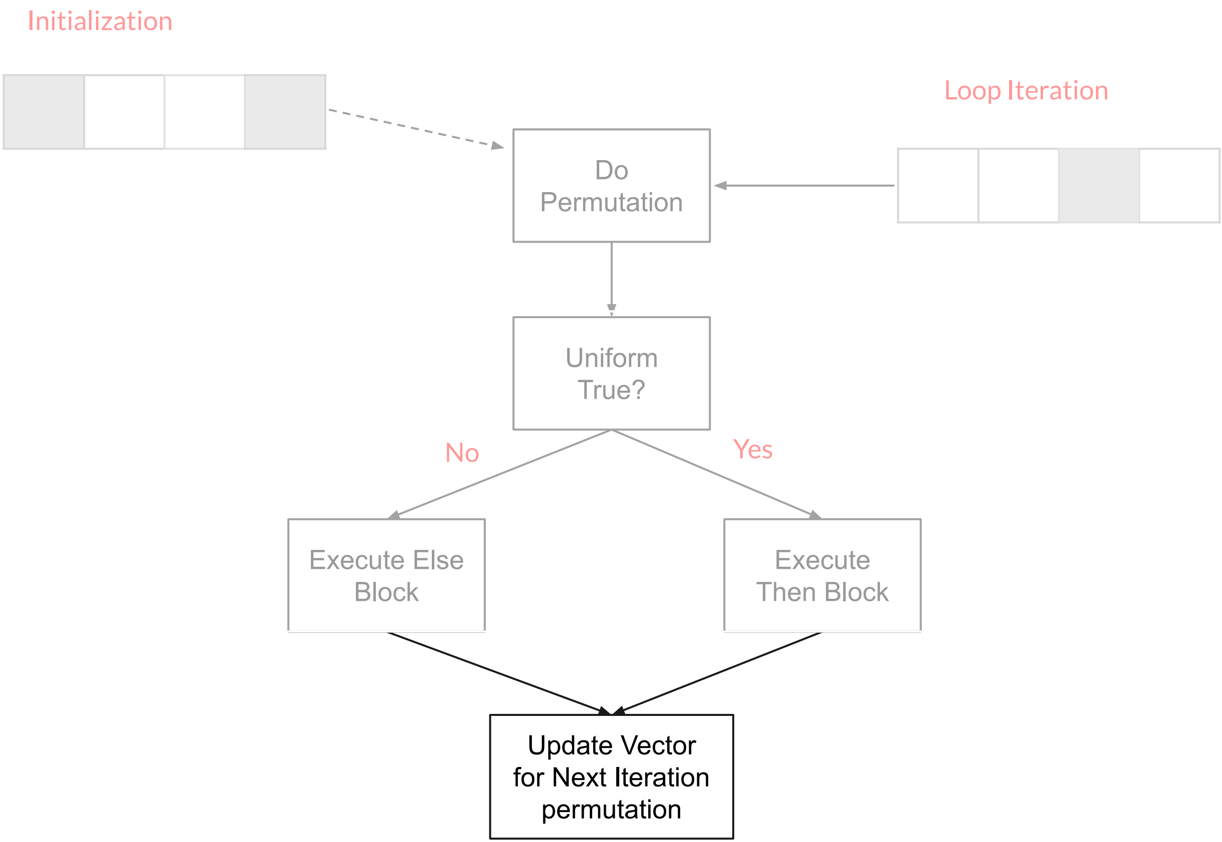
Active Lane Consolidation (ALC)

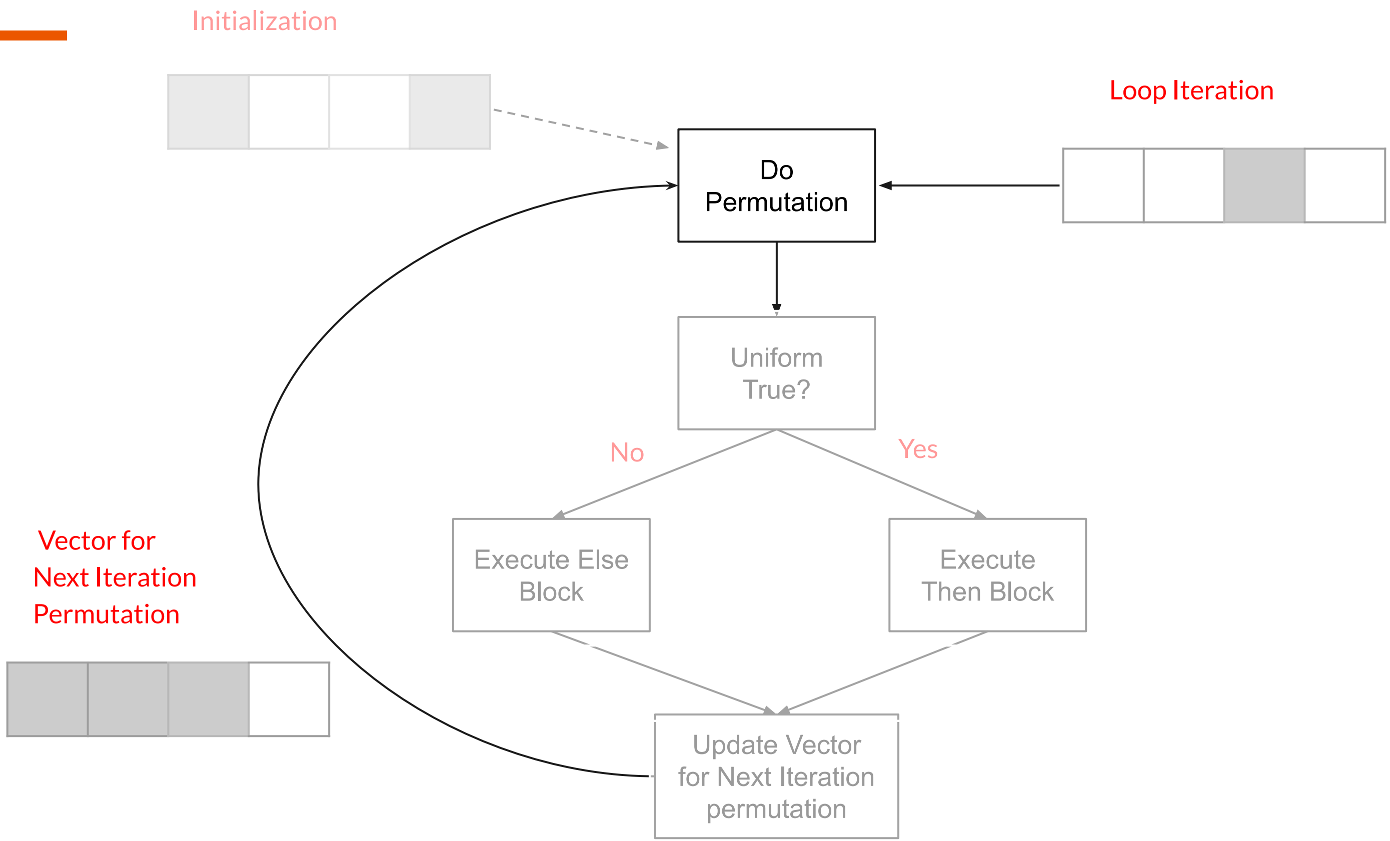
Permutation: Gathering True elements




Initialization
Do
Permutation
Loop Iteration


Initialization
Loop Iteration
Do Permutation
Uniform True?
No
Yes
Execute Else Block
Execute Then Block


Initialization
Loop Iteration
Do Permutation
Uniform True?
No
Yes
Execute Else Block
Execute Then Block
Update Vector for Next Iteration permutation



Initialization
Loop Iteration
Do Permutation
Uniform True?
No
Yes
Execute Else Block
Execute Then Block
Update Vector for Next Iteration permutation
Vector for Next Iteration Permutation

How does it perform?

Experimental Setup

Machine:

- Fujitsu’s A64FX 4 nodes x 12 threads (48 threads) – 32GB RAM
- VL = 512-bits

Compiler: Arm’s Clang

Test Kernel

```
for (int i = 0; i < n; ++i) {
        if (cond[i]) {



        } else {



        }
}
```

Test Kernel

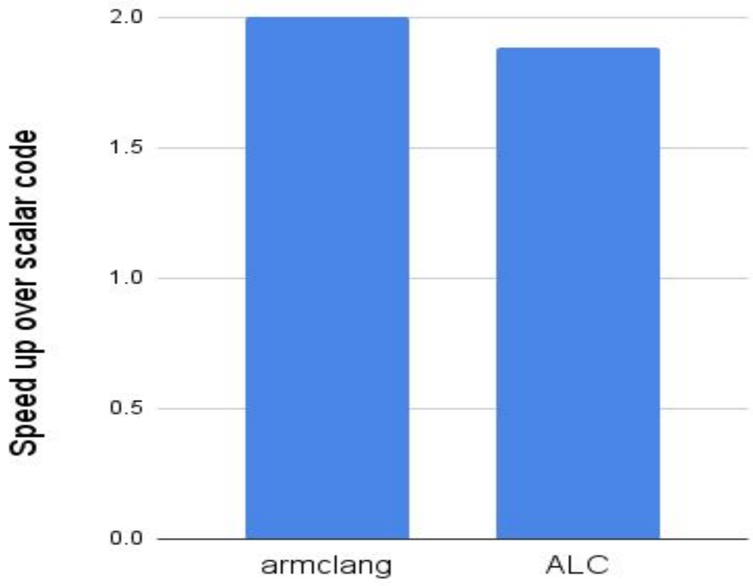
```
for (int i = 0; i < n; ++i) {

        if (cond[i]) {
                a[i] = (2 * a[i] - 2 * c[i]) + (b[i] - 2 * a[i]);
                a[i] += 2 * i + i * b[i];
                b[i] = 2 - 2 * b[i] + (2 * a[i] - 2 * c[i]);
                b[i] -= 3 * i + i * c[i];
                c[i] += 2 * b[i] + 2 * a[i] - 3 * (2 * c[i] - 2 * b[i] + i * i);
        } else {
                a[i] *= 2 + b[i] - 3 * c[i];
                c[i] = a[i] * b[i] - 1 + c[i];
                b[i] = 3 * a[i] - 2 * c[i];
                b[i] -= 2 * c[i] + 7 + a[i] ;
                a[i] -= 4 + b[i] * 2;
                c[i] += 5 * a[i] + 2 * b[i];
        }
}
```

Speedup Over Scalar Code

- 1.88X faster than scalar
- Still slower than Armclang vectorization!!

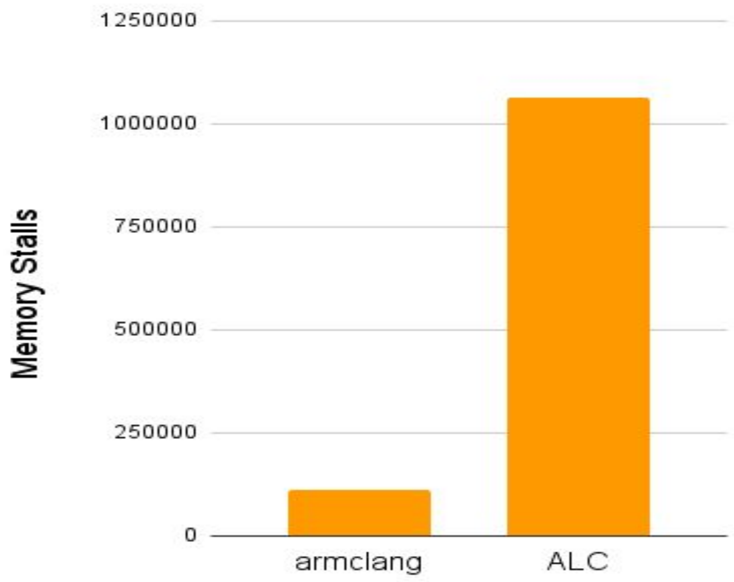
But What's Wrong with ALC?



ALC Bottleneck

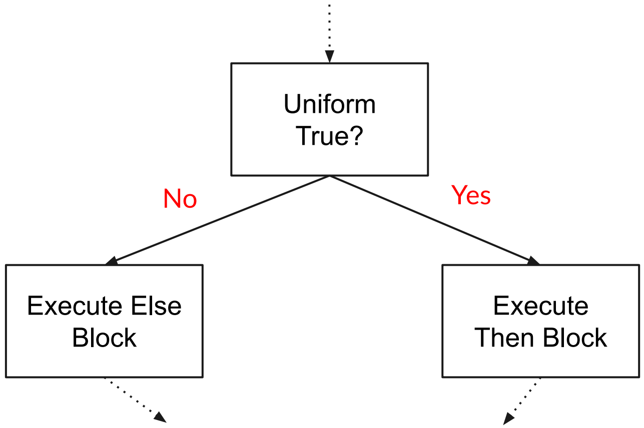
- Permutation overhead?
  - SVE vector manipulation instructions are so fast
  - Takes less than 5% of execution time
- Measure more metrics

~10X more stalls due to memory!!!



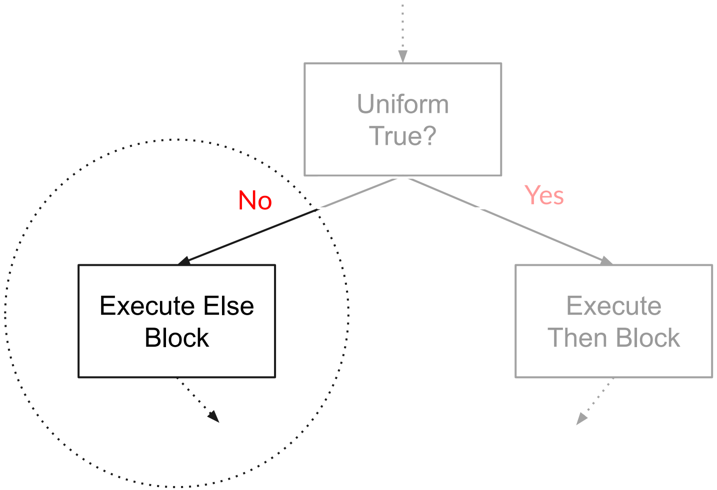
Why Memory Stalls?

- Problem happens in uniform blocks



Why Memory Stalls?

- Problem happens in uniform blocks

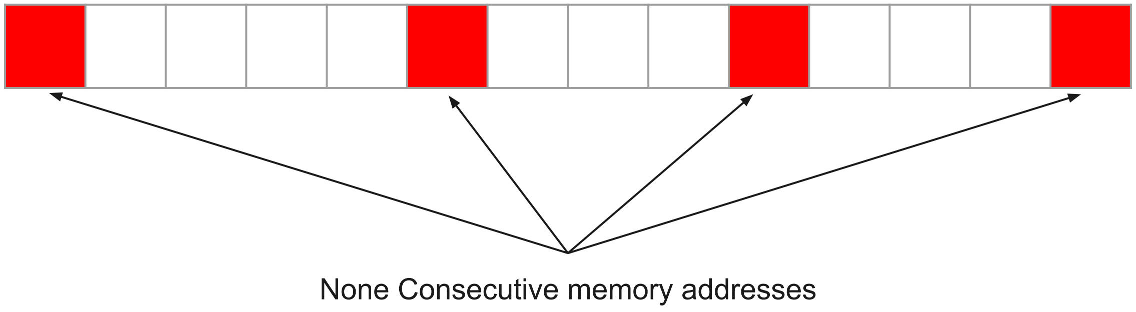


- Need to load following indices of array B and C
- Gather load instruction are used

Uniform vector to execute else block

0	5	9	13

```
For ( int i = 0 ; i < n ; ++i ) {
        if( cond[i]) {
                A[i] = B[i] * C[i];
        }else{
                A[i] = B[i] + C[i];
        }
}
```



Causing High Latency!!

Solution?

- Need to eliminate gather instructions.
- Want to do regular vector loads from consecutive memory addresses

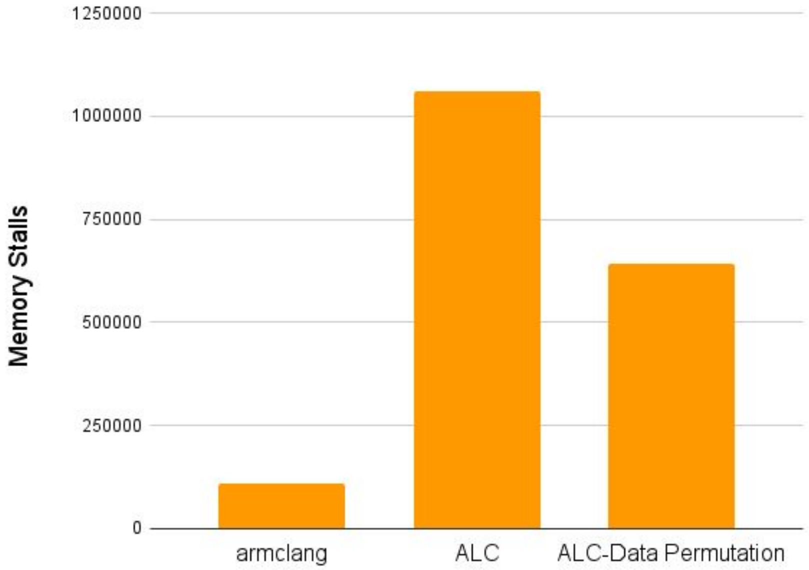
→

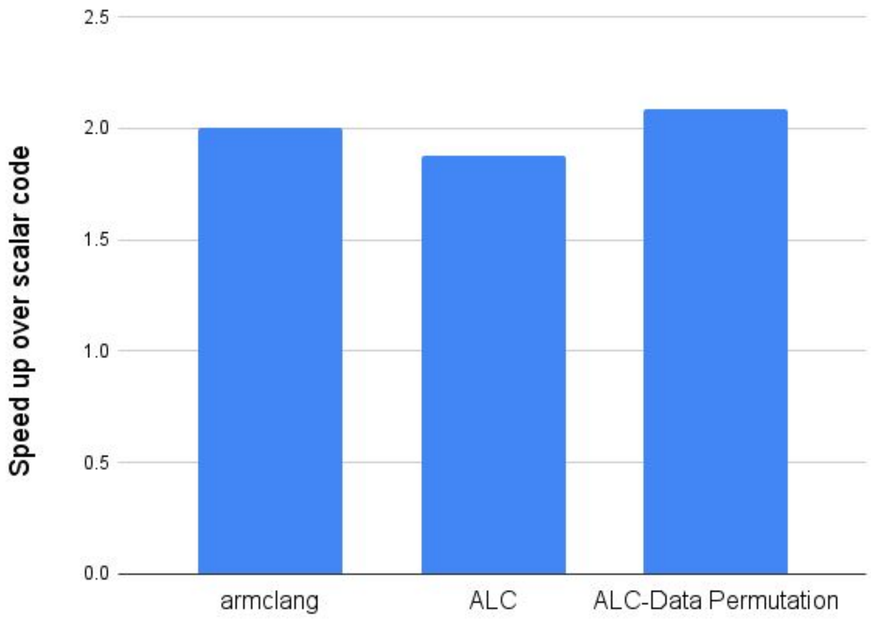
Data Permutation

- Load all indices of the array
- Permute them in each iteration

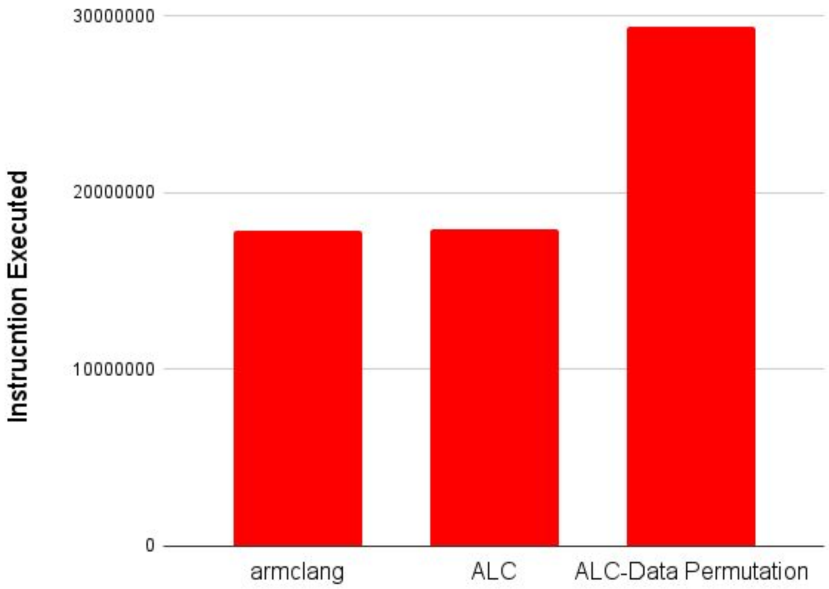
Memory Stalls

- Reduced stalls by 40%
- Still much more than armclang
- Scatter stores should also be eliminated





Speedup over scalar code



Number of Instructions executed

- Executing 69% more Instructions
- Still ~9% more speedup over previous version
- More improvement by eliminating Scatter Store

Gather/Scatter Instructions are BAD!!!

Thank You