Parallel Numerical Algorithms

Chapter 6 – Matrix Models
Section 6.2 – Low Rank Approximation

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

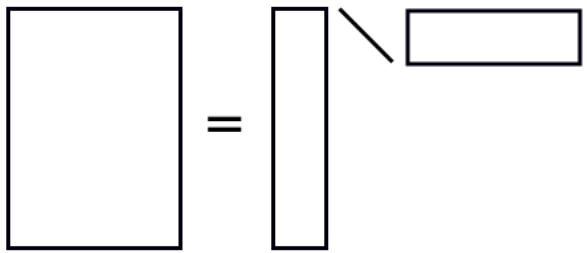
CS 554 / CSE 512

Outline

Rank-k Singular Value Decomposition (SVD)

For any matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ of rank k there exists a factorization

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$



- $\boldsymbol{U} \in \mathbb{R}^{m \times k}$ is a matrix of orthonormal left singular vectors
- $\boldsymbol{D} \in \mathbb{R}^{k \times k}$ is a nonnegative diagonal matrix of singular values in decreasing order $\sigma_1 \geq \cdots \geq \sigma_k$
- $\boldsymbol{V} \in \mathbb{R}^{n \times k}$ is a matrix of orthonormal right singular vectors

Truncated SVD

Given $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ seek its best $k < \text{rank}(A)$ approximation

$$\boldsymbol{B} = \underset{\boldsymbol{B} \in \mathbb{R}^{m \times n}, \text{rank}(\boldsymbol{B}) \leq k}{\text{argmin}} (||\boldsymbol{A} - \boldsymbol{B}||_2)$$

- Eckart-Young theorem: given SVD

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{D}_1 & \\ & \boldsymbol{D}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{V}_1 & \boldsymbol{V}_2 \end{bmatrix}^T \Rightarrow \boldsymbol{B} = \boldsymbol{U}_1 \boldsymbol{D}_1 \boldsymbol{V}_1^T$$

  where $\boldsymbol{D}_1$ is $k \times k$.
- $\boldsymbol{U}_1 \boldsymbol{D}_1 \boldsymbol{V}_1^T$ is the rank-k *truncated SVD* of $\boldsymbol{A}$ and

$$||\boldsymbol{A} - \boldsymbol{U}_1 \boldsymbol{D}_1 \boldsymbol{V}_1^T||_2 = \underset{\boldsymbol{B} \in \mathbb{R}^{m \times n}, \text{rank}(\boldsymbol{B}) \leq k}{\min} (||\boldsymbol{A} - \boldsymbol{B}||_2) = \sigma_{k+1}$$

Computational Cost

Given a rank k truncated SVD $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$

- Performing approximately $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ requires $O(mk)$ work

$$\boldsymbol{y} \approx \boldsymbol{U}(\boldsymbol{D}(\boldsymbol{V}^T\boldsymbol{x}))$$

- Solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ requires $O(mk)$ work via approximation

$$\boldsymbol{x} \approx \boldsymbol{V}\boldsymbol{D}^{-1}\boldsymbol{U}^T\boldsymbol{b}$$

Computing the Truncated SVD

Reduction to upper-Hessenberg form via two-sided orthogonal updates can compute full SVD

- Given full SVD can obtain truncated SVD by keeping only largest singular value/vector pairs
- Given set of transformations $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_s$ so that $\boldsymbol{U} = \boldsymbol{Q}_1 \cdots \boldsymbol{Q}_s$, can obtain leading k columns of $\boldsymbol{U}$ by computing

$$\boldsymbol{U}_1 = \boldsymbol{Q}_1 \left(\cdots \left(\boldsymbol{Q}_s \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \end{bmatrix} \right) \right)$$

- This method requires $O(mn^2)$ work for the computation of singular values and $O(mnk)$ for k singular vectors

Computing the Truncated SVD by Krylov Subspace Methods

Seek $k \ll m, n$ leading right singular vectors of $\boldsymbol{A}$

- Find a basis for Krylov subspace of $\boldsymbol{B} = \boldsymbol{A}^T \boldsymbol{A}$
- Rather than computing $\boldsymbol{B}$, compute products $\boldsymbol{B}\boldsymbol{x} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x})$
- For instance, do $k' \geq k + O(1)$ iterations of Lanczos and compute k Ritz vectors to estimate singular vectors $\boldsymbol{V}$
- Left singular vectors can be obtained via $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{D}$
- This method requires $O(mnk)$ work for k singular vectors
- However, $\Theta(k)$ sparse-matrix-vector multiplications are needed (high latency and low flop/byte ratio)

Generic Low-Rank Factorizations

A matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is rank k, if for some$\boldsymbol{X} \in \mathbb{R}^{m \times k}, \boldsymbol{Y} \in \mathbb{R}^{n \times k}$ with $k \leq \min(m, n)$,

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{Y}^T$$

- If $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{Y}^T$ (exact low rank factorization), we can obtain reduced SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ via
  1. $[\boldsymbol{U}_1, \boldsymbol{R}] = \text{QR}(\boldsymbol{X})$
  2. $[\boldsymbol{U}_2, \boldsymbol{D}, \boldsymbol{V}] = \text{SVD}(\boldsymbol{R}\boldsymbol{Y}^T)$
  3. $\boldsymbol{U} = \boldsymbol{U}_1\boldsymbol{U}_2$

  with cost $O(mk^2)$ using an SVD of a $k \times k$ rather than $m \times n$ matrix
- If instead $||\boldsymbol{A} - \boldsymbol{X}\boldsymbol{Y}^T||_2 \leq \varepsilon$ then $||\boldsymbol{A} - \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T||_2 \leq \varepsilon$
- So we can obtain a truncated SVD given an optimal generic low-rank approximation

Rank-Revealing QR

If $\boldsymbol{A}$ is of rank k and its first k columns are linearly independent

$$\boldsymbol{A} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

where $\boldsymbol{R}_{11}$ is upper-triangular and $k \times k$ and $\boldsymbol{Q} = \boldsymbol{Y}\boldsymbol{T}\boldsymbol{Y}^T$ with $n \times k$ matrix $\boldsymbol{Y}$

- For arbitrary $\boldsymbol{A}$ we need column ordering permutation $\boldsymbol{P}$

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{P}$$

- *QR with column pivoting* (due to Gene Golub) is an effective method for this
  - pivot so that the leading column has largest 2-norm
  - method can break in the presence of roundoff error (see Kahan matrix), but is very robust in practice

Low Rank Factorization by QR with Column Pivoting

QR with column pivoting can be used to either

- determine the (numerical) rank of $\boldsymbol{A}$
- compute a low-rank approximation with a bounded error

performs only $O(mnk)$ rather than $O(mn^2)$ work for a full QR or SVD

Parallel QR with Column Pivoting

In distributed-memory, column pivoting poses further challenges

- Need at least one message to decide on each pivot column, which leads to $\Omega(k)$ synchronizations
- Existing work tries to pivot many columns at a time by finding subsets of them that are sufficiently linearly independent
- Randomized approaches provide alternatives and flexibility

Randomization Basics

Intuition: consider a random vector $\boldsymbol{w}$ of dimension n, all of the following holds with high probability in exact arithmetic

- Given any basis $\boldsymbol{Q}$ for the n dimensional space, random $\boldsymbol{w}$ is not orthogonal to any row of $\boldsymbol{Q}^T$
- Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$ where $\boldsymbol{V}^T \in \mathbb{R}^{n\times k}$
- Vector $\boldsymbol{w}$ is at random angle with respect to any row of $\boldsymbol{V}^T$, so $\boldsymbol{z} = \boldsymbol{V}^T\boldsymbol{w}$ is a random vector
- $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{z}$ is random linear combination of cols of $\boldsymbol{U}\boldsymbol{D}$
- Given k random vectors, i.e., random matrix $\boldsymbol{W} \in \mathbb{R}^{n\times k}$
- Columns of $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{W}$ gives k random linear combinations of columns of in $\boldsymbol{U}\boldsymbol{D}$
- *$\boldsymbol{B}$ has the same span as $\boldsymbol{U}$!*

Using the Basis to Compute a Factorization

If $\boldsymbol{B}$ has the same span as the range of $\boldsymbol{A}$

- $[\boldsymbol{Q}, \boldsymbol{R}] = \text{QR}(\boldsymbol{B})$ gives orthogonal basis $\boldsymbol{Q}$ for $\boldsymbol{B} = \boldsymbol{AW}$
- $\boldsymbol{QQ}^T\boldsymbol{A} = \boldsymbol{QQ}^T\boldsymbol{UDV}^T = (\boldsymbol{QQ}^T\boldsymbol{U})\boldsymbol{DV}^T$, now $\boldsymbol{Q}^T\boldsymbol{U}$ is orthogonal and so $\boldsymbol{QQ}^T\boldsymbol{U}$ is a basis for the range of $\boldsymbol{A}$
- so compute $\boldsymbol{H} = \boldsymbol{Q}^T\boldsymbol{A}$, $\boldsymbol{H} \in \mathbb{R}^{k\times n}$ and compute $[\boldsymbol{U}_1, \boldsymbol{D}, \boldsymbol{V}] = \text{SVD}(\boldsymbol{H})$
- then compute $\boldsymbol{U} = \boldsymbol{QU}_1$ and we have a rank k truncated SVD of $\boldsymbol{A}$

$$\boldsymbol{A} = \boldsymbol{UDV}^T$$

Cost of the Randomized Method

- Matrix multiplications e.g. $\boldsymbol{AW}$, all require $O(mnk)$ operations
- QR and SVD require $O((m+n)k^2)$ operations
- If $k \ll \min(m,n)$ the bulk of the computation here is within matrix multiplication, which can be done with fewer synchronizations and higher efficiency than QR with column pivoting or Arnoldi

Randomized Approximate Factorization

Now lets consider the case when $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T + \boldsymbol{E}$ where $\boldsymbol{D} \in \mathbb{R}^{k\times k}$ and $\boldsymbol{E}$ is a small perturbation

- $\boldsymbol{E}$ may be noise in data or numerical error
- To obtain a basis for $\boldsymbol{U}$ it is insufficient to multiply by random $\boldsymbol{W} \in \mathbb{R}^{n\times k}$, due to influence of $\boldsymbol{E}$
- However, oversampling, for instance $l = k + 10$, and random $\boldsymbol{W} \in \mathbb{R}^{n\times l}$ gives good results
- A Gaussian random distribution provides particularly good accuracy
- So far the dimension of $\boldsymbol{W}$ has assumed knowledge of the target approximate rank k, to determine it dynamically generate vectors (columns of $\boldsymbol{W}$) one at a time or a block at a time, which results in a provably accurate basis

Cost Analysis of Randomized Low-rank Factorization

- The cost of the randomized algorithm for is

$$T_p^{\mathrm{MM}}(m,n,k) + T_p^{\mathrm{QR}}(m,k,k)$$

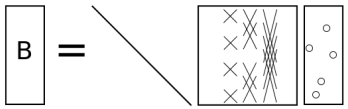
  which means that the work is $O(mnk)$ and the algorithm is well-parallelizable
- This assumes we factorize the basis by QR and SVD of $\boldsymbol{R}$

Fast Algorithms via Pseudo-Randomness

We can lower the number of operations needed by the randomized algorithm by generating $\boldsymbol{W}$ so that $\boldsymbol{AW}$ can be computed more rapidly

- Generate $\boldsymbol{W}$ as a pseudo-random matrix

$$\boldsymbol{W} = \boldsymbol{DFR}$$



- $\boldsymbol{D}$ is diagonal with random elements
- $\boldsymbol{F}$ can be applied to a vector in $O(n \log(n))$ operations
  - e.g. DFT or Hadamard matrix $\boldsymbol{H}_{2n} = \begin{bmatrix} \boldsymbol{H}_n & \boldsymbol{H}_n \\ \boldsymbol{H}_n & -\boldsymbol{H}_n \end{bmatrix}$
- $\boldsymbol{R}$ is $p \approx k$ columns of the $n \times n$ identity matrix
- Computes $\boldsymbol{AW}$ with $O(mn \log(n))$ operations (if $m > n$)

Cost of Pseudo-Randomized Factorization

Instead of matrix multiplication, apply m FFTs of dimension n

- Each FFT is independent, so it suffices to perform a single transpose
- So we have the following overall cost

$$O\left(\frac{mn\log(n)}{p}\cdot\gamma\right)+T_p^{\text{all-to-all}}(mn/p)+T_p^{\text{QR}}(m,k,k)$$

  assuming $m > n$
- This is lower with respect to the unstructured/randomized version, however, this idea does not extend well to the case when $\boldsymbol{A}$ is sparse

Hierarchical Low Rank Structure

- Consider two-way partitioning of vertices of a graph
- The connectivity within each partition is given by a block diagonal matrix

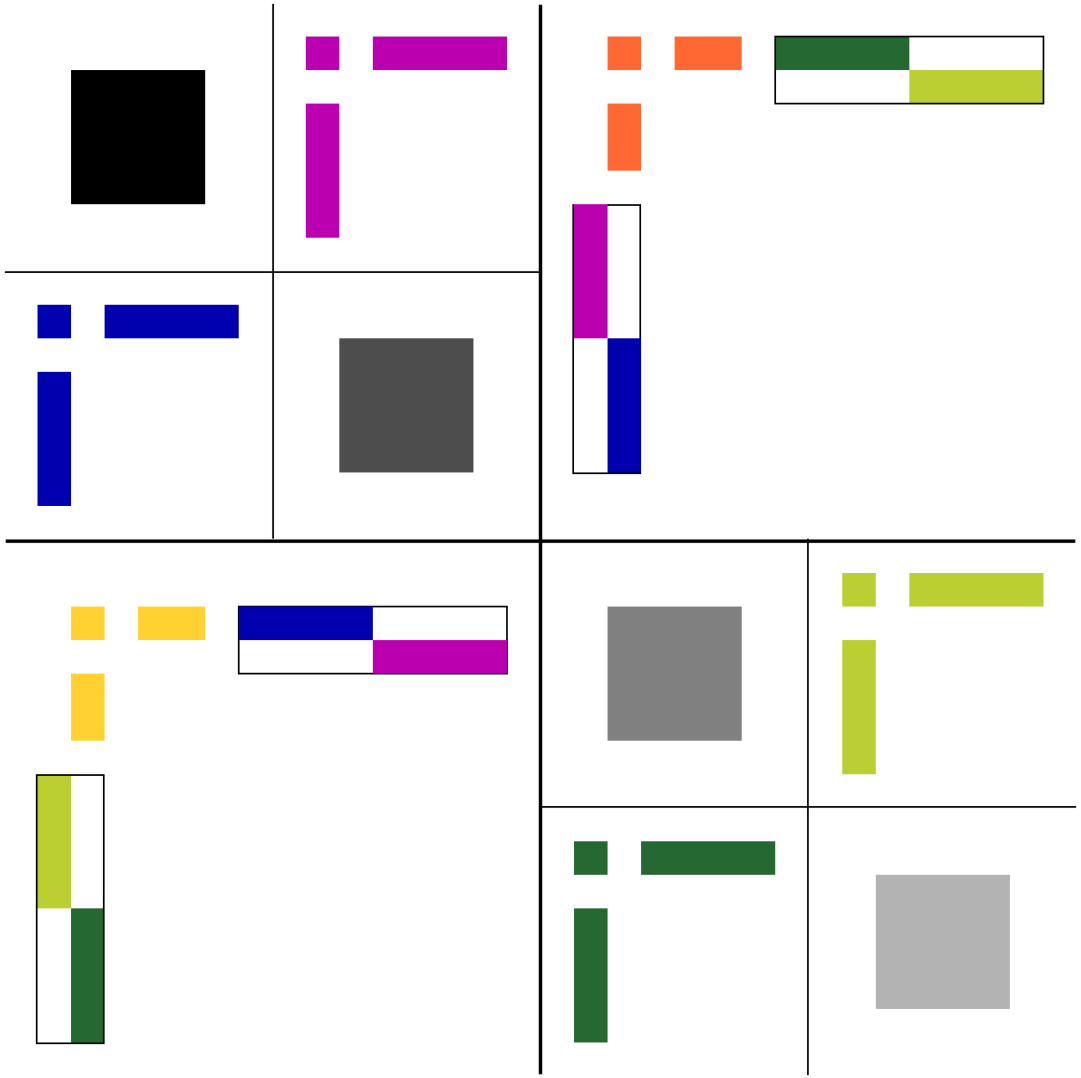
$$\begin{bmatrix} \boldsymbol{A}_1 & \\ & \boldsymbol{A}_2 \end{bmatrix}$$

- If the graph is nicely *separable* there is little connectivity between vertices in the two partitions
- Consequently, it is often possible to approximate the off-diagonal blocks by low-rank factorization

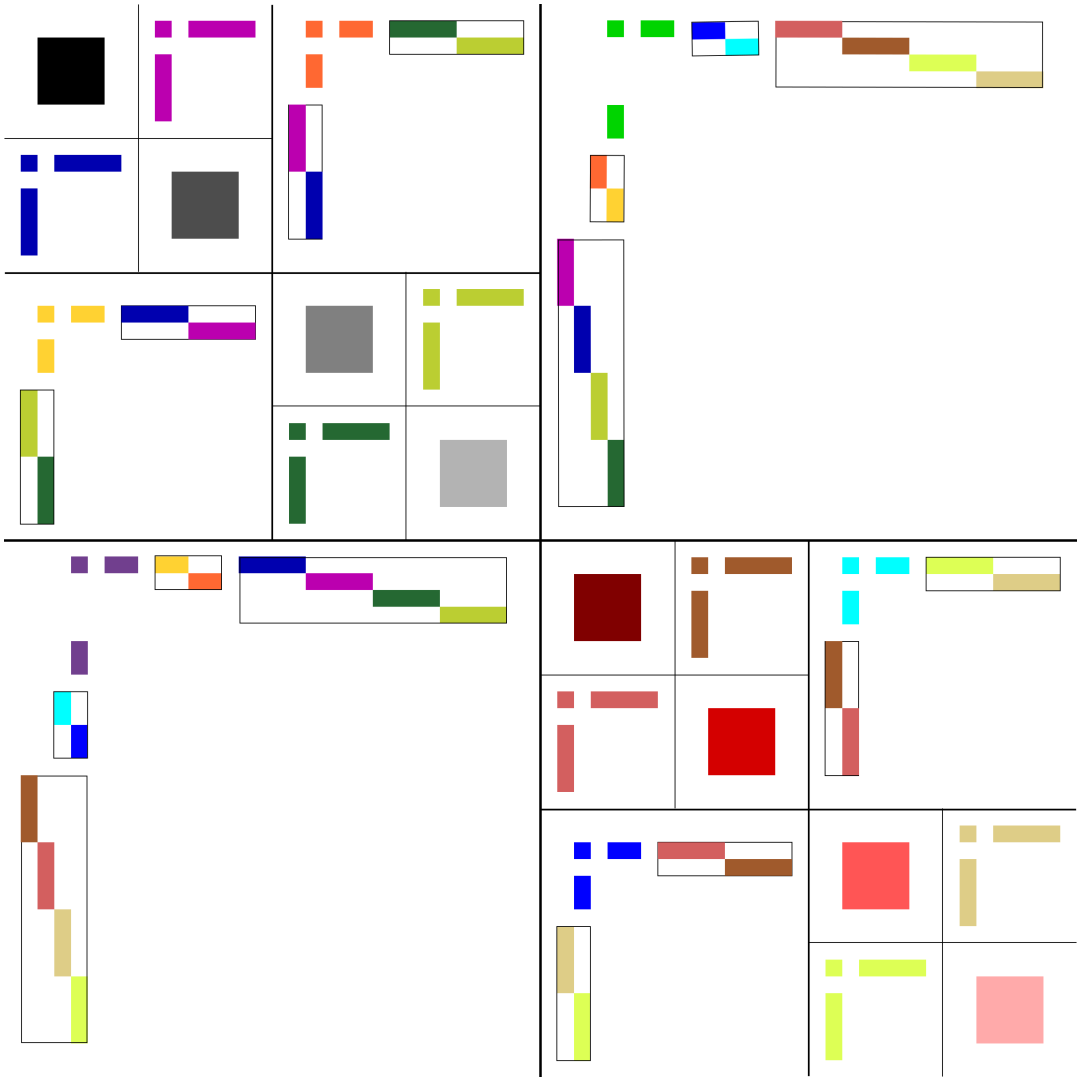
$$\begin{bmatrix} \boldsymbol{A}_1 & \boldsymbol{U}_1\boldsymbol{D}_1\boldsymbol{V}_1^T \\ \boldsymbol{U}_2\boldsymbol{D}_2\boldsymbol{V}_2^T & \boldsymbol{A}_2 \end{bmatrix}$$

- Doing this recursively to $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ yields a matrix with hierarchical low-rank structure

HSS Matrix, Two Levels



HSS Matrix, Three Levels



HSS Matrix Formal Definition

- The l-level HSS factorization is described by

$$\mathcal{H}_l(\boldsymbol{A}) = \begin{cases} \{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{T}_{12}, \boldsymbol{T}_{21}, \boldsymbol{A}_{11}, \boldsymbol{A}_{22}\} & : l = 1 \\ \{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{T}_{12}, \boldsymbol{T}_{21}, \mathcal{H}_{l-1}(\boldsymbol{A}_{11}), \mathcal{H}_{l-1}(\boldsymbol{A}_{22})\} & : l > 1 \end{cases}$$

- The low-rank representation of the diagonal blocks is given by $\boldsymbol{A}_{21} = \bar{\boldsymbol{U}}_2 \boldsymbol{T}_{21} \bar{\boldsymbol{V}}_1^T$, $\boldsymbol{A}_{12} = \bar{\boldsymbol{U}}_1 \boldsymbol{T}_{12} \bar{\boldsymbol{V}}_2^T$ where for $a \in \{1, 2\}$,

$$\bar{\boldsymbol{U}}_a = \mathcal{U}_a(\mathcal{H}_l(\boldsymbol{A})) = \begin{cases} \boldsymbol{U}_a & : l = 1 \\ \begin{bmatrix} \mathcal{U}_1(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) & \boldsymbol{0} \\ \boldsymbol{0} & \mathcal{U}_2(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) \end{bmatrix} \boldsymbol{U}_a & : l > 1 \end{cases}$$

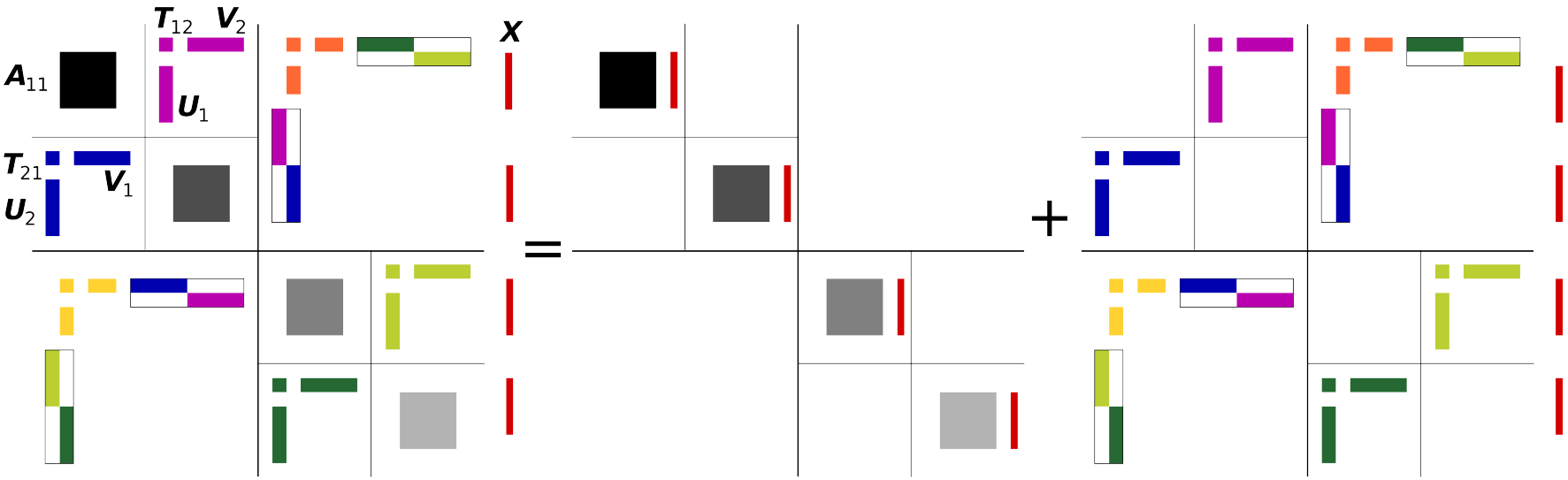
$$\bar{\boldsymbol{V}}_a = \mathcal{V}_a(\mathcal{H}_l(\boldsymbol{A})) = \begin{cases} \boldsymbol{V}_a & : l = 1 \\ \begin{bmatrix} \mathcal{V}_1(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) & \boldsymbol{0} \\ \boldsymbol{0} & \mathcal{V}_2(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) \end{bmatrix} \boldsymbol{V}_a & : l > 1 \end{cases}$$

HSS Matrix–Vector Multiplication

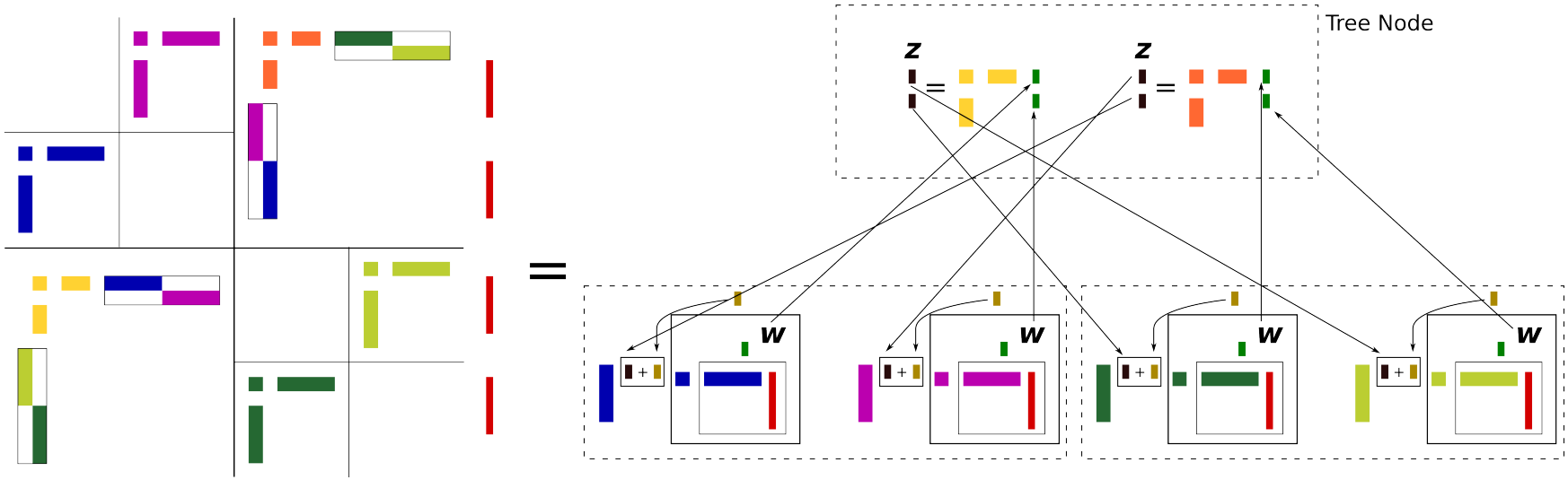
We now consider computing $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$

- With $\mathcal{H}_1(\boldsymbol{A})$ we would just compute
  $\boldsymbol{y}_1 = \boldsymbol{A}_{11}\boldsymbol{x}_1 + \boldsymbol{U}_1(\boldsymbol{T}_{12}(\boldsymbol{V}_2^T\boldsymbol{x}_2))$ and
  $\boldsymbol{y}_2 = \boldsymbol{A}_{22}\boldsymbol{x}_2 + \boldsymbol{U}_2(\boldsymbol{T}_{21}(\boldsymbol{V}_1^T\boldsymbol{x}_1))$
- For general $\mathcal{H}_l(\boldsymbol{A})$ perform up-sweep and down-sweep
  - up-sweep computes $\boldsymbol{w} = \begin{bmatrix} \bar{\boldsymbol{V}}_1^T\boldsymbol{x}_1 \\ \bar{\boldsymbol{V}}_2^T\boldsymbol{x}_2 \end{bmatrix}$ *at every tree node*
  - down-sweep computes a tree sum of $\begin{bmatrix} \bar{\boldsymbol{U}}_1\boldsymbol{T}_{12}\boldsymbol{w}_2 \\ \bar{\boldsymbol{U}}_2\boldsymbol{T}_{21}\boldsymbol{w}_1 \end{bmatrix}$

HSS Matrix Vector Product



HSS Matrix Vector Product



HSS Matrix–Vector Multiplication, Up-Sweep

The up-sweep is performed by using the nested structure of $\bar{\boldsymbol{V}}$

$$\boldsymbol{w} = \mathcal{W}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x}) = \begin{cases} \begin{bmatrix} \boldsymbol{V}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_2^T \end{bmatrix} \boldsymbol{x} & : l = 1 \\ \begin{bmatrix} \boldsymbol{V}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_2^T \end{bmatrix} \begin{bmatrix} \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{A}_{11}), \boldsymbol{x}_1) \\ \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{A}_{22}), \boldsymbol{x}_2) \end{bmatrix} & : l > 1 \end{cases}$$

HSS Matrix–Vector Multiplication, Down-Sweep

Use $\boldsymbol{w} = \mathcal{W}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x})$ from the root to the leaves to get

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{U}_1\boldsymbol{T}_{12}\boldsymbol{w}_2 \\ \boldsymbol{U}_2\boldsymbol{T}_{21}\boldsymbol{w}_1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{x}_1 \\ \boldsymbol{A}_{22}\boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{U}}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \bar{\boldsymbol{U}}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{0} \end{bmatrix} \boldsymbol{w} + \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{22} \end{bmatrix} \boldsymbol{x}$$

- using the nested structure of $\bar{\boldsymbol{U}}_a$ and $\boldsymbol{v} = \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{0} \end{bmatrix} \boldsymbol{w}$,

$$\boldsymbol{y}_a = \begin{bmatrix} \mathcal{U}_1(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) & \boldsymbol{0} \\ \boldsymbol{0} & \mathcal{U}_2(\mathcal{H}_{l-1}(\boldsymbol{A}_{aa})) \end{bmatrix} \boldsymbol{v}_a + \boldsymbol{A}_{aa}\boldsymbol{x}_a \quad \text{for} \quad a \in \{1, 2\}$$

- which gives the down-sweep recurrence

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{z} = \mathcal{Y}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x}, \boldsymbol{z}) = \begin{cases} \begin{bmatrix} \boldsymbol{U}_1\boldsymbol{q}_1 \\ \boldsymbol{U}_2\boldsymbol{q}_2 \end{bmatrix} + \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{x}_1 \\ \boldsymbol{A}_{22}\boldsymbol{x}_2 \end{bmatrix} & : l = 1 \\ \begin{bmatrix} \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{A}_{11}), \boldsymbol{x}_1, \boldsymbol{U}_1\boldsymbol{q}_1) \\ \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{A}_{22}), \boldsymbol{x}_2, \boldsymbol{U}_2\boldsymbol{q}_2) \end{bmatrix} & : l > 1 \end{cases}$$

where $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{0} \end{bmatrix} \boldsymbol{w} + \boldsymbol{z}$

Prefix Sum as HSS Matrix–Vector Multiplication

We can express the n-element prefix sum $y(i) = \sum_{j=1}^{i-1} x(j)$ as

$$\boldsymbol{y} = \boldsymbol{L}\boldsymbol{x} \quad \text{where} \quad \boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{0} \\ \boldsymbol{L}_{21} & \boldsymbol{L_{22}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 1 & \cdots & 1 & 0 \end{bmatrix}$$

- $\boldsymbol{L}$ is an $\mathcal{H}$-matrix since $\boldsymbol{L}_{21} = \mathbf{1_n}\mathbf{1}_{\boldsymbol{n}}^T = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}$
- $\boldsymbol{L}$ also has rank-1 HSS structure, in particular

$$\mathcal{H}_l(\boldsymbol{L}) = \begin{cases} \left\{ \mathbf{1_2}, \mathbf{1_2}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} 0 \end{bmatrix} \right\} & : l = 1 \\ \left\{ \mathbf{1_4}, \mathbf{1_4}, \begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} 1 \end{bmatrix}, \mathcal{H}_{l-1}(\boldsymbol{L}_{11}), \mathcal{H}_{l-1}(\boldsymbol{L}_{22}) \right\} & : l > 1 \end{cases}$$

so each $\boldsymbol{U}, \boldsymbol{V}, \bar{\boldsymbol{U}}, \bar{\boldsymbol{V}}$ is a vector of 1s, $\boldsymbol{T}_{12} = \begin{bmatrix} 0 \end{bmatrix}$ and $\boldsymbol{T}_{21} = \begin{bmatrix} 1 \end{bmatrix}$

Prefix Sum HSS Up-Sweep

We can use the HSS structure of $\boldsymbol{L}$ to compute the prefix sum of $\boldsymbol{x}$

- recall that the up-sweep recurrence has the general form

$$\boldsymbol{w} = \mathcal{W}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x}) = \begin{cases} \begin{bmatrix} \boldsymbol{V}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_2^T \end{bmatrix} \boldsymbol{x} & : l = 1 \\ \begin{bmatrix} \boldsymbol{V}_1^T & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_2^T \end{bmatrix} \begin{bmatrix} \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{A}_{11}), \boldsymbol{x}_1) \\ \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{A}_{22}), \boldsymbol{x}_2) \end{bmatrix} & : l > 1 \end{cases}$$

- for the prefix sum this becomes

$$\boldsymbol{w} = \mathcal{W}(\mathcal{H}_l(\boldsymbol{L}), \boldsymbol{x}) = \begin{cases} \boldsymbol{x} & : l = 1 \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{L}_{11}), \boldsymbol{x}_1) \\ \mathcal{W}(\mathcal{H}_{l-1}(\boldsymbol{L}_{22}), \boldsymbol{x}_2) \end{bmatrix} & : l > 1 \end{cases}$$

- so the up-sweep computes $\boldsymbol{w} = \begin{bmatrix} \mathcal{S}(\boldsymbol{x_1}) \\ \mathcal{S}(\boldsymbol{x_2}) \end{bmatrix}$ where $\mathcal{S}(\boldsymbol{y}) = \sum_i y_i$

Prefix Sum HSS Down-Sweep

The down-sweep has the general structure

$$\boldsymbol{y} = \mathcal{Y}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x}, \boldsymbol{z}) = \begin{cases} \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{U}_2 \end{bmatrix} \boldsymbol{q} + \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{22} \end{bmatrix} \boldsymbol{x} & : l = 1 \\ \begin{bmatrix} \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{A}_{11}), \boldsymbol{x}_1, \boldsymbol{U}_1 \boldsymbol{q}_1) \\ \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{A}_{22}), \boldsymbol{x}_2, \boldsymbol{U}_2 \boldsymbol{q}_2) \end{bmatrix} & : l > 1 \end{cases}$$

where $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{0} \end{bmatrix} \mathcal{W}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x}) + \boldsymbol{z}$, for the prefix sum

- $\begin{bmatrix} \boldsymbol{0} & \boldsymbol{T}_{12} \\ \boldsymbol{T}_{21} & \boldsymbol{0} \end{bmatrix} \mathcal{W}(\mathcal{H}_l(\boldsymbol{L}), \boldsymbol{x}) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \mathcal{S}(\boldsymbol{x}_1) \\ \mathcal{S}(\boldsymbol{x}_2) \end{bmatrix} = \begin{bmatrix} 0 \\ \mathcal{S}(\boldsymbol{x}_1) \end{bmatrix} = \boldsymbol{q} - \boldsymbol{z}$

$$\boldsymbol{y} = \mathcal{Y}(\mathcal{H}_l(\boldsymbol{L}), \boldsymbol{x}, \boldsymbol{z}) = \begin{cases} \begin{bmatrix} z_1 \\ x_1 + z_2 \end{bmatrix} & : l = 1 \\ \begin{bmatrix} \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{L}_{11}), \boldsymbol{x}_1, \boldsymbol{1_2} z_1) \\ \mathcal{Y}(\mathcal{H}_{l-1}(\boldsymbol{L}_{22}), \boldsymbol{x}_2, \boldsymbol{1_2}(\mathcal{S}(\boldsymbol{x}_1) + z_2)) \end{bmatrix} & : l > 1 \end{cases}$$

- Initially the prefix $\boldsymbol{z} = \boldsymbol{0}$ and it will always be the case that $z_1 = z_2$

Cost of HSS Matrix–Vector Multiplication

The down-sweep and the up-sweep perform small dense matrix–vector multiplications at each recursive step

- Lets assume k is the dimension of the leaf blocks and the rank at each level (number of columns in each $\boldsymbol{U}_a$, $\boldsymbol{V}_a$)
- The work for both the down-sweep and up-sweep is

$$Q(n,k) = 2Q(n/2,k) + O(k^2 \cdot \gamma), \quad Q(k,k) = O(k^2 \cdot \gamma)$$
$$Q(n,k) = O(nk \cdot \gamma)$$

- The depth of the algorithm scales as $D = \Theta(\log(n))$ for fixed k

Parallel of HSS Matrix–Vector Multiplication

- If we assign each tree node to a single processor for the first $\log_2(p)$ levels, and execute a different leaf subtree with a different processor

$$
\begin{aligned}
T_1(n,k) &= (nk \cdot \gamma) \\
T_p(n,k) &= T_{p/2}(n/2,k) + O(k^2 \cdot \gamma + k \cdot \beta + \alpha) \\
&= O((nk/p + k^2 \log(p)) \cdot \gamma + k \log(p) \cdot \beta + \log(p) \cdot \alpha)
\end{aligned}
$$

Synchronization-Efficient HSS Multiplication

- The leaf subtrees can be computed independently

$$T_p^{\text{leaf}-\text{subtrees}}(n,k) = O(nk/p \cdot \gamma + k \cdot \beta + \alpha)$$

- Consider up-sweep and down-sweep with $\log_2(p)$ levels
- Executing the root subtree sequentially takes time

$$T_p^{\text{root}-\text{subtree}}(pk,k) = O(pk^2 \cdot \gamma + pk \cdot \beta + \alpha)$$

- Instead have p^r $(r < 1)$ processors compute subtrees with p^{1-r} leaves, then recurse on the p^r roots

$$\begin{aligned} T_p^{\text{rec}-\text{tree}}(pk,k) &= T_{p^r}^{\text{rec}-\text{tree}}(p^r k,k) + O(p^{1-r}k^2 \cdot \gamma + p^{1-r}k \cdot \beta + \alpha) \\ &= O(p^{1-r}k^2 \cdot \gamma + p^{1-r}k \cdot \beta + \log_{1/r}(\log(p)) \cdot \alpha) \end{aligned}$$

Synchronization-Efficient HSS Multiplication

- Consider the top tree with p leaves (leaf subtrees)
- Assign each processor a unique path from a leaf to the root
- Given $\boldsymbol{w} = \mathcal{W}(\mathcal{H}_l(\boldsymbol{A}), \boldsymbol{x})$ at every node each processor can compute a down-sweep path in the subtree independently
- For the up-sweep, realize that the tree applies a linear transformation, so can sum the results computed in each path
- For each tree node, there is a contribution from every processor assigned a leaf of the subtree of the node
- Therefore, there are $p-1$ sums of a total of $O(p\log(p))$ contributions, for a total of $O(kp\log(p))$ elements
- Do these with $\min(p, k\log_2(p))$ processors, each obtaining $\max(p, k\log_2(p))$ contributions, so

$$T_p^{\mathrm{root-paths}}(k) = O(k^2\log(p)\cdot\gamma + (k\log(p)+p)\cdot\beta + \log(p)\cdot\alpha)$$

HSS Multiplication by Multiple Vectors

Consider multiplication $\boldsymbol{C} = \boldsymbol{AB}$ where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is HSS and $\boldsymbol{B} \in \mathbb{R}^{n \times b}$

- lets consider the case that $p \leq b \ll n$
- if we assign each processor all of $\boldsymbol{A}$, each can compute a column of $\boldsymbol{C}$ simultaneously
- this requires a prohibitive amount of memory usage
  - perform leaf-level multiplications, processing n/p rows of $\boldsymbol{B}$ with each processor (call intermediate $\bar{\boldsymbol{C}}$)
  - transpose $\bar{\boldsymbol{C}}$ and apply $\log_2(p)$ root levels of HSS tree to columns of $\bar{\boldsymbol{C}}$ independently
- this algorithm requires replication only of the root $O(\log(p))$ levels of the HSS tree, $O(pb)$ data
- for large k or larger p different algorithms may be desirable

References

- Michiel E. Hochstenbach, A Jacobi–Davidson Type SVD Method, *SIAM Journal on Scientific Computing* 2001 23:2, 606-628
- Chan, T. F. (1987). Rank revealing QR factorizations. *Linear algebra and its applications*, 88, 67-82.
- Businger, P., and Golub, G. H. (1965). Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7(3), 269-276.

References

- Quintana-Ortí, G., Sun, X., and Bischof, C. H. (1998). A BLAS-3 version of the QR factorization with column pivoting. *SIAM Journal on Scientific Computin*g, 19(5), 1486-1494.
- Demmel, J.W., Grigori, L., Gu, M. and Xiang, H., 2015. Communication avoiding rank revealing QR factorization with column pivoting. *SIAM Journal on Matrix Analysis and Applications*, 36(1), pp.55-89.
- Bischof, C.H., 1991. A parallel QR factorization algorithm with controlled local pivoting. *SIAM Journal on Scientific and Statistical Computing*, 12(1), pp.36-57.
- Halko, N., Martinsson, P.G. and Tropp, J.A., 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2), pp.217-288.