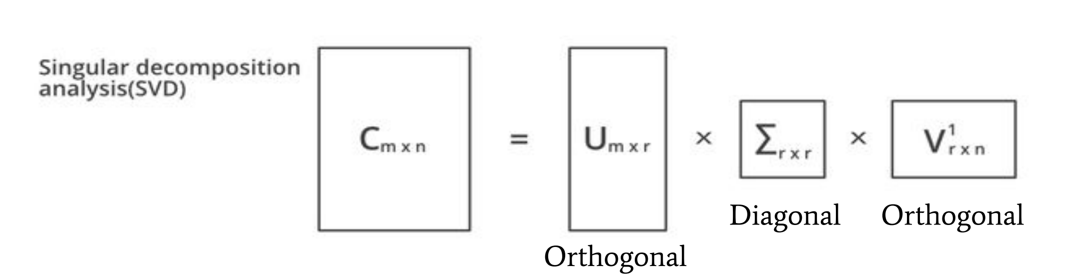
Team 19: Parallelization of SVD
Plan for Parallel Design

Sibi Raja, Matthew Villaroman, Eliel Dushime, Bowen Zhu

Quick Recap

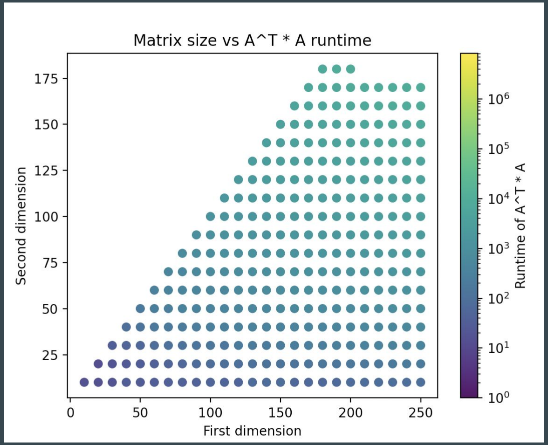
- Singular Value Decomposition: the factorization of a matrix into an equivalent product of 3 smaller matrices
- Main functions in our SVD algorithm: vector/matrix operations, power method, singular value calculation, Gauss-Jordan elimination, back substitution

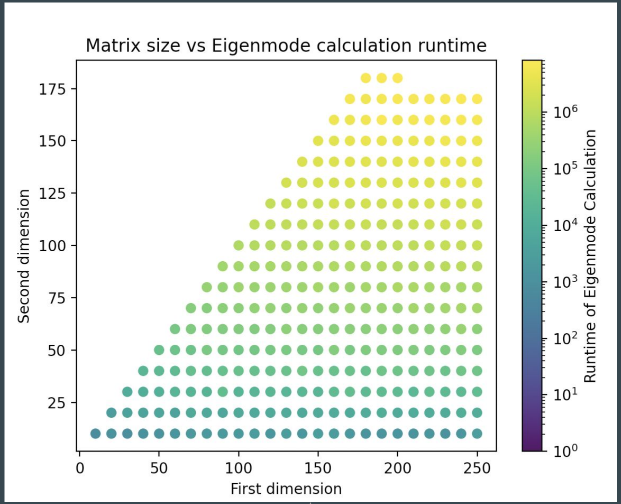


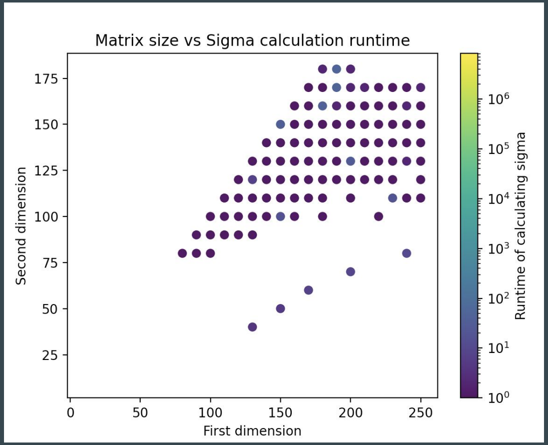
GeeksforGeeks

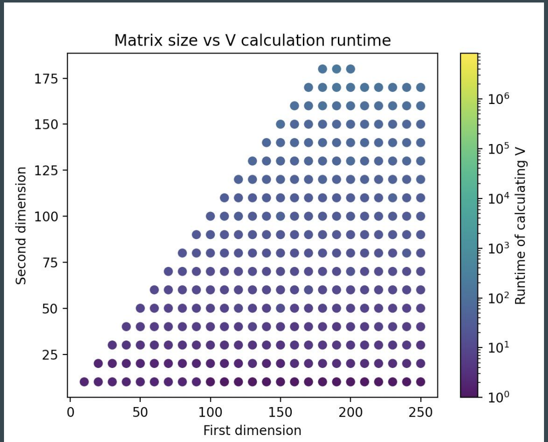
```
Dimensions: 5 x 4
U:
[[0.282081, -0.317569, 0.273586, 0.28105, 0.81593],
[0.2935, 0.206195, -0.512016, 0.771919, -0.115422],
[0.328003, 0.565869, 0.712557, 0.168352, -0.190068],
[0.568179, -0.650964, 0.1324, -0.0268382, -0.484942],
[0.635511, 0.335665, -0.37111, -0.544142, 0.222804]]
Sigma:
[[2.18839, 0, 0, 0],
[0, 0.733674, 0, 0],
[0, 0, 0.429775, 0],
[0, 0, 0, 0.267276],
[0, 0, 0, 0]]
V:
[[0.435381, 0.597634, -0.508387, -0.441384],
[0.568319, -0.376993, -0.45417, 0.573254],
[0.528335, -0.510377, 0.354684, -0.578425],
[0.456419, 0.490128, 0.6399, 0.376808]]
Original A:
[[0.0365859, 0.428322, 0.443311, 0.271098],
[0.390859, 0.526207, 0.0647482, 0.304231],
[0.385083, 0.138134, 0.249939, 0.744018],
[0.230162, 0.85674, 0.925015, 0.367136],
[0.897961, 0.68661, 0.636644, 0.598605]]
Reconstructed:
[[0.0365859, 0.428322, 0.443311, 0.271098],
[0.390859, 0.526207, 0.0647482, 0.304231],
[0.385083, 0.138134, 0.249939, 0.744018],
[0.230162, 0.85674, 0.925015, 0.367136],
[0.897961, 0.68661, 0.636644, 0.598605]]
```

Profiling - SVD Calculation



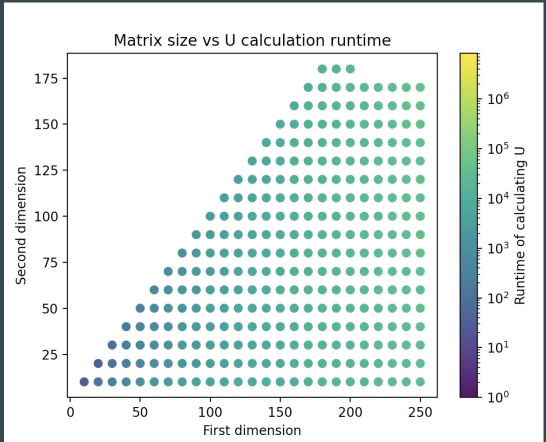




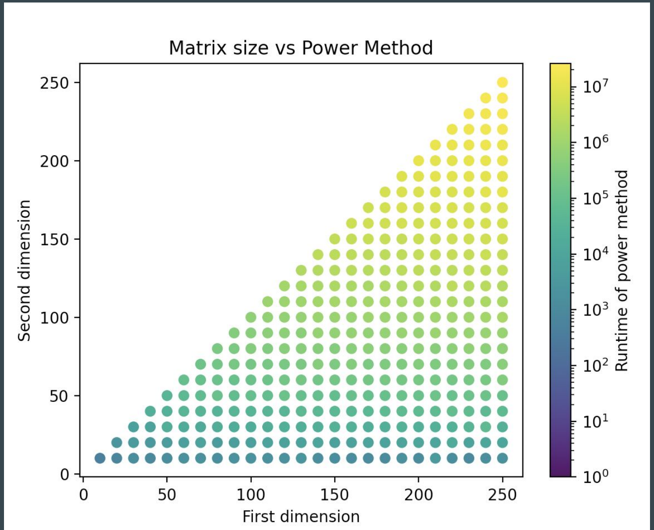


Bottlenecks

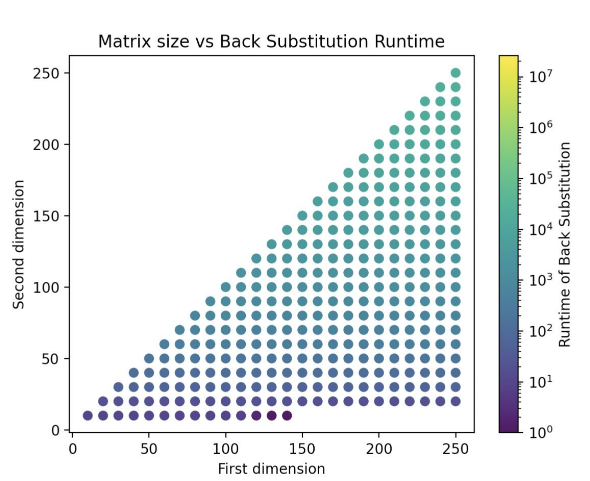
Note: the missing dots represent runtime of 0, which is not possible to display in a log-scale

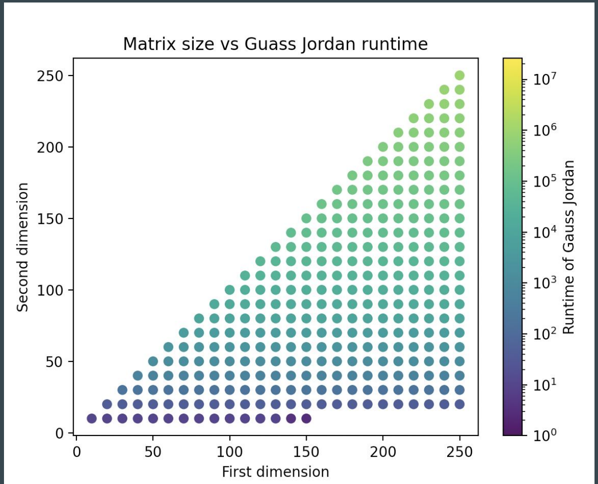


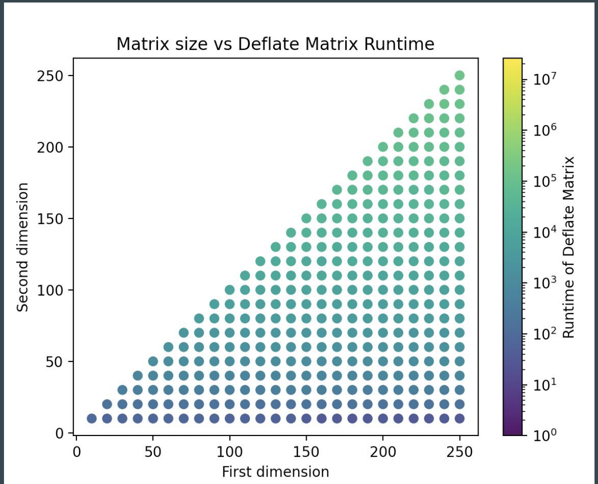
Profiling - Eigendecomposition calculation



Note: the missing dots represent runtime of 0, which is not possible to display in a log-scale







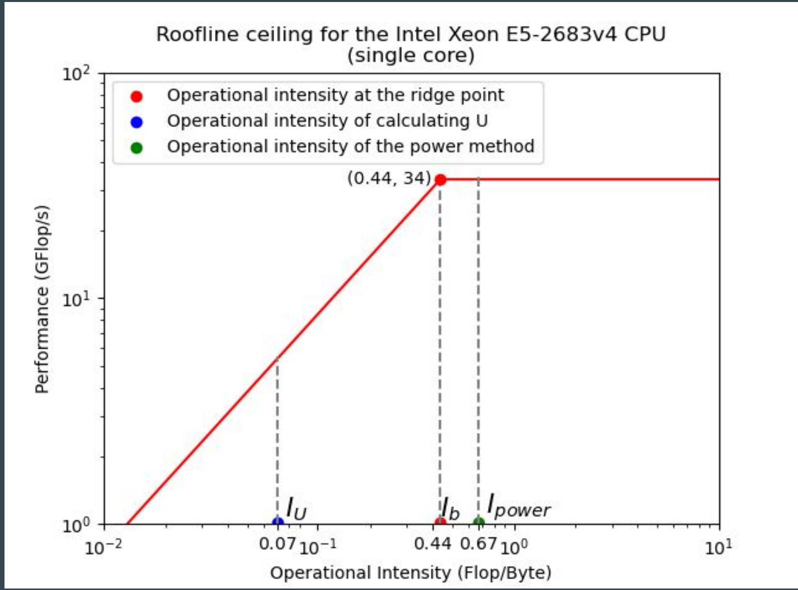
Bottleneck

Bottlenecks & Roofline Analysis

$$\pi = f \times n_c \times l_s \times \phi = 33.6 \text{ Gflop/s}$$

$$\beta = 2 \times f_{DDR} \times c \times w = 76.8 \text{ GB/s}$$

- Bottlenecks:
  - Power method function
    - k is the ratio between the largest and smallest eigenvalues
  - Calculate U function
- Operational intensities:
    - Ridge point: 0.44
    - Power method: $(2kM^2 + 2M^2) / (3kM^2 + 2M^2) = 0.67 > 0.44$
    - U: $(8M^3 + 3M^2 + 2M) / (104M^3 + 72M^2 + 24M) = 0.07 < 0.44$
- Conclusions:
  - Power method is compute bound
  - Calculating U is memory bound



Proposed Parallel Design

- Based on our analysis, CalculateU and PowerMethod are the main bottlenecks
- Sequential calculation of U:
  - Memory bandwidth-limited and performs at a very low Flop rate (memory-bound)
  - Hitting more often DRAM data movements
  - Minimal use of cache lines and little memory reuse.
  - *Intended Parallel design:* Make use of spatial locality and cache blocking to improve this bound.
- Gram-Schmidt implementation (inside Calculate U function):
  - Many independent vector multiplications
  - *Intended Parallel design:* Use of Data-level parallelism to assign calculations to separate threads.
- Sequential Power method (and deflation):
  - Compute-bound, mainly due to the iterative nature of calculating eigenvalue-eigenvector pairs in descending order
  - *Intended Parallel design:* Many flavors of the power method on distinct threads to calculate eigenvalues asynchronously
  - Data-level parallelism: distributing the eigenvalue calculation to threads with the similar instructions.
- Sequential Matrix Operations (GEMM)
  - *Intended Parallel design:* Split matrix operations across simultaneous threads

Parallel Code Implementation

- Parallel Matrix Operations (i.e. GEMM)
  - ❑ Distribute the different sub-operations on rows/columns across different threads
  - ❑ Synchronize after collectively completing the entire operation
  - ❑ This can speedup the calculation of U → calculating U consists of several matrix operations that can be parallelized
    - ❑ We will conduct further investigation to ensure this parallelization is worth the overhead
- Parallel Gram-Schmidt implementation (inside CalculateU function):
  - ❑ Independent vector projections can be calculated by separate threads, and the threads will reduce to the final resultant vector for normalization
  - ❑ Synchronization points before completing a new orthogonal vector
- Parallel Power method and deflation:
  - ❑ Inverse power method (smallest eigenvalue) and Shifted inverse power method (nearest eigenvalue to a guess)
    - ❑ Allows multiple eigenvalues to be calculated simultaneously (descending from the largest, ascending from the smallest, guesses near the middle of the ordering)
  - ❑ Eigenvectors can be calculated by idle threads or threads finished with load-imbalanced power method iterations
  - ❑ After or during calculation of eigenvalues, we will require thread synchronization to account for all eigenvalues