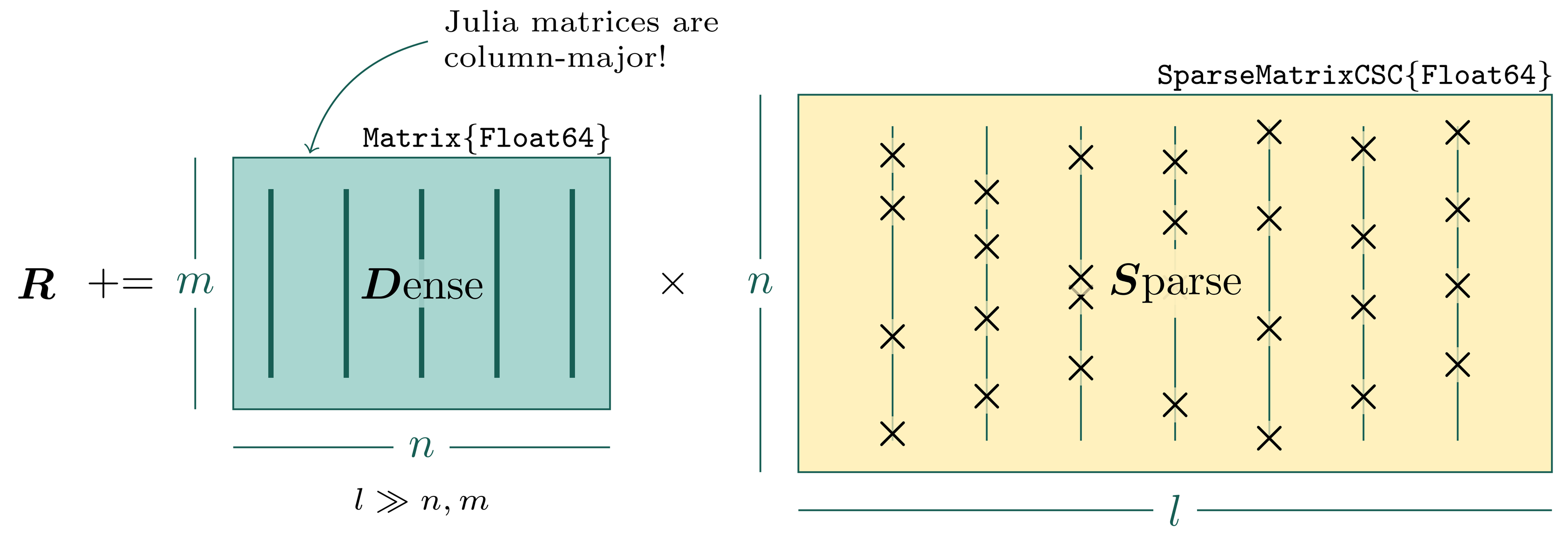
# `ThreadedDenseSparseMul.jl`

## Outperforming `MKLSparse` in 6 LOC.

Romeo Valentin (romeov@stanford.edu), Mykel J. Kochenderfer
Stanford Intelligent Systems Lab

Stanford University

### We need...

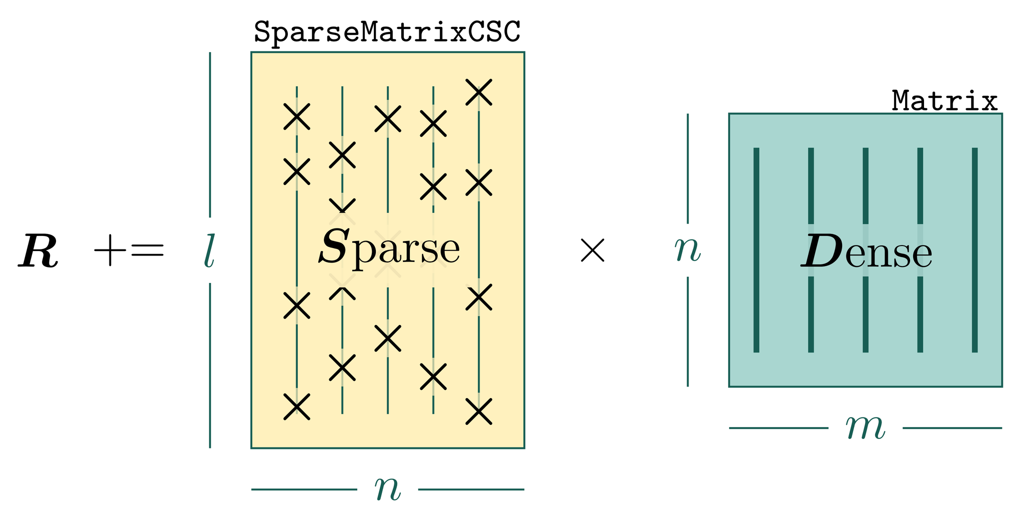



i.e., threaded dense times sparse matmul.

Our options:

- `ThreadedDenseSparseMul.jl` (this poster)

### ...but everyone else does

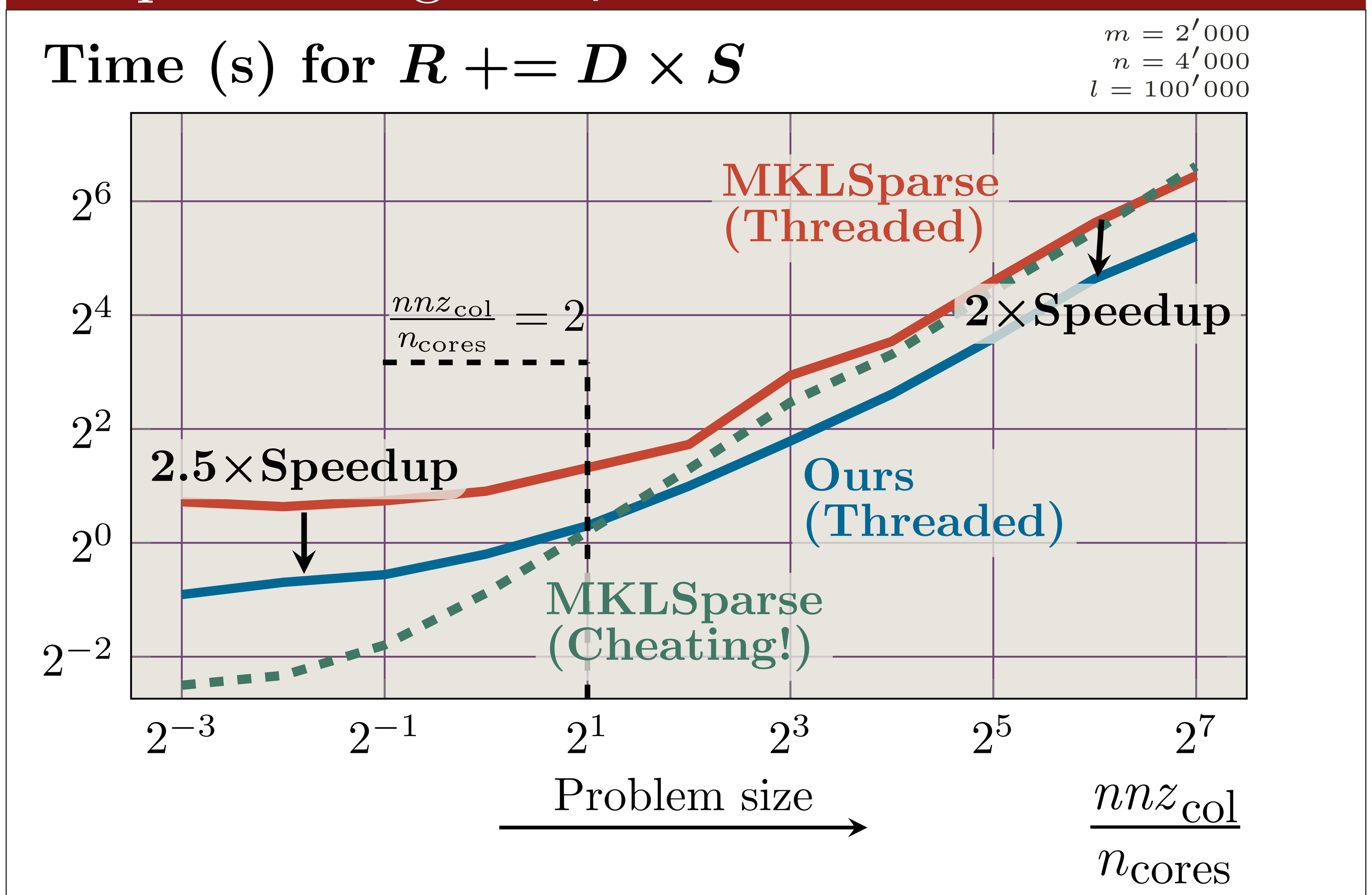



i.e., threaded sparse times dense matmul.

Our options:

- `MKLSparse.jl`, `ThreadedSparseCSR.jl`, `ThreadedSparseArrays.jl`

## Outperforming `MKLSparse`: Benchmark results.




**Ours** outperforms **MKLSparse** with speedup 2×!
For large $nnz_{\text{col}}$, **Ours** even outperforms **MKLSparse (Cheating!)** which computes sparse×dense directly.

## The Secret Sauce: Chris Elrod's `Polyester.jl`

Hand-rolling loops has never been so easy! Also, memory layout is actually perfect for dense×sparse!

```julia
using Polyster
function mul!(C::Matrix{T}, A::Matrix{T}, B::SparseMatrixCSC{T},
              α::Number, β::Number)
  @inbounds begin
    C .*= β
    @batch for j in axes(B, 2)
        C[:, j] .+= A * (α.*B[:, j])
    end
    return C
  end
end
```

## Why can we win this?

Better memory layout. `MKLSparse` transposes (twice).
Let's count memory movements of `mul!(R', S', D')'`.

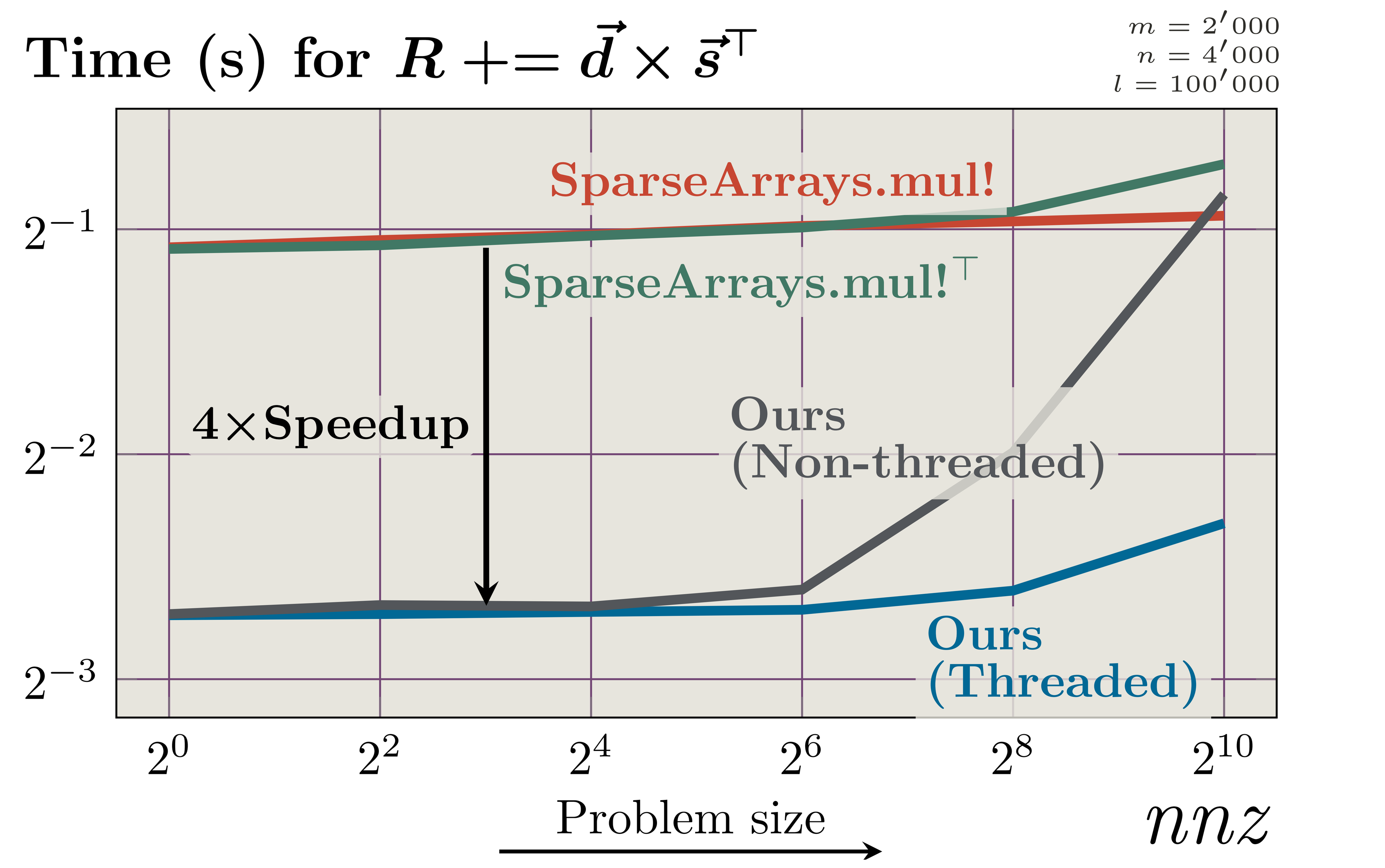
- Transpose: $\approx 2 \times \Theta(m \cdot l)$
- Multiply: $\Theta(nnz_{\text{col}} \cdot m \cdot l)/n_{\text{cores}}$

$$\Rightarrow \left(2 + \frac{nnz_{col}}{n_{\text{cores}}}\right) \times \Theta(m \cdot l)$$

$\Rightarrow$ For $(nnz_{\text{col}} > 2 \times n_{\text{cores}})$, transposing dominates!

## Things get even better: outer product.

`MKLSparse` doesn't have an outer product at all!
And we can outperform `SparseArrays`.




**Ours** beats **SparseArrays.mul!/mul!$^\top$** with speedup 4× in 5 LOC (single-threaded, no `Polyster.jl`), or 9 LOC (multi-threaded, w/ `Polyster.jl`).

## Takeaways

- Good memory layout lets us outperform MKL.
- Don't be afraid to "hand-roll" tight loops.
- Know when you are memory or compute bound.
- Benchmark, benchmark, benchmark!

All benchmarks collected on mobile workstation with `11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz, 32GB RAM`, using the `Chairmarks.jl` library.

**Acknowledgements:** This work was created as a part of `KSVD.jl`, which is funded by a grant from Stanford HCAI and Google. We also thank our collaborators at SISL, as well as Chris Elrod for his work on Polyester.jl

SISL