Задание

Реализовать на OpenMP многопоточную версию функцию DGEMV (matrix-vector product).

Определить предельные размеры матрицы и вектора (параметры m, n), которые помещаются в память вычислительного узла кластера (ноутбука, машины в учебном

классе). Оценить ускорение параллельной программы при числе потоков: 2, 4, 6, 8 и n = m =

15000, 20000, 25000 (построить график зависимости коэффициента S ускорения параллельной программы от числа P потоков).

#include <inttypes.h>

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define THREADS 8

double wtime() {

struct timeval t;

gettimeofday(&t, NULL);

return (double)t.tv\_sec + (double)t.tv\_usec \* 1E-6;

}

void matrix\_vector\_product(double\* a, double\* b, double\* c, int m, int n) {

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

c[i] = 0.0;

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

c[i] += a[i \* n + j] \* b[j];

}

}

void\* xmalloc(size\_t size) {

void\* result = malloc(size);

if (result)

return result;

exit(1);

}

void run\_serial(int n, int m) {

double \*a, \*b, \*c;

a = xmalloc(sizeof(\*a) \* m \* n);

b = xmalloc(sizeof(\*b) \* n);

c = xmalloc(sizeof(\*c) \* m);

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

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

a[i \* n + j] = i + j;

}

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

b[j] = j;

double t = wtime();

matrix\_vector\_product(a, b, c, m, n);

t = wtime() - t;

printf("Elapsed time (serial): %.6f sec.\n", t);

free(a);

free(b);

free(c);

}

void matrix\_vector\_product\_omp(double\* a, double\* b, double\* c, int m, int n) {

#pragma omp parallel num\_threads(THREADS)

{

int nthreads = omp\_get\_num\_threads();

int threadid = omp\_get\_thread\_num();

int items\_per\_thread = m / nthreads;

int lb = threadid \* items\_per\_thread;

int ub =

(threadid == nthreads - 1) ? (m - 1) : (lb + items\_per\_thread - 1);

for (int i = lb; i <= ub; i++) {

c[i] = 0.0;

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

c[i] += a[i \* n + j] \* b[j];

}

}

}

void run\_parallel(int n, int m) {

double \*a, \*b, \*c;

a = xmalloc(sizeof(\*a) \* m \* n);

b = xmalloc(sizeof(\*b) \* n);

c = xmalloc(sizeof(\*c) \* m);

#pragma omp parallel num\_threads(THREADS)

{

int nthreads = omp\_get\_num\_threads();

int threadid = omp\_get\_thread\_num();

int items\_per\_thread = m / nthreads;

int lb = threadid \* items\_per\_thread;

int ub =

(threadid == nthreads - 1) ? (m - 1) : (lb + items\_per\_thread - 1);

for (int i = lb; i <= ub; i++) {

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

a[i \* n + j] = i + j;

c[i] = 0.0;

}

}

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

b[j] = j;

double t = wtime();

matrix\_vector\_product\_omp(a, b, c, m, n);

t = wtime() - t;

printf("Elapsed time (parallel): %.6f sec.\n", t);

free(a);

free(b);

free(c);

}

int main(int argc, char\*\* argv) {

int m = 25000;

int n = 25000;

printf("Matrix-vector product (c[m] = a[m, n] \* b[n]; m = %d, n = %d)\n", m,

n);

printf("Memory used: %" PRIu64 " MiB\n",

((m \* n + m + n) \* sizeof(double)) >> 20);

//run\_serial(n, m);

run\_parallel(n, m);

return 0;

}

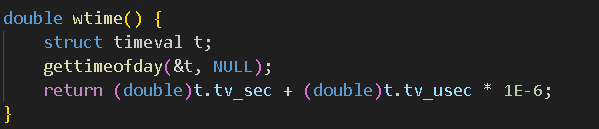
Код реализует параллельное умножение матрицы на вектор с использованием OpenMP.

Код определяет умножение матрицы a[m][n] на вектор b[n], чтобы получить результирующий вектор c[m].

Умножение матрицы на вектор реализовано двумя способами: последовательно и параллельно с использованием OpenMP.

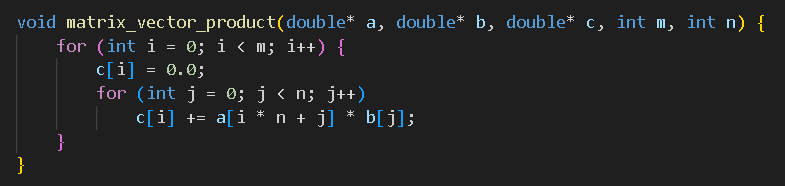
Параллельная версия предназначена для использования нескольких потоков, чтобы ускорить вычисления, распределяя работу между потоками.

Функция wtime



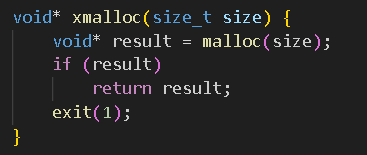
Эта функция возвращает текущее время в секундах, используя системное время. Она используется для измерения времени выполнения других функций.

Функция matrix\_vector\_product



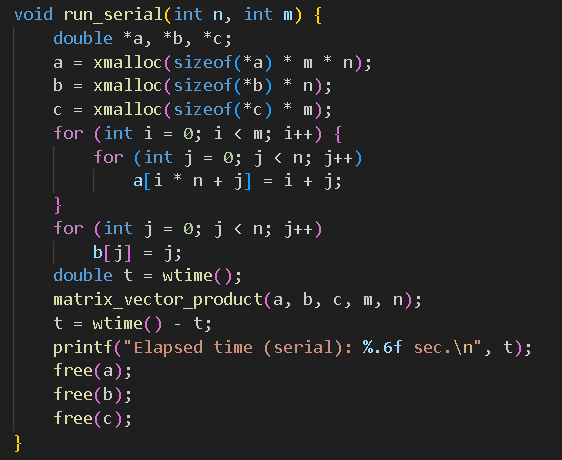
Эта функция последовательно вычисляет произведение матрицы на вектор. Она перебирает все элементы матрицы и вектора, выполняя необходимые операции умножения и сложения.

Функция xmalloc



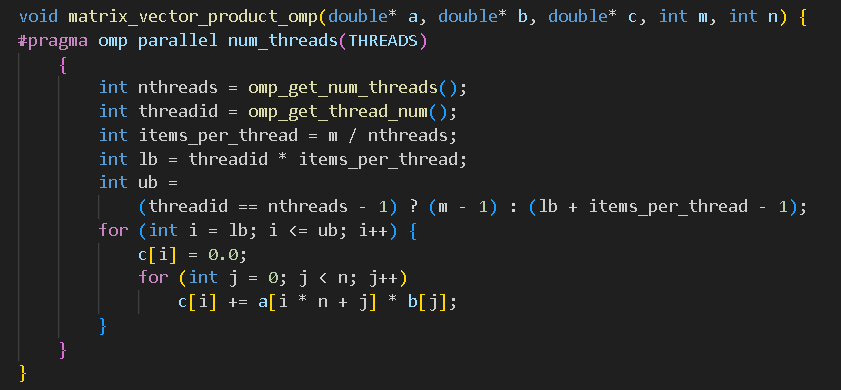
Эта функция выделяет память и проверяет, была ли память успешно выделена. Если нет, программа завершает выполнение.

Функция run\_serial



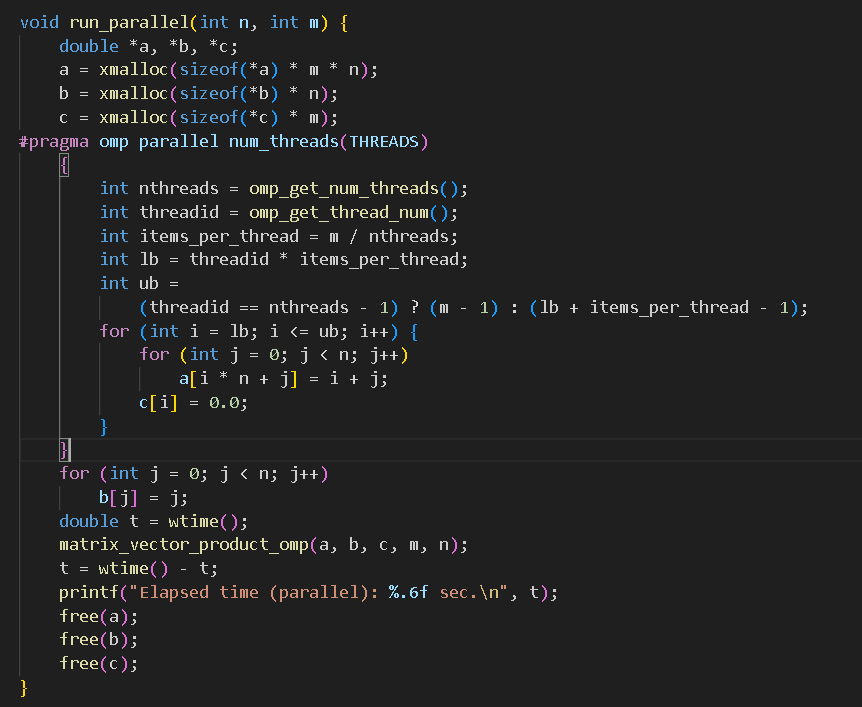
Эта функция выделяет память для матрицы и векторов, инициализирует их, выполняет последовательное умножение матрицы на вектор и выводит время выполнения.

Функция matrix\_vector\_product\_omp



Эта функция параллельно вычисляет произведение матрицы на вектор с использованием OpenMP. Каждому потоку назначается часть работы, и потоки выполняют операции умножения и сложения независимо друг от друга.

Функция run\_parallel



Эта функция выделяет память для матрицы и векторов, инициализирует их параллельно, выполняет параллельное умножение матрицы на вектор и выводит время выполнения.

Функция main

