#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
#include <time.h>

#define N 10 // Large matrix and vector size

// Vector addition  
void vectorAdd(float \*a, float \*b, float \*c, int n) {  
 #pragma omp parallel for  
 for (int i = 0; i < n; i++) {  
 c[i] = a[i] + b[i];  
 }  
}

// Matrix multiplication  
void matrixMul(float \*a, float \*b, float \*c, int n) {  
 #pragma omp parallel for collapse(2)  
 for (int row = 0; row < n; row++) {  
 for (int col = 0; col < n; col++) {  
 float sum = 0.0f;  
 for (int k = 0; k < n; k++) {  
 sum += a[row \* n + k] \* b[k \* n + col];  
 }  
 c[row \* n + col] = sum;  
 }  
 }  
}

int main() {  
 float \*a, \*b, \*c;  
 size\_t size = N \* N \* sizeof(float);

// Allocate memory  
 a = (float\*)malloc(size);  
 b = (float\*)malloc(size);  
 c = (float\*)malloc(size);

if (!a || !b || !c) {  
 printf("Memory allocation failed\n");  
 return 1;  
 }

// Initialize vectors and matrices  
 srand(time(NULL));  
 #pragma omp parallel for  
 for (int i = 0; i < N \* N; i++) {  
 a[i] = rand() / (float)RAND\_MAX;  
 b[i] = rand() / (float)RAND\_MAX;  
 }

double start, end;

// Vector addition  
 printf("Performing vector addition...\n");  
 start = omp\_get\_wtime();  
 vectorAdd(a, b, c, N \* N);  
 end = omp\_get\_wtime();  
 printf("Vector addition completed in %.3f seconds.\n", end - start);

// Matrix multiplication  
 printf("Performing matrix multiplication...\n");  
 start = omp\_get\_wtime();  
 matrixMul(a, b, c, N);  
 end = omp\_get\_wtime();  
 printf("Matrix multiplication completed in %.3f seconds.\n", end - start);

// Sample output  
 printf("Sample output of matrix multiplication (top-left 2x2):\n");  
 for (int i = 0; i < 2; i++) {  
 for (int j = 0; j < 2; j++) {  
 printf("%.2f ", c[i \* N + j]);  
 }  
 printf("\n");  
 }

free(a);  
 free(b);  
 free(c);

return 0;  
}