GENERATING EFFICIENT CPU CODE WITH MLIR

Scalable Vector Extensions in an end-to-end case study

Andrzej Warzyński
andrzej.warzynski@arm.com

Ege Beysel
beysel@roofline.ai

arm

roofline

DATA-TILED MATRIX MULTIPLICATIONS ARE IMPLEMENTED ON LINALG-LEVEL IN MLIR

Matmul in MLIR

```
%matmul = linalg.matmul
  ins(%lhs, %rhs : tensor<4x4xf32>,
  tensor<4x4xf32>)
  outs(%acc : tensor<4x4xf32>) ->
  tensor<4x4xf32>
```

Data tiling

Data-tiled Matmul in MLIR

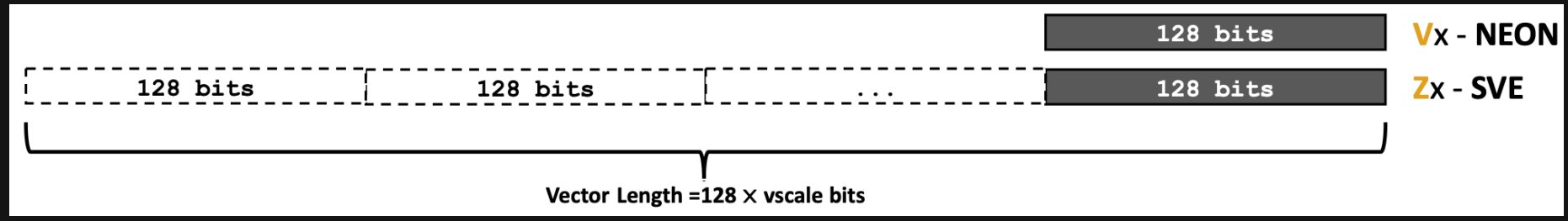
```
%packed_lhs = linalg.pack %lhs ...
  inner_tiles = [2, 2] into %lhs_dest :
  tensor<4x4xf32> -> tensor<2x2x2x2xf32>
%packed_rhs = linalg.pack %rhs ...
  inner_tiles = [2, 2] into %rhs_dest :
  tensor<4x4xf32> -> tensor<2x2x2x2xf32>
%packed_acc = linalg.pack %acc ...
  inner_tiles = [2, 2] into %acc_dest:
  tensor<4x4xf32> -> tensor<2x2x2x2xf32>

%mmt4d = linalg.mmt4d ins(%packed_lhs, %packed_rhs
  : tensor<2x2x2x2xf32>, tensor<2x2x2x2xf32>)
  outs(%acc_dt : tensor<2x2x2x2xf32>)

%unpack = linalg.unpack %mmt4d
  inner_tiles = [2, 2] into %acc :
  tensor<2x2x2x2xf32> -> tensor<4x4xf32>
```

- Pack operands in a SIMD, cache-friendly layout
- Layouts dependent on target hardware
- Pack and unpack operations can be fused into their producers and consumers

ARM'S SCALABLE VECTOR EXTENSION (SVE) ENABLES DIFFERENT VECTOR LENGTHS DECIDED BY HARDWARE IMPLEMENTATION

32 scalable vector registers (Z0-Z31):

- 128-2048 bits vector length is decided by implementation
- Always vscale * base size (128 bits)



ISA designed for Vector Length Agnostic (VLA) programming:

- VL (vector length) is unknown at compile-time, but known at run-time
- 16 scalable predicate registers to facilitate this (p0-p15)

```
void foo(int *out, int *a, int *b, int N)
{
    for (int i=0; i<N; i++)
        out[i] = a[i] + b[i];
}
```

clang -O2 -march=armv8a+sve

```
.L_loopStart:
ld1w z1.s, p1/Z, [x1, x9, LSL #2]
ld1w z2.s, p1/Z, [x2, x9, LSL #2]
add z1.s, p1/M, z1.s, z2.s
st1w z1.s, p1, [x0, x9, LSL #2]
incw x9
whilelt p1.s, x9, x3
b.first .L_loopStart
```

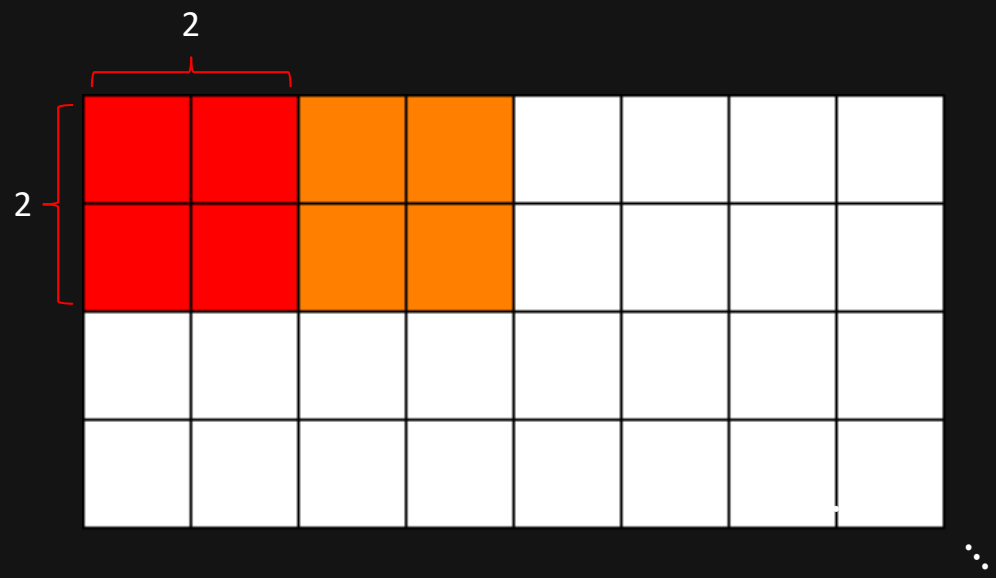
Increment by vscale!

ARM‘S FIXED VECTOR LENGTH EXTENSION, NEON, HAS STATIC DATA-TILED LAYOUTS

NEON overview

```
%lhs_dt = linalg.pack %lhs ...
  inner_tiles = [2, 2] into %lhs_dest :
  tensor<256x256xf32> -> tensor<128x128x2x2xf32>
%rhs_dt = linalg.pack %rhs ...
  inner_tiles = [2, 2] into %rhs_dest :
  tensor<256x128xf32> -> tensor<64x128x2x2xf32>
%acc_dt = linalg.pack %acc ...
  inner_tiles = [2, 2] into %acc_dest:
  tensor<256x128xf32> -> tensor<128x64x2x2xf32>

%mmt4d = linalg.mmt4d ins(%lhs_dt, %rhs_dt :
  tensor<128x128x2x2xf32>, tensor<64x128x2x2xf32>)
  outs(%acc_dt : tensor<128x64x2x2xf32>)

%unpack = linalg.unpack %mmt4d ...
  inner_tiles = [2, 2] into %acc :
  tensor<128x64x2x2xf32> -> tensor<256x128xf32>
```

Static data-tiled layouts



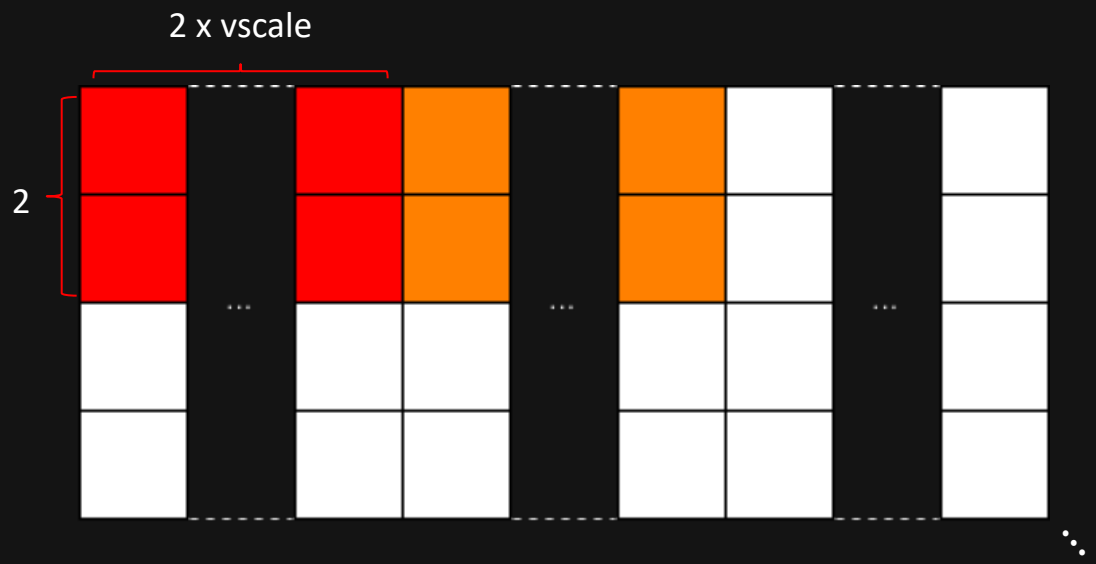
WHEN USING SVE INSTEAD OF NEON, DATA-TILED LAYOUTS BECOME SCALABLE

SVE overview

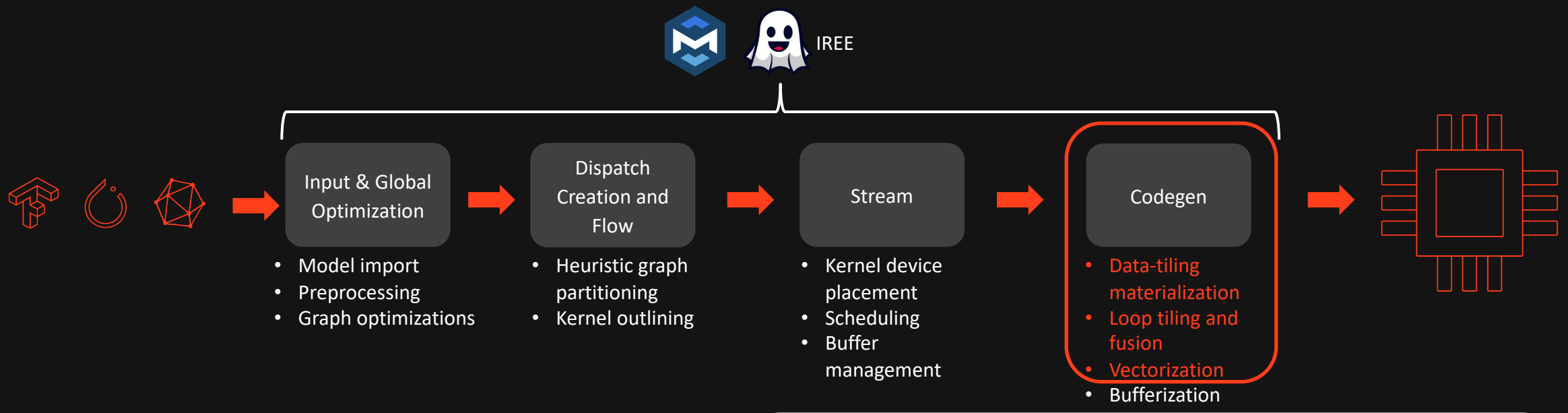
```
%vscale = vector.vscale
%c2_vscale = arith.muli %c2, %vscale : index
%packed_lhs = linalg.pack %lhs ... tensor<256x256xf32> ->
  tensor<128x128x2x2xf32>
%packed_rhs = linalg.pack %rhs ...
  inner_tiles = [%c2_vscale, 2] into %rhs_dest :
  tensor<256x128xf32> -> tensor<?x128x?x2xf32>
%packed_acc = linalg.pack %acc
  inner_tiles = [2, %c2_vscale] into %acc_dest:
  tensor<256x128xf32> -> tensor<128x?x2x?xf32>

%mmt4d = linalg.mmt4d ins(%packed_lhs, %packed_rhs :
  tensor<128x128x2x2xf32>, tensor<?x128x?x2xf32>)
  outs(%packed_acc : tensor<128x?x2x?xf32>)

%unpack = linalg.unpack %mmt4d ...
  inner_tiles = [2, %c2_vscale] into %acc :
  tensor<128x?x2x?xf32> -> tensor<256x128xf32>
```

Scalable tiles!

Scalable (dynamic) data-tiled layouts!



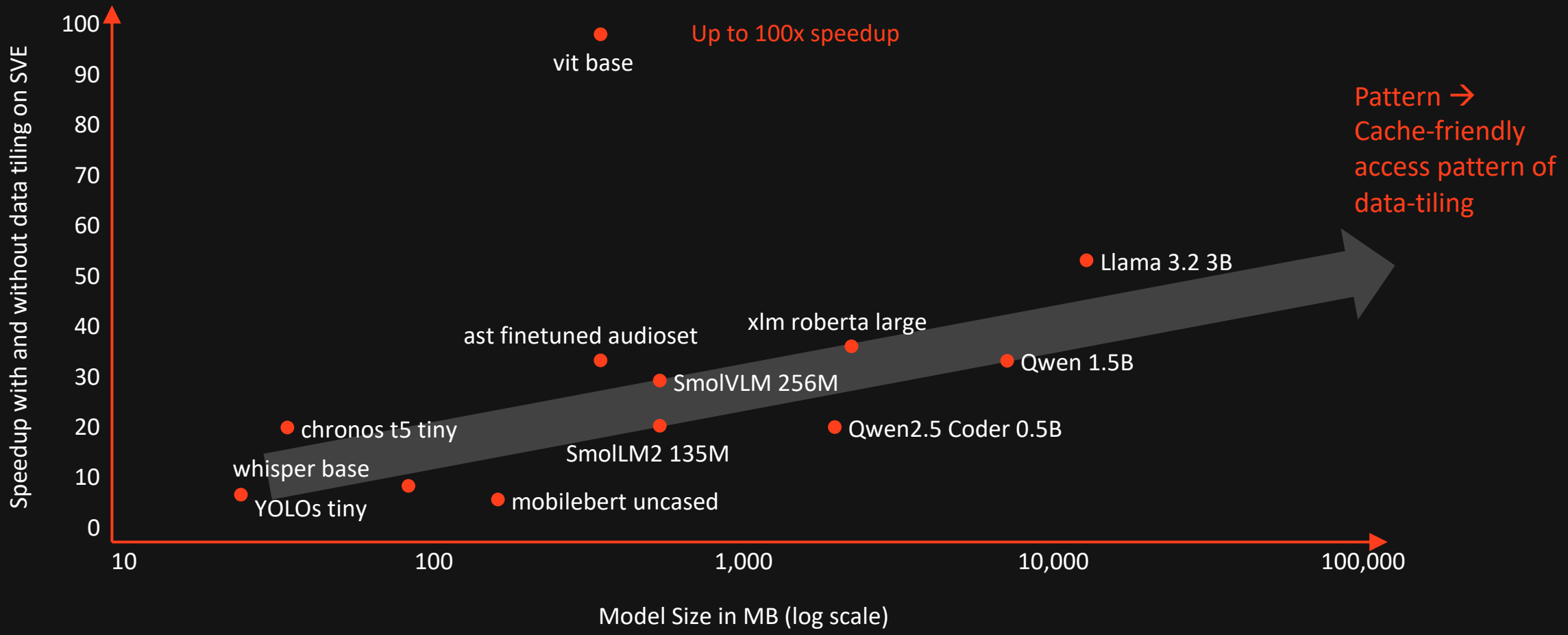
WE USE IREE AS THE UNDERLYING COMPILER INFRASTRUCTURE FOR OUR END-TO-END CASE STUDY



Our contribution:

- **Data-tiling:** Scalable tile size selection and materialization
- **Tiling and fusion:** Aligned scalable data-tiled layouts with tile sizes and tiling interfaces
- **Vectorization:** Scalable vectorization of data-tiled operations

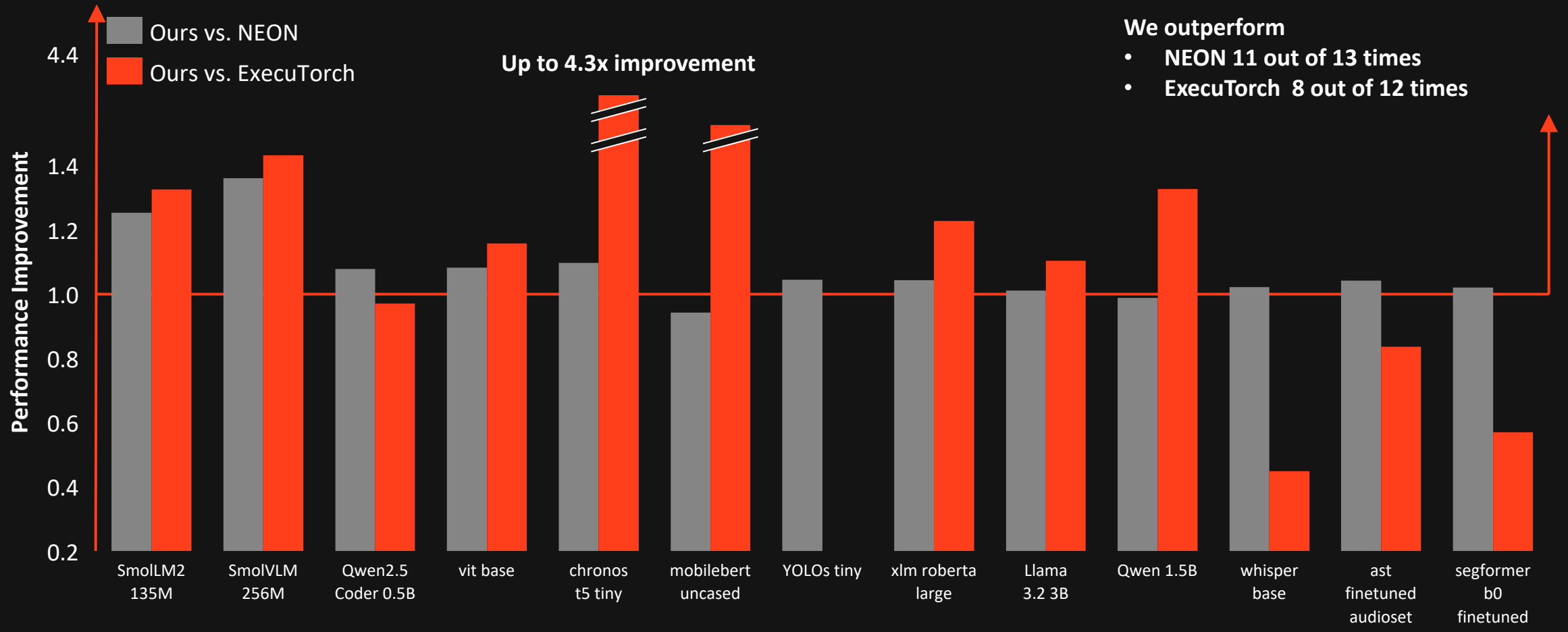
COMPARING DATA-TILED AND NON-DATA-TILED SVE CODE, WE FIND THAT THE SPEEDUP IS LINKED TO THE MODEL SIZE



Please note: Benchmarks from Orion O6 board, with 128-bit vector length (same as NEON), FP32 weights

RESULTS

COMPARING OUR SVE CODE WITH EXISTING SOLUTIONS, WE OUTPERFORM NEON AND EXECUTORCH FOR MANY LLMS



Please note: Benchmarks from Orion O6 board, with 128-bit vector length (same as NEON), FP32 weights

SUMMARY

WE ENABLED DATA TILING FOR SCALABLE VECTOR EXTENSION, IMPROVING THE PERFORMANCE BY UP TO 4.3X

What we achieved

- ✓ Promising performance on FP32 workloads
- ✓ Little overhead from scalable vectors
- ✓ Supports Arm BF16 and I8MM extensions
- ✓ Extendable for similar architectures

What we're working on

- ❑ Not all functionality in upstream MLIR/IREE yet
- ❑ Arm BF16 and I8MM performance needs further improvements
- ❑ Extend to Arm Scalable Matrix Extension (SME) and RISC-V Vector extension