Monarch & Structured Matrices

→ Matrices are a programming language
→ Standard algorithm is an interpreter. $O(n^2)$
→ BUT, we can also build an optimizing compiler

Worried about matrix & vector multiplication

Fast Fourier Transformer: $O(n^2)$ to $O(n \log n)$ (FFT)

Structured & Sparse Matrices

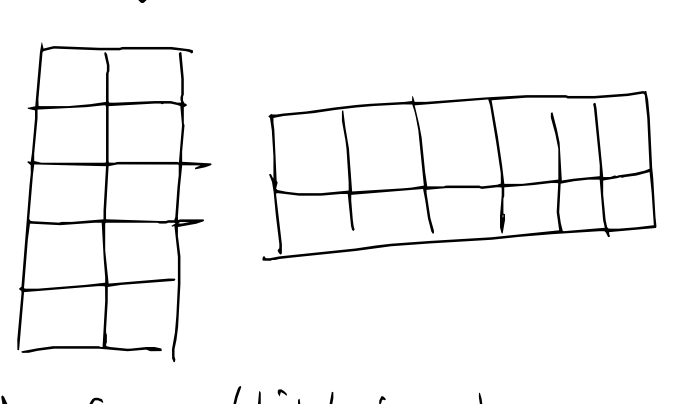
A structured matrix is any matrix class where:
- → A representative can be described in $O(n^2)$ memory
- → Matrix vector multiply can be done in $O(n^2)$ compute

Characterization
- → Take ANY element M of a structured matrix class
- → Write efficient Mv as a linear arithmetic unit
- → Equivalent to factoring M into a product of sparse matrices (

Structured Matrices

→ Low Rank

Can be expressed as a product of one very tall but thin matrix & one very wide but short matrix



Instead of multiplying large matrix with vector, we can multiply vector individually with each of the above, which is much faster. Each matrix is smaller

Sparse

Not predictably structured

Multi-level low rank
1. Take low rank approx for the entire matrix
2. Then, you can take low rank approx for some blocks along diagonal
3. Then, take Low rank approx for even more smaller blocks along the diagonal

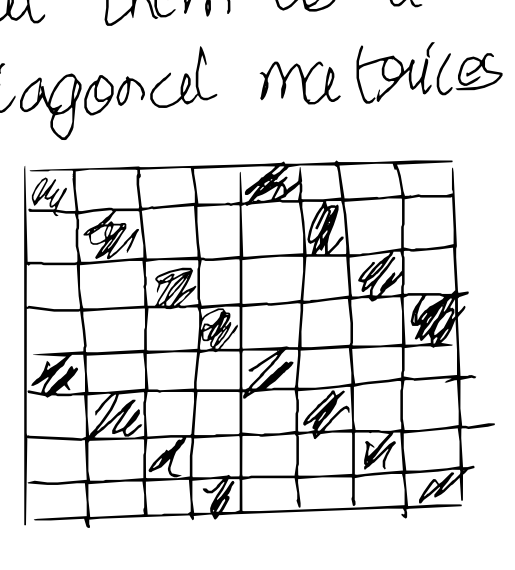
Spectral clustering :- Can be used to identify high rank submatrices

Hierachical matrix
- → Write whole matrix as a tree of smaller blocks
- → Tree can get deep in some parts of the matrix & not as deep in some other parts
- → Each block at the leaf turns out to be some kind of low rank matrix

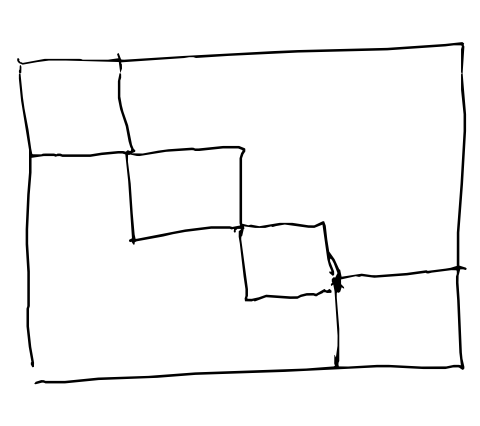
Butterfly & Monarch matrices

→ Tri Dao 2020: Any class of structured matrix is a product of butterfly matrices & their transposes
- → Butterfly inspired by FFT
- → Bit reversal permutation
- → Can be expressed as certain kinds of very sparse matrices

Butterfly matrices
- → Butterfly tile: Only has non-zero elements along diagonal, AND smaller secondary diagonals
- → Look at them as a 4x4 grid of diagonal matrices



Butterfly Factor Matrix
- → Block diagonal matrix where each block is a butterfly tile
- → Butterfly tiles can be different sizes



Butterfly matrix
- → Take a bunch of butterfly block diagonal matrices, with different tile sizes, & multiply them together
- → Scale tile size generally in powers of 2
- → Taking product gives butterfly matrix
- → Good building block for many common matrices. Eg: FFT, convolutions, Hadamard product

Butterfly matrices : Recap

Pros	Cons
– Sparse	– Not most multiplication
– Predictable structure	– Must be square
– Expressive	– Must be power of 2 size

Monarch Matrices: M = PLPR

→ Matrix M will be a product of 4 matrices

R – Block diagonal matrix
- → Lowers param count
- Multiply each block with a chunk of I/P vector
- Each of the smaller pieces still looks like matrix multiplication → can efficiently utilize hardware
- Can perform transformations only within each chunk of the I/P vector

WE NEED TO MIX THINGS AROUND!
↓
Do this with a permutation

Permutations are square matrices
Spreads out elements of each chunk of R so that they are all in different chunks

Elements in diff chunks can now interact with each other. Apply a different block diagonal matrix 'L'

'P' → Permutation matrices

Where did this monarch matrix idea come from?

Suppose you split the factorization
Right side will be all block diagonal matrices
Product of block diagonal will stay block diagonal.
Even if we remultiply, we still get block diagonal

On left side, we get:
8x8 blocks, each of which is a diagonal matrix

Hence, we get diagonal block matrix (don't confuse with block diagonal matrix).

This property is ALSO PRESERVED by matrix multiplication.
Multiplying all matrices on the left will give a diagonal block matrix

∴ Monarch Matrices
Product of diagonal block matrix & block diagonal matrix

DB – BD Form

Write all butterfly matrices in this format of DB-BD Form.
Instead of having $O(n \log n)$ parameters, we now have $O(n^{1.5})$ params, which is better.

More freedom, more param count, but more efficient

Conjugate a block diagonal matrix with a special permutation, it gives a diagonal block matrix.
This the matrix 'P'

∴ Monarch matrix = PLPR

Is monarch matrix an approximation? YES

Approximate row vs column matrix mul using singular value decomposition

Higher Powers

M: Monarch matrices
M^*: Transpose of monarch matrices
MM^* and M^*M: More expressive matrices

$$MM^* = (PL_1PR_1)(PL_2PR_2)$$
$$= PL_1PR_1PL_2PR_2$$
$$= PL_1PR_1R_2^TPL_2^TP$$
$$= PAPBPCP$$

$$M^*M = (PL_1PR_1)(PL_2PR_2)$$
$$= R_1^TPL_1^TPPL_2PR_2$$
$$= R_1^TPL_1^TL_2PR_2$$ Not sure how this works
$$= APBPC$$

$$M_{ij} = A_i D_{ij} C_j$$
↓ BD, ↓ DB, ↓ BD

Unclear how to find factors

Whats Next?

→ $A = P_1MP_2$ — M – Monarch; P_1, P_2 → permutations

P_1 is free: absorbed in M
Decomposition algorithm would be nice

→ Hierachical decomposition
- → Integrate to existing flow
- → Use approximations when they are good enough
- → Incorporate monarch
- → Decompose by tiling

Result: optimizing matrix compiler

MLR matrices apparently better than monarch matrices