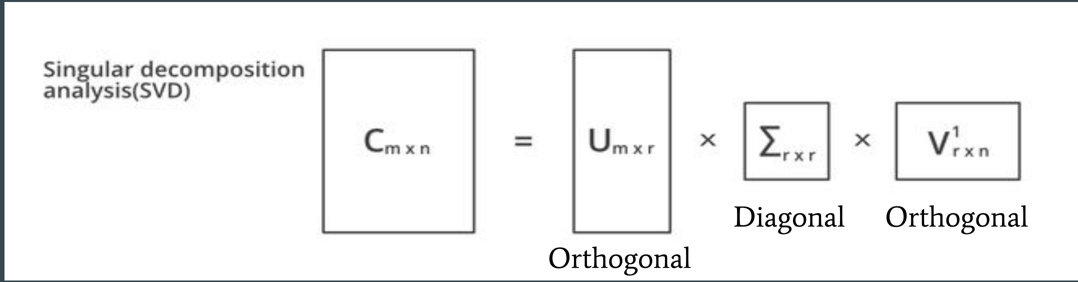
Team 19: Parallelization of SVD
Final Presentation - CS205 Spring 2023

Sibi Raja, Matthew Villaroman, Eliel Dushime, Bowen Zhu

Background Information

- Singular Value Decomposition: the factorization of a matrix into an equivalent product of 3 smaller matrices
  - Computationally expensive process as matrices become larger
- Main functions in our SVD algorithm: vector/matrix operations, power method, singular value calculation, Gauss-Jordan elimination, back substitution
- *J. SairaBanu, Rajasekhara Babu and Reeta Pandey*: found using a parallelized SVD algorithm allows for faster runtime of 10-16% for image compression



GeeksforGeeks

SVD Algorithm

Sequential Description

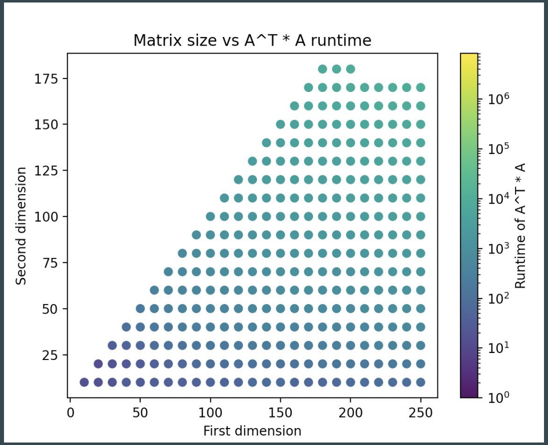
Required vector/matrix operations:

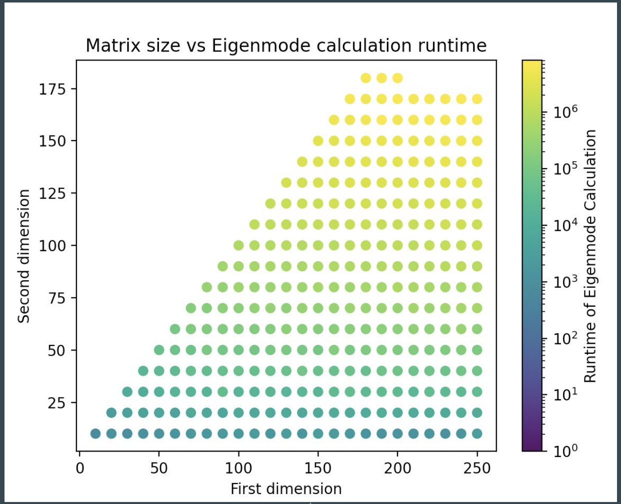
gemm, gemv, dot, normalize, scalar-vector multiplication, transpose, matrix subtraction, vector to matrix conversion

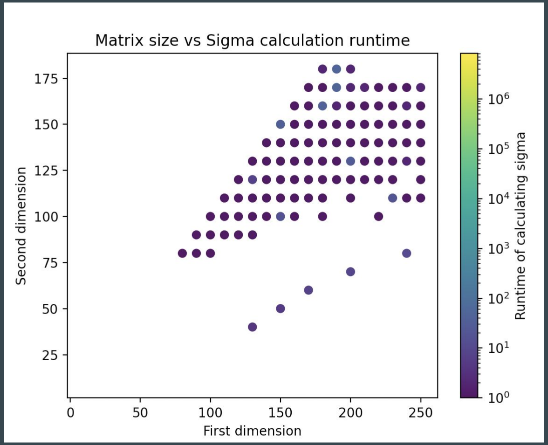
SVD can be split into 5 main parts:

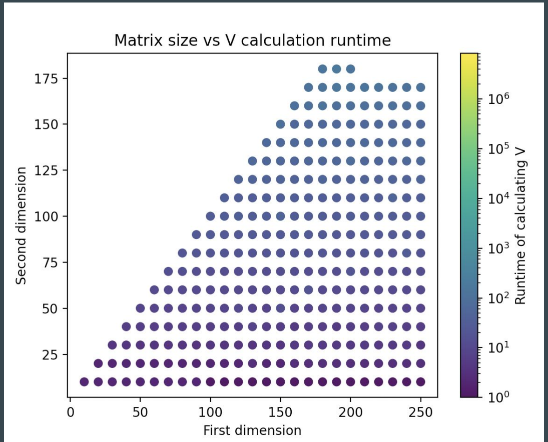
- Calculate the product A^TA
  - Transpose and gemm
- Find eigenvalues and eigenvectors of A^TA
  - Gemv, normalize, dot, Matrix_subtraction,scalar-vector multiplication, gemm
- Construct Σ
- Construct V
  - Transpose
- Construct U
  - Gemm, transpose, scalar-vector multiplication, dot, normalize

Identifying Initial Bottlenecks - Overall SVD Calculation



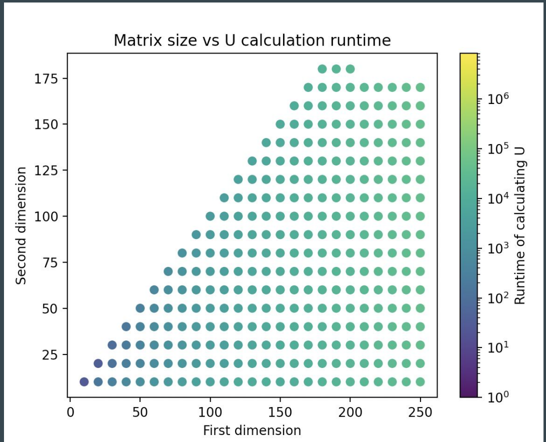




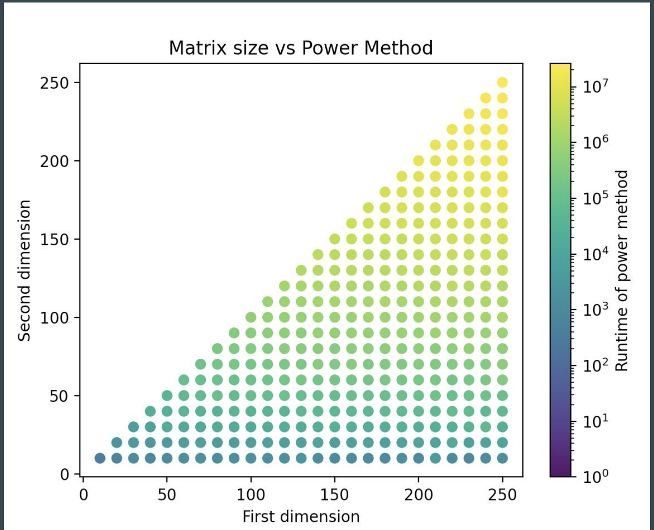


Bottlenecks

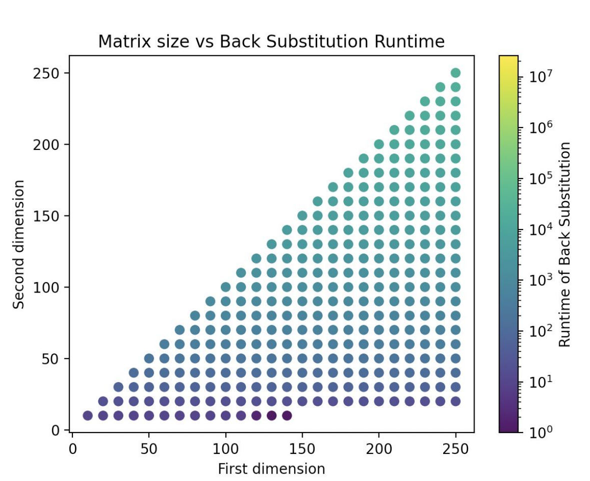
Note: the missing dots represent runtime of 0, which is not possible to display in a log-scale



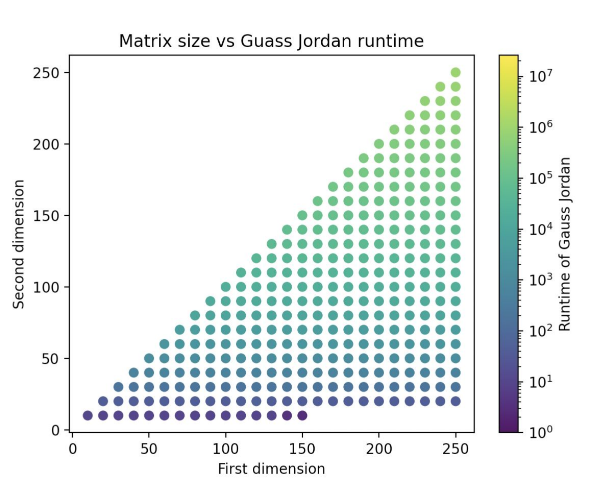
Deeper Bottlenecks within Eigendecomposition

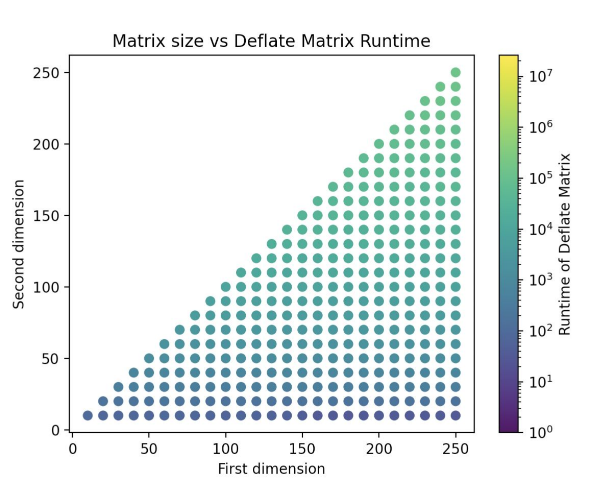


Note: the missing dots represent runtime of 0, which is not possible to display in a log-scale



Bottleneck





Areas of Focus: Calculate U and Power Method

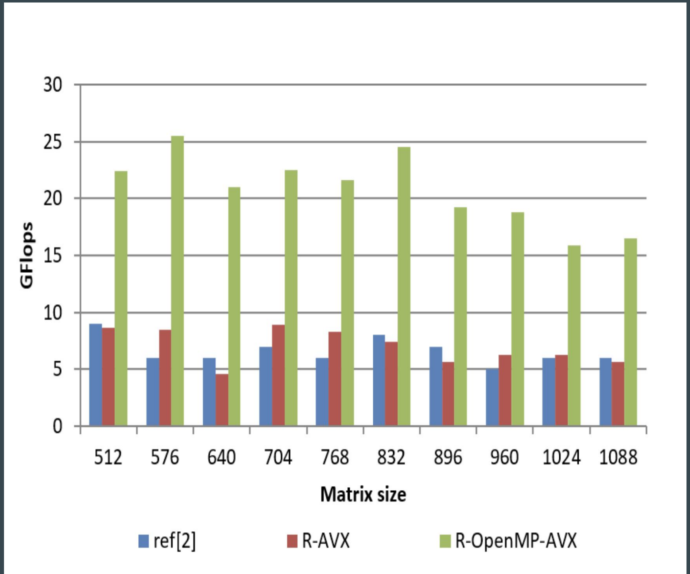
Parallelization of SVD Algorithm

- GEMV and Dot:
  - ➔ OpenMP: outer loop of matrix mult to speed up iterations
  - ➔ SIMD/AVX: to limit bottleneck imposed by FP operations

- Normalize, scalar vector multiplication, transpose:
  - ➔ OpenMP and SIMD/AVX: division, mult, and matrix traversal for normalize, scalar vector mult, and transpose

- GEMM: dominated by floating-point operations and iteration loops
  - ➔ SIMD/AVX, Tiling, OpenMP
  - ➔ Loop unrolling, cache blocking & other cache-aware methods

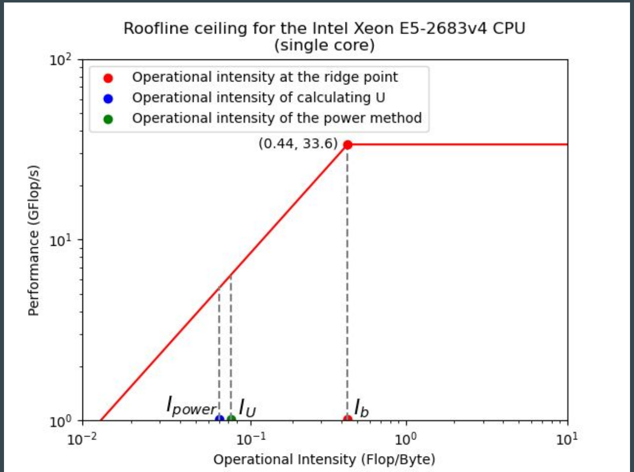
- Power Method, Calculate U, final SVD
  - ➔ Power Method and Calculate gain significant speedup from the smaller functions

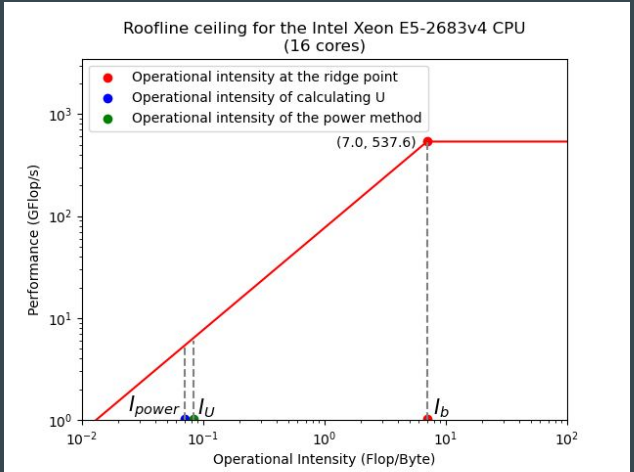


Akoushideh, A., & Shahbahrami, A. (2022, October 11). Performance Evaluation of Matrix-Matrix Multiplication using Parallel Programming Models on CPU Platforms (Version 1) [Preprint]. Research Square. https://doi.org/10.21203/rs.3.rs-2135830/v1

Roofline Analysis

- Nominal peak memory performance: 76.8 GB/s
- Nominal peak arithmetic performance:
  - Single core: 33.6 Gflop/s
  - 16 cores: 537.6 Gflop/s
- Ridge point:
  - 0.44 flop/byte
  - 7.0 flop/byte
- Operational intensities:
  - Power method: 0.076 flop/byte
  - Calculate U: 0.083 flop/byte





Overall Performance Benchmarks

Strong Scaling Analysis

Number of cores	Total Runtime (ms)	Power Method Runtime (ms)	Calculate U Method Runtime (ms)
1	2.01951e+07	1.89196e+07	36947
2	2.25662e+07	2.12887e+07	36257
4	2.54485e+07	2.40804e+07	36571
8	3.70329e+07	3.55985e+07	36020
16	5.51495e+07	5.37376e+07	33411

Table 1: Runtime (microseconds) with respect to the number of cores

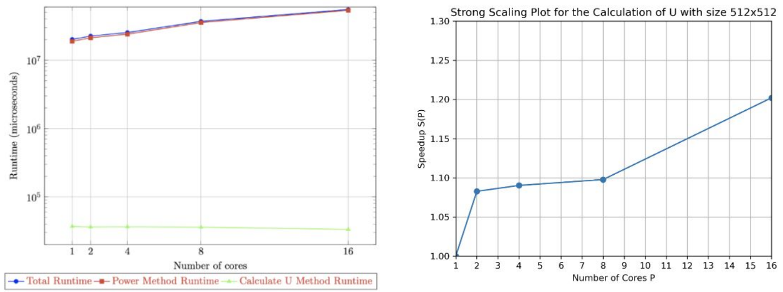
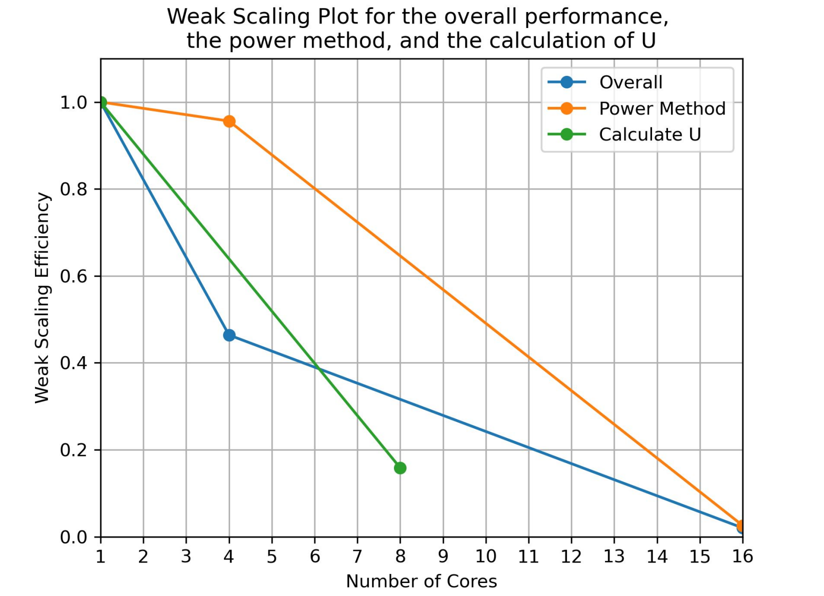


Figure 4: Plot on left: table 1 visualized, plot on right: speedup of Calculate U - Strong Scaling

Weak Scaling Analysis



Takeaways and Future Work

- In general, parallelization of SVD offers up speedups of the SVD operations
- Future work would focus on:
  - ➔ Better improvements of power method
  - ➔ Implementation and testing of more parallelization techniques of GEMM
  - ➔ GPU acceleration with CUDA or similar
  - ➔ Considerations of other SVD sub-operations more amenable to parallelization

Thank you!