ENABLE EDGE AI PRODUCTS YOU DREAM OF

Understanding linalg.pack and linalg.unpack

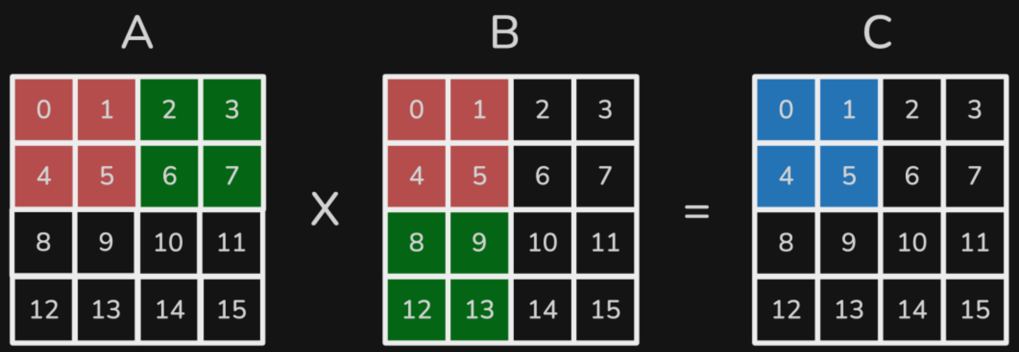
28 OCTOBER 2025

MAXIMILIAN BARTEL

roofline

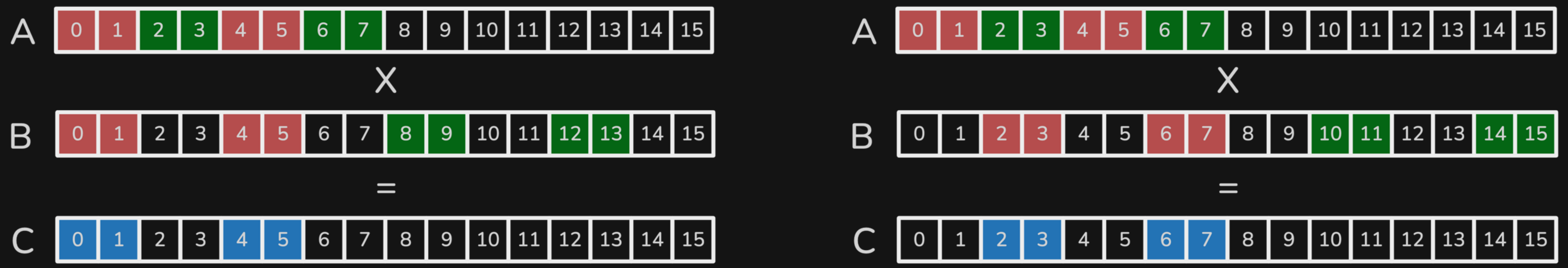
WHY IS A BLOCKED MATMUL IMPLEMENTATION NOT GETTING THE BEST PERFORMANCE?

Logical layout



- Fast algorithm, but it generates cache misses:
- Reads from A are good
- Reads from B and C are not

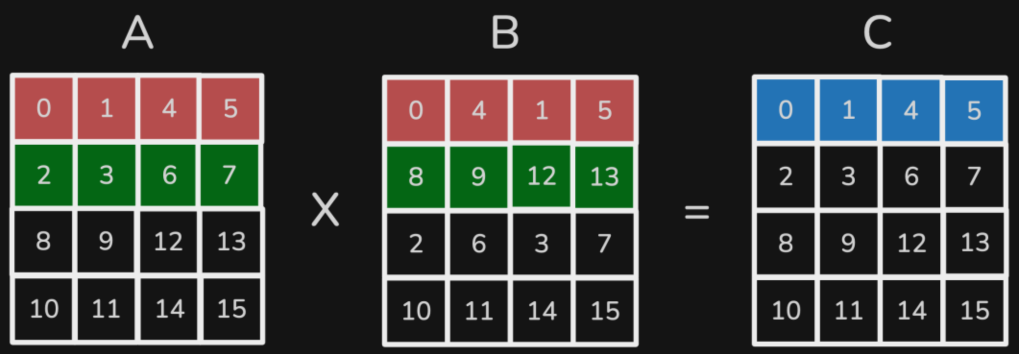
In-memory layout



DATA TILED MATMUL

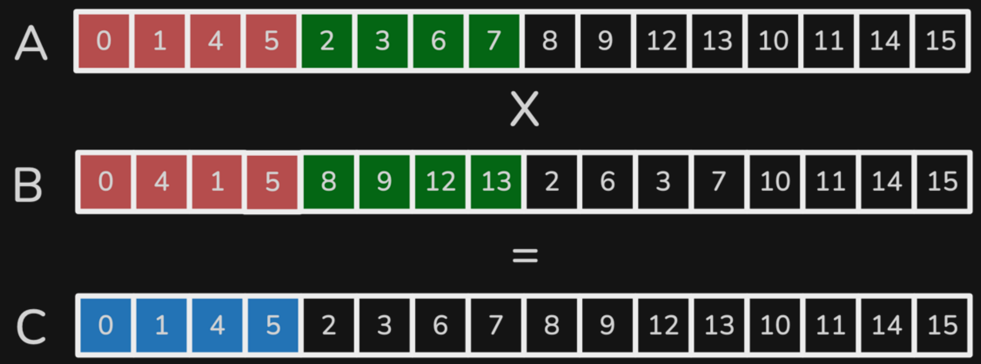
WE CAN IMPROVE MEMORY ACCESS PATTERNS BY CHANGING THE MEMORY LAYOUT

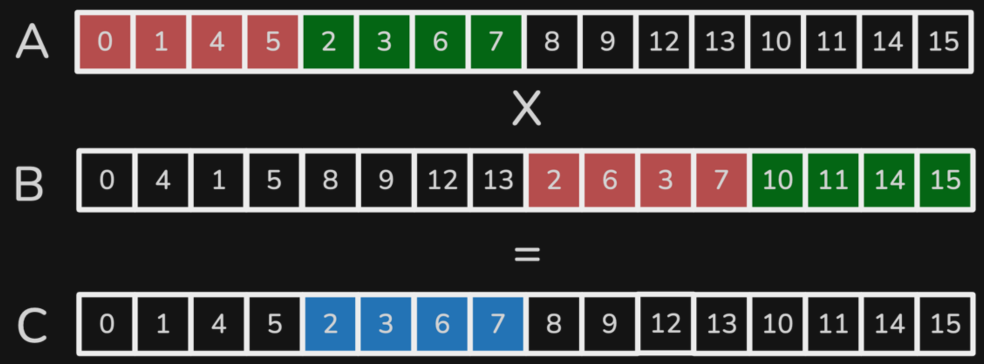
Logical layout



- Changing the memory layout improves cache hit rate
- Optimal layout aligns perfectly with the reads
- This is called data tiling or packed layouts

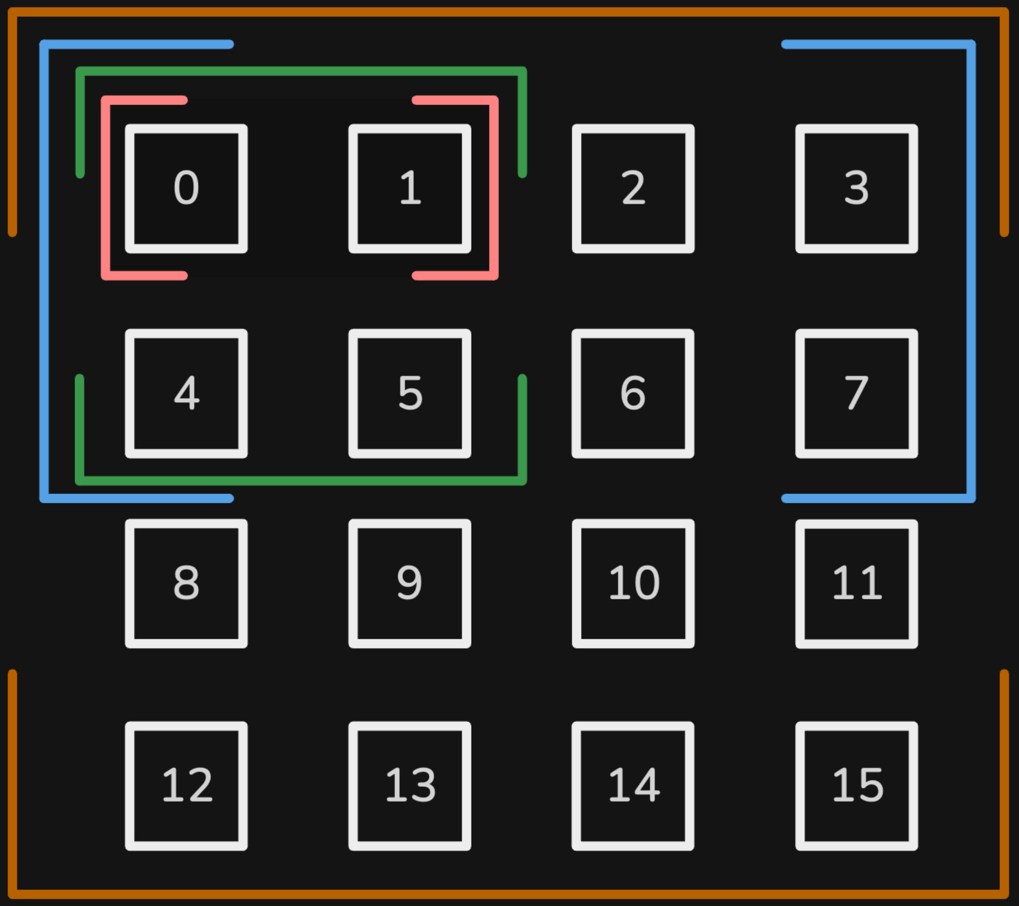
In-memory layout





PACKING INCREASES THE DIMENSIONALITY OF THE TENSOR

Exemplary 2D matrix

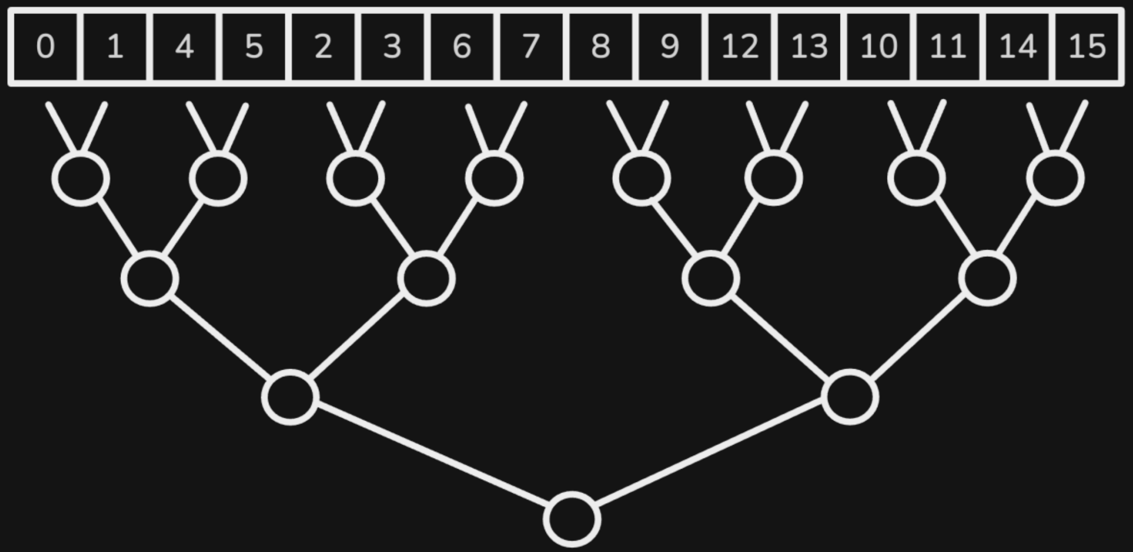


Insights

- Starting form the original logical layout, the new dimensions are marked in different colors
- We go from 4x4 to 2x2x2x2
- This is so far just a view

PACKING MEANS ACTIVELY FLATTEN THE LAYOUT IN MEMORY

Flattened Tensor



Insights

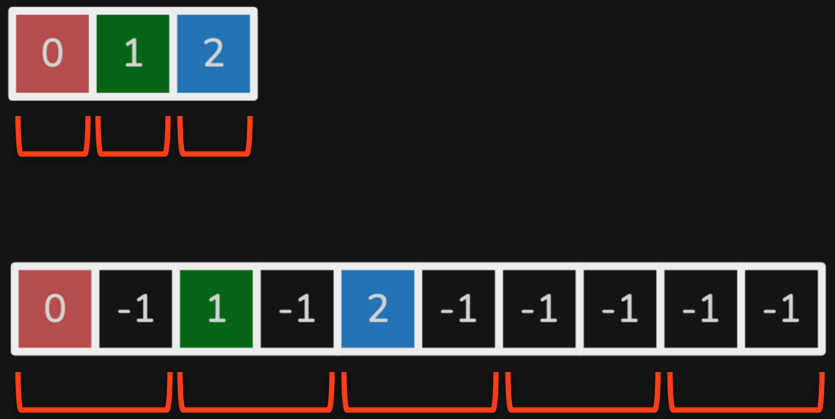
- Packing takes the view and flattens it
- The reduction tree visualizes the dimensions
- These data movements come at a cost!

A PACK OP IS ALLOWED TO PAD TO A SPECIFIC TILE SIZE

Exemplary code

```
func.func @simple_pad_and_pack_static_tiles(
    %input: tensor<3x1xf32>,
    %output: tensor<1x1x5x2xf32>,
    %pad: f32)
        -> tensor<1x1x5x2xf32> {
  %0 = linalg.pack %input
        padding_value(%pad : f32)
        inner_dims_pos = [0, 1]
        inner_tiles = [5, 2]
    into %output : tensor<5x1xf32> -> tensor<1x1x5x2xf32>
  return %0 : tensor<1x1x5x2xf32>
}
```

Insights



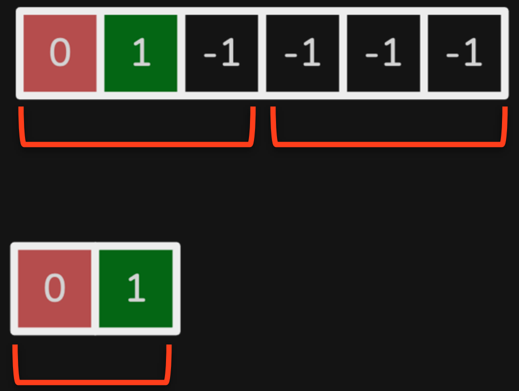
EXAMPLE 2

WE ALSO NEED TO BE ABLE TO REVERT THE PADDING

Exemplary code

```
func.func @unpack_as_pad(
    %arg0: tensor<1x1x2x3xf32>, %arg1: tensor<1x2xf32>)
        -> tensor<1x2xf32> {
  %pack = linalg.unpack %arg0
        inner_dims_pos = [0, 1]
        inner_tiles = [2, 3]
    into %arg1 : tensor<1x1x2x3xf32> -> tensor<1x2xf32>
  return %pack : tensor<1x2xf32>
}
```

Insights



EXAMPLE 3

DYNAMIC SHAPES ARE FULLY SUPPORTED BY THE UNPACK OP

Exemplary code

```
func.func @unpack_fully_dynamic(
    %source: tensor<?x?x?x?xf32>, %dest: tensor<?x?xf32>,
    %tile_n : index, %tile_m : index)
        -> tensor<?x?xf32> {
  %0 = linalg.unpack %source
        inner_dims_pos = [0, 1]
        inner_tiles = [%tile_n, %tile_m]
    into %dest : tensor<?x?x?x?xf32> -> tensor<?x?xf32>
  return %0 : tensor<?x?xf32>
}
```

Insights

- Dynamic dimensions are fully supported by this op
- The inner_tiles can take SSA values
- A custom parser treats it as part of an attribute

EXAMPLE 3

UNPACK WILL DECOMPOSE AND INSERT OPERATIONS TO GET THE SHAPES OF THE TENSORS

Exemplary code

```
func.func @unpack_fully_dynamic(
    %arg0: tensor<?x?x?x?xf32>, %arg1: tensor<?x?xf32>,
    %arg2: index, %arg3: index)
        -> tensor<?x?xf32> {
    ... // Get the dimensions of the packed tensor
    %0 = tensor.empty(%dim, %dim_1, %dim_0, %dim_2) : tensor<?x?x?x?xf32>
    %transposed = linalg.transpose ... permutation = [0, 2, 1, 3]
    %collapsed = tensor.collapse_shape %transposed [[0, 1], [2, 3]]
    %dim_3 = tensor.dim %arg1, %c0 : tensor<?x?xf32>
    %dim_4 = tensor.dim %arg1, %c1 : tensor<?x?xf32>
    %extracted_slice = tensor.extract_slice
        %collapsed[0, 0] [%dim_3, %dim_4] [1, 1] : tensor<?x?xf32> to tensor<?x?xf32>
    %1 = linalg.copy
    return %1 : tensor<?x?xf32>
}
```

Insights

- Decomposing dynamic shapes leads to many dim operations
- Even a padded layout is supported by dynamic shapes
- The output is parsed for its dimensions

roofline

EXAMPLE 4

PACK ALSO SUPPORTS DYNAMIC SHAPES – BUT THE LOWERING CANNOT HANDLE IT YET

Exemplary code

```
func.func @pack_fully_dynamic(
    %source: tensor<?x?xf32>, %dest: tensor<?x?x?x?xf32>,
    %tile_n : index, %tile_m : index, %cst_0 : f32)
        -> tensor<?x?x?x?xf32> {
  %0 = linalg.pack %source
        padding_value(%cst_0 : f32)
        inner_dims_pos = [0, 1]
        inner_tiles = [%tile_n, %tile_m]
    into %dest : tensor<?x?xf32> -> tensor<?x?x?x?xf32>
  return %0 : tensor<?x?x?x?xf32>
}
```

Insights

- This is a valid pack operation
- Decomposition should be like unpack
- However, it doesn’t lower (yet)
- A lowering can be enabled by inser_slice supporting dynamic shapes

EXAMPLE 5

UNIT DIMENSIONS ARE THE MOST COMMON PITFALLS AND PRODUCERS OF BUGS

Exemplary code

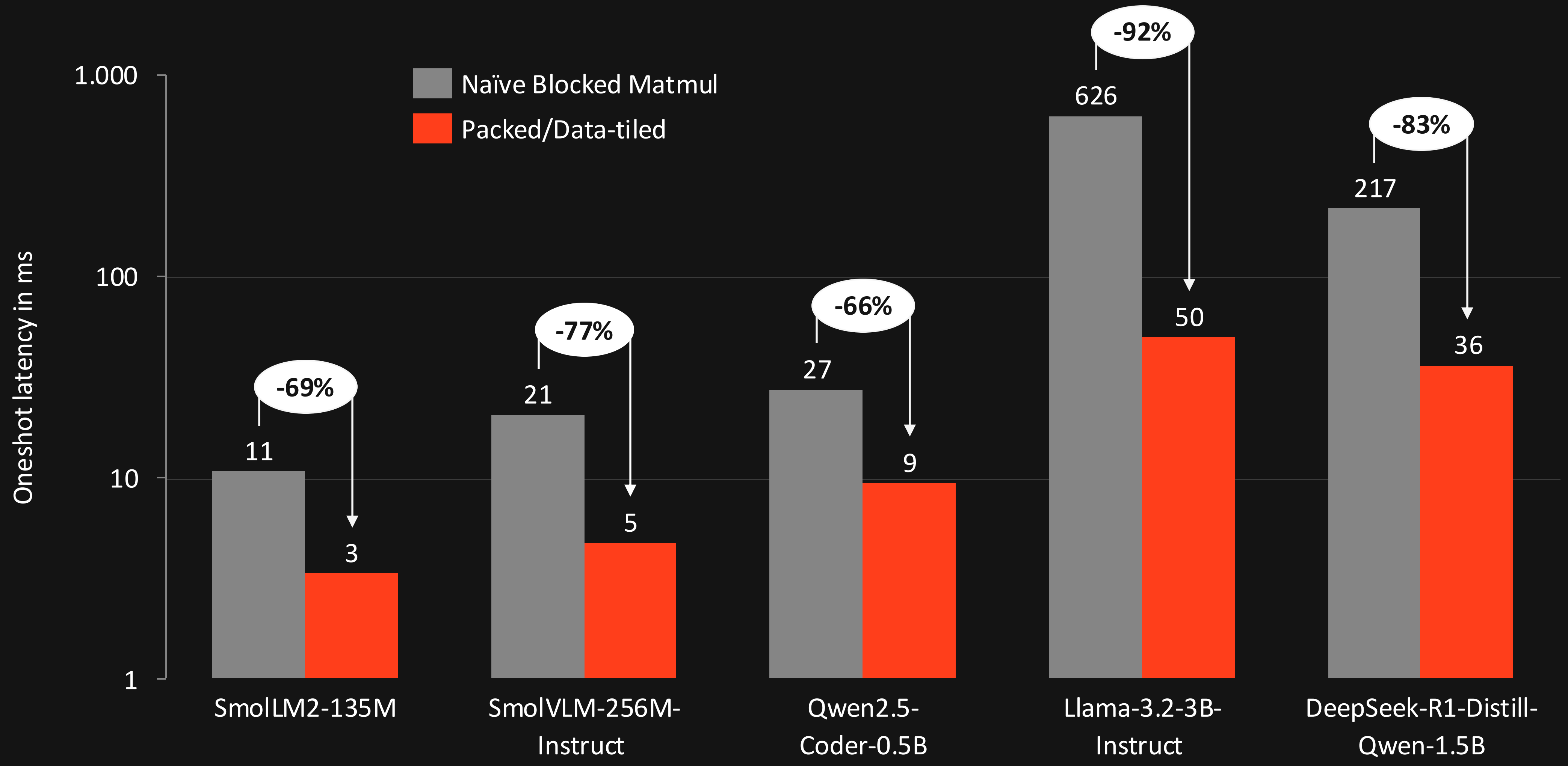
```
func.func @unit_dims(
    %arg0: tensor<1x1x1x4x1xf32>, %arg1: tensor<1x1x4xf32>)
        -> tensor<1x1x1x4x1xf32> {
  %pack = linalg.pack %arg1
        outer_dims_perm = [1, 2, 0]
        inner_dims_pos = [2, 0]
        inner_tiles = [4, 1]
    into %arg0 : tensor<1x1x4xf32> -> tensor<1x1x1x4x1xf32>
  return %pack : tensor<1x1x1x4x1xf32>
}
```

Insights

- Most of the tricky behavior arises with unit dimensions
- This behavior is common after tiling
- Special handling is required to produce “better“ IR in these cases
- This op eliminates data movements
- Untiled non-unit dim dimension in between unit dims are possible

FULL MODEL BENCHMARKS

WHEN RUNNING FULL MODELS WE SEE LATENCY DECREASES FROM 69% TO 92%



roofline

PACK AND UNPACK MAKE MEMORY ACCESSES VERY EFFICIENT, BUT THEY COME AT A COST

How to use pack effectively

- Packing can degrade performance for single kernels
- Only full model compilation can fuse packing operators into producers to hide the movement cost

What are unexplored paths?

- Explore data tiling for other kernels like convolutions
- Explore matrices as representations remove complex attributes (Triton)