M5470: Introduction to Parallel Scientific Computing

Homework 1

Dheeraj M
EE21BTECH11015

Question 1

Methodology

- The experiment involves running the code with the following configurations:
  Format Flag = 0 (ASCII): Generates a human-readable text file.
  Format Flag = 1 (Binary): Generates a machine-readable binary file.

To estimate the memory required to store the array, the following calculation was used:

Size in memory = $8 \times n^2 \; Bytes$

Where:

- 8 bytes represent the size of a double-precision floating-point number.
- n^2 represents the total number of elements in the array.

Results

Format Flag	File Name	File Size on Disk	Estimated Memory Usage
ASCII (0)	array_004000_asc.out	332 MB	122 MB
Binary (1)	array_004000_bin.out	122 MB	122 MB

Discussion

The observed file sizes demonstrate the following key points:

ASCII Format (Text-Based):

- Larger file size (332 MB) compared to memory usage due to additional characters (spaces, line breaks, number formatting).
- Text representation of floating-point values introduces overhead from character encoding and precision representation.

Binary Format (Machine-Readable):

- File size is closer to memory usage (122 MB), as binary storage preserves raw floating-point values without extra formatting overhead.
- Binary format is more space-efficient and faster for read/write operations.

Suitability for Large Data:

- ASCII files are useful for human readability and debugging but become inefficient for large datasets.
- Binary files are more efficient in terms of storage and performance, making them preferable for scientific computations and large-scale data storage.

Conclusion

The results indicate that the binary format is more space-efficient and suitable for storing large arrays, as it minimises overhead and closely matches the expected memory size. While ASCII format provides readability, it introduces significant storage overhead due to text formatting.

Question 2

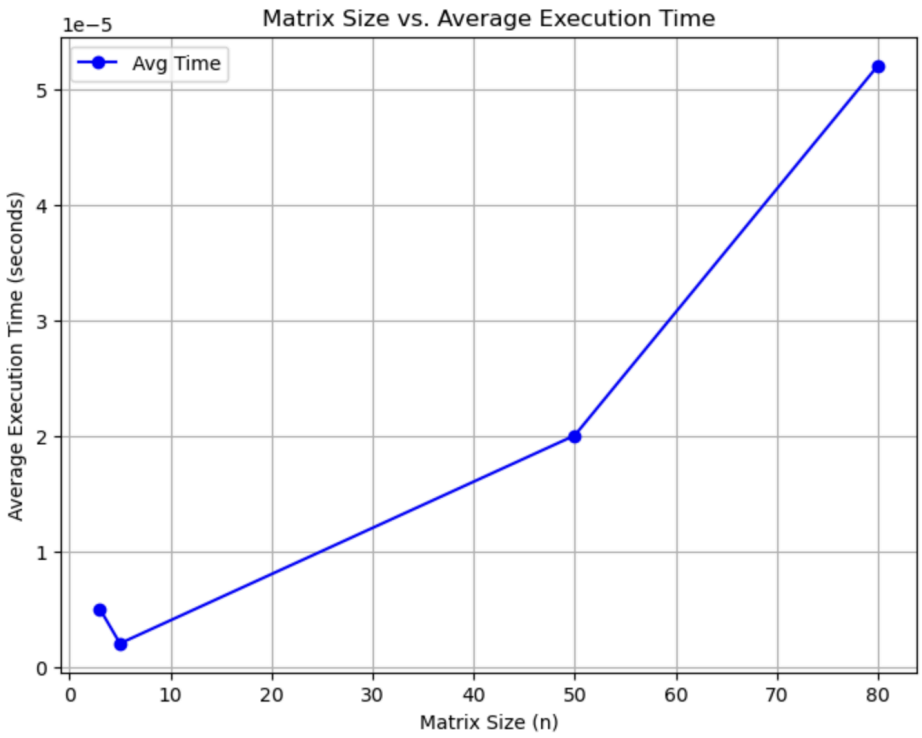
Methodology

The implemented code reads an integer 'n' from an input file to determine the size of the matrix. It then reads an n x n matrix and corresponding vectors of size 'n' each. For each vector, the code checks if it is an eigenvector by calculating the matrix-vector product and comparing it with the scaled version of the vector.

Execution time is measured using the `clock()` function with high precision, and the average time for processing all vectors associated with a matrix is computed and logged in a single file. The workflow includes reading matrix and vector files, processing them, and writing the results.

Execution Time Analysis

Graph representing execution time vs. matrix size will be included here.



Results

Vector File	Eigenvector Status	Eigenvalue
vec_000003_000001.in	Yes	-6.000000
vec_000003_000002.in	Yes	-6.000000
vec_000003_000003.in	Yes	-1.000000
vec_000003_000004.in	Not an eigenvector	-
vec_000005_000001.in	Yes	0.268098
vec_000005_000002.in	Not an eigenvector	-
vec_000005_000003.in	Yes	0.986875
vec_000005_000004.in	Yes	1.399039
vec_000050_000001.in	Not an eigenvector	-
vec_000050_000002.in	Yes	0.479628
vec_000050_000003.in	Yes	1.337887
vec_000050_000004.in	Not an eigenvector	-
vec_000080_000001.in	Yes	0.333018
vec_000080_000002.in	Yes	0.493142
vec_000080_000003.in	Yes	0.939275