HOMEWORK -1

Name: Shushhma Dewie Koppireddy

Id:02190512

**PROBLEM - 5**

**Dot Product of Two Vectors ( in Python)**

import numpy as np

import time

n = 10000

a = np.random.rand(n)

b = np.random.rand(n)

c = 0

start\_loop = time.time()

for i in range(n):

c += a[i] \* b[i]

time\_loop = time.time() - start\_loop

start\_vec = time.time()

cc = np.dot(a, b) # Vectorized dot product using NumPy

time\_vec = time.time() - start\_vec

print(f"Dot Product (loop): {c}")

print(f"Dot Product (vectorized): {cc}")

print(f"Difference between results: {abs(c - cc)}")

speedup = time\_loop / time\_vec

print(f"Time taken by loop: {time\_loop} seconds")

print(f"Time taken by vectorization: {time\_vec} seconds")

print(f"Speedup: {speedup}")

**Matrix vector – python**

import numpy as np

import time

n = 300

A = np.random.rand(n, n)

B = np.random.rand(n, n)

C = np.zeros((n, n))

CC = np.zeros((n, n))

start = time.perf\_counter()

for i in range(n):

for j in range(n):

for k in range(n):

C[i, j] += A[i, k] \* B[k, j]

timeloop = time.perf\_counter() - start

print(f"Time for loop-based computation: {timeloop:.6f} seconds")

start = time.perf\_counter()

for j in range(n):

CC[:, j] = np.dot(A, B[:, j])

timeloopvec = time.perf\_counter() - start

print(f"Time for partial vectorized computation: {timeloopvec:.6f} seconds")

start = time.perf\_counter()

CCC = np.dot(A, B)

timevec = time.perf\_counter() - start

print(f"Time for fully vectorized computation: {timevec:.6f} seconds")

norm1 = np.linalg.norm(C - CC)

norm2 = np.linalg.norm(C - CCC)

print(f"Norm between C and CC: {norm1:.6f}")

print(f"Norm between C and CCC: {norm2:.6f}")

speedup = timeloop / timeloopvec if timeloopvec > 0 else float('inf')

speedup2 = timeloop / timevec if timevec > 0 else float('inf')

speedup3 = timeloopvec / timevec if timevec > 0 else float('inf')

print(f"Speedup of loop-based vs partial vectorization: {speedup:.2f}")

print(f"Speedup of loop-based vs full vectorization: {speedup2:.2f}")

print(f"Speedup of partial vectorization vs full vectorization: {speedup3:.2f}")

**Matrix-Matrix product ( In python)**

import numpy as np

import time

n = 300

A = np.random.rand(n, n)

B = np.random.rand(n, n)

C = np.zeros((n, n))

CC = np.zeros((n, n))

start = time.perf\_counter()

for i in range(n):

for j in range(n):

for k in range(n):

C[i, j] += A[i, k] \* B[k, j]

timeloop = time.perf\_counter() - start

print(f"Time for loop-based computation: {timeloop:.6f} seconds")

start = time.perf\_counter()

for i in range(n):

CC[:, i] = np.dot(A, B[:, i])

timeloopvec = time.perf\_counter() - start

print(f"Time for partial vectorized computation: {timeloopvec:.6f} seconds")

start = time.perf\_counter()

CCC = np.dot(A, B)

timevec = time.perf\_counter() - start

print(f"Time for fully vectorized computation: {timevec:.6f} seconds")

norm1 = np.linalg.norm(C - CCC)

norm2 = np.linalg.norm(CC - CCC)

print(f"Norm between C and CC: {norm1:.6f}")

print(f"Norm between C and CCC: {norm2:.6f}")

speedup = timeloop / timeloopvec if timeloopvec > 0 else float('inf')

speedup2 = timeloop / timevec if timevec > 0 else float('inf')

speedup3 = timeloopvec / timevec if timevec > 0 else float('inf')

print(f"Speedup of loop-based vs partial vectorization: {speedup:.2f}")

print(f"Speedup of loop-based vs full vectorization: {speedup2:.2f}")

print(f"Speedup of partial vectorization vs full vectorization: {speedup3:.2f}")

**Dot Product of Two Vectors ( in C++)**

#include <iostream>

#include <vector>

#include <chrono>

#include <cmath>

int main() {

int n = 10000;

std::vector<double> a(n), b(n);

for (int i = 0; i<n; i++) {

a[i] = static\_cast<double>(rand()) / RAND\_MAX;

b[i] = static\_cast<double>(rand()) / RAND\_MAX;

}

double c = 6.0;

auto start\_loop = std::chrono: :high\_resolution\_clock: :now();

for (int i = @; i<n; i++) {

c += afi] \* b[i];

auto end\_loop = std::chrono: :high\_resolution\_clock: :now();

std::chrono::duration<double> timeloop = end\_loop - start\_loop;

auto start\_vec = std::chrono::high\_resolution\_clock: :now();

double cc = 0.0;

for (int i = @; i <n; i++) {

cc += afi] \* b[i];

}

auto end\_vec = std::chrono::high\_resolution\_clock: :now();

std::chrono::duration<double> timevec = end\_vec - start\_vec;

std::cout << "Dot Product (loop): " << c << std::endl;

std::cout << "Dot Product (vectorized): " << cc << std::endl;

std::cout << "Difference between results: " << std::fabs(c - cc) << std::endl;

double speedup = timeloop.count() / timevec.count();

std::cout << "Time taken by loop: " << timeloop.count() << " seconds" << std::endl;

std::cout << “Time taken by vectorization: " << timevec.count() << " seconds” << std::endl;

std::cout << "Speedup: " << speedup << std::endl;

return 0;

}

**Matrix vector (C++)**

#include <iostream>

#include <vector>

#include <chrono>

#include <cmath>

int main() {

const int n = 100; // Size of the matrix and vector

std::vector<std::vector<double>> A(n, std::vector<double>(n)); // Matrix A

std::vector<double> x(n); // Vector x

std::vector<double> b(n, 0.0); // Result of loop-based multiplication

std::vector<double> bb(n, 0.0); // Result of partial vectorized multiplication

std::vector<double> bbb(n, 0.0); // Result of fully vectorized multiplication

for (int i = 0; i < n; ++i) {

x[i] = static\_cast<double>(rand()) / RAND\_MAX;

for (int j = 0; j < n; ++j) {

A[i][j] = static\_cast<double>(rand()) / RAND\_MAX;

}

}

auto start = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

b[i] += A[i][j] \* x[j];

}

}

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timeloop = end - start;

std::cout << "Time for loop-based computation: " << timeloop.count() << " seconds\n";

start = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

bb[i] += A[i][j] \* x[j]; // This effectively mimics partial vectorization

}

}

end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timeloopvec = end - start;

std::cout << "Time for partial vectorized computation: " << timeloopvec.count() << " seconds\n";

start = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < n; ++i) {

bbb[i] = 0.0;

for (int j = 0; j < n; ++j) {

bbb[i] += A[i][j] \* x[j];

}

}

end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timevec = end - start;

std::cout << "Time for fully vectorized computation: " << timevec.count() << " seconds\n";

double norm1 = 0.0, norm2 = 0.0;

for (int i = 0; i < n; ++i) {

norm1 += (b[i] - bb[i]) \* (b[i] - bb[i]);

norm2 += (b[i] - bbb[i]) \* (b[i] - bbb[i]);

}

norm1 = std::sqrt(norm1);

norm2 = std::sqrt(norm2);

std::cout << "Norm between b and bb: " << norm1 << "\n";

std::cout << "Norm between b and bbb: " << norm2 << "\n";

double speedup = timeloop.count() / timeloopvec.count();

double speedup2 = timeloop.count() / timevec.count();

double speedup3 = timeloopvec.count() / timevec.count();

std::cout << "Speedup of loop-based vs partial vectorization: " << speedup << "\n";

std::cout << "Speedup of loop-based vs full vectorization: " << speedup2 << "\n";

std::cout << "Speedup of partial vectorization vs full vectorization: " << speedup3 << "\n";

return 0;

}

**Matrix matrix C++**

#include <iostream>

#include <vector>

#include <chrono>

#include <cmath>

double calculateNorm(const std::vector<std::vector<double>>& A, const std::vector<std::vector<double>>& B, int n) {

double norm = 0.0;

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

norm += (A[i][j] - B[i][j]) \* (A[i][j] - B[i][j]);

}

}

return std::sqrt(norm);

}

int main() {

const int n = 1000;

std::vector<std::vector<double>> A(n, std::vector<double>(n));

std::vector<std::vector<double>> B(n, std::vector<double>(n));

std::vector<std::vector<double>> C(n, std::vector<double>(n, 0.0));

std::vector<std::vector<double>> CC(n, std::vector<double>(n, 0.0));

std::vector<std::vector<double>> CCC(n, std::vector<double>(n, 0.0));

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

A[i][j] = static\_cast<double>(rand()) / RAND\_MAX;

B[i][j] = static\_cast<double>(rand()) / RAND\_MAX;

}

}

auto start = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

for (int k = 0; k < n; ++k) {

C[i][j] += A[i][k] \* B[k][j];

}

}

}

auto end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timeloop = end - start;

std::cout << "Time for loop-based computation: " << timeloop.count() << " seconds\n";

start = std::chrono::high\_resolution\_clock::now();

for (int j = 0; j < n; ++j) {

for (int i = 0; i < n; ++i) {

for (int k = 0; k < n; ++k) {

CC[i][j] += A[i][k] \* B[k][j];

}

}

}

end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timeloopvec = end - start;

std::cout << "Time for partial vectorized computation: " << timeloopvec.count() << " seconds\n";

start = std::chrono::high\_resolution\_clock::now();

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

CCC[i][j] = 0.0;

for (int k = 0; k < n; ++k) {

CCC[i][j] += A[i][k] \* B[k][j];

}

}

}

end = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double> timevec = end - start;

std::cout << "Time for fully vectorized computation: " << timevec.count() << " seconds\n";

double norm1 = calculateNorm(C, CC, n);

double norm2 = calculateNorm(C, CCC, n);

std::cout << "Norm between C and CC: " << norm1 << "\n";

std::cout << "Norm between C and CCC: " << norm2 << "\n";

double speedup = timeloop.count() / timeloopvec.count();

double speedup2 = timeloop.count() / timevec.count();

double speedup3 = timeloopvec.count() / timevec.count();

std::cout << "Speedup of loop-based vs partial vectorization: " << speedup << "\n";

std::cout << "Speedup of loop-based vs full vectorization: " << speedup2 << "\n";

std::cout << "Speedup of partial vectorization vs full vectorization: " << speedup3 << "\n";

return 0;

}