**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«ВЕКТОРИЗАЦИЯ ВЫЧИСЛЕНИЙ»

студентки 2 курса, 19201 группы

Хаецкой Дарьи Владимировны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2020

**СОДЕРЖАНИЕ**

ЦЕЛЬ…………………………………………………………………...……3

ЗАДАНИЕ……………………………………………………………………3

ЛИСТИНГИ………………………………………………………………… 4

РЕЗУЛЬТАТЫ ИЗМЕРЕНИЙ…………………….……………………….11

ЗАКЛЮЧЕНИЕ……………...…………………………………………..…12

**ЦЕЛЬ**

Изучить SIMD-расширения архитектуры x86/x86-64 и способы использования SIMD-расширений в программах на языке Си. Получить навыки использования SIMD-расширений.

**ЗАДАНИЕ**

Написать три варианта программы, реализующей алгоритм обращения матрицы A размером N×N с помощью разложения в ряд:

* вариант без ручной векторизации
* вариант с ручной векторизацией
* вариант с матричными операциями, выполненными с использованием оптимизированной библиотеки BLAS.

Для элементов матриц использовать тип данных float.

**Листинги**

**Вариант #1**

#include "time.h"

#include <stdio.h>

#include <stdlib.h>

#define **N** 2048

#define **M** 10

*float*\*\* allocateMatrix(*int* n){

*float* \*\*A = (*float* \*\*)calloc(n, *sizeof*(*float* \*));

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

A[i] = (*float* \*)calloc(n, *sizeof*(*float*));

}

*return* A;

}

*void* freeMatrix(*float*\*\* A){

*for* (*int* i = 0; i < **N**; i++) {

free(A[i]);

}

free(A);

}

*float*\*\* transpose(*float*\*\* matrix, *float*\*\* A){

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++) {

A[i][j] = matrix[j][i];

}

}

*return* A;

}

*float* calcA1(*float*\*\* A){

*float* max = 0, sum = 0;

*for*(*int* i = 0; i < **N**; i++){

*for*(*int* j = 0; j < **N**; j++){

sum += abs(A[i][j]);

}

*if* (sum > max){

max = sum;

}

sum = 0;

}

*return* max;

}

*float* calcAinf(*float*\*\* A){

*float*\*\* TA = allocateMatrix(**N**);

transpose(A, TA);

*float* max = calcA1(TA);

freeMatrix(TA);

*return* max;

}

*void* sumMatrices(*float*\*\* a, *float*\*\* b, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++){

c[i][j] = a[i][j] + b[i][j];

}

}

}

*void* subMatrices(*float*\*\* a, *float*\*\* b, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++){

c[i][j] = a[i][j] - b[i][j];

}

}

}

*void* scalarDivMatrices(*float*\*\* a, *float* scalar, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++){

c[i][j] = a[i][j] / scalar;

}

}

}

*void* printMatrix(*float*\*\* A){

printf("\n");

*for* (*int* i = 0; i < **N**; ++i){

printf("%d: ", i);

*for* (*int* j = 0; j < **N**; ++j){

printf("%f ", A[i][j]);

}

printf("\n");

}

printf("\n");

}

*float*\*\* mulMatrices(*float*\*\* a, *float*\*\* b, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++) {

*for* (*int* k = 0; k < **N**; k++){

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

}

}

}

}

*float*\*\* generateE(){

*float*\*\* E = allocateMatrix(**N**);

*for* (*int* i = 0; i < **N**; i++){

*for* (*int* j = 0; j < **N**; j++){

*if* (i == j){

E[i][j] = 1;

} *else* {

E[i][j] = 0;

}

}

}

*return* E;

}

*float*\*\* invertMatrix(*float*\*\* A){

*float*\*\* sum = allocateMatrix(**N**);

*float*\*\* E = generateE();

*float*\*\* B = allocateMatrix(**N**);

*float*\*\* AT = allocateMatrix(**N**);

scalarDivMatrices(transpose(A,AT), calcA1(A)\*calcAinf(A), B);

freeMatrix(AT);

*float*\*\* BA = allocateMatrix(**N**);

mulMatrices(A, B, BA);

*float*\*\* R = allocateMatrix(**N**);

subMatrices(E, BA, R);

freeMatrix(BA);

*float*\*\* R\_prev = allocateMatrix(**N**);

mulMatrices(R, R, R\_prev);

sumMatrices(E, R, sum);

freeMatrix(E);

*float*\*\* R\_cur = allocateMatrix(**N**);

*for* (*int* i = 3; i < **M**; i ++){

*if* (i % 2 == 1){

sumMatrices(sum, R\_prev, sum);

mulMatrices(R, R\_prev, R\_cur);

} *else* {

sumMatrices(sum, R\_cur, sum);

mulMatrices(R, R\_cur, R\_prev);

}

}

*float*\*\* result = allocateMatrix(**N**);

mulMatrices(sum, B, result);

freeMatrix(sum);

freeMatrix(B);

freeMatrix(R);

freeMatrix(R\_prev);

freeMatrix(R\_cur);

*return* result;

}

*int* main() {

*float* \*\*A = allocateMatrix(**N**);

*float* \*\*A2;

srand(time(**NULL**));

*for* (*int* i = 0; i < **N**; i++){

*for* (*int* j = 0; j < **N**; j++){

A[i][j] = rand() % 11;

}

}

clock\_t start = clock();

A2 = invertMatrix(A);

printf("time taken is : %lf sec \n ", (*double*)(clock() - start)

/ **CLOCKS\_PER\_SEC**);

freeMatrix(A);

freeMatrix(A2);

*return* 0;

}

**Вариант #2**

#include "time.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define **N** 2048

#define **M** 10

*typedef enum* { absum, sum, sub, mul }operation;

*typedef struct*{

*float* x, y, z, w;

} Vector4;

**bool** isNullVec(Vector4 \*a){

printf("vec is null\n ");

*if* ( a->x == 0 &&

a->y == 0 &&

a->z == 0 &&

a->w == 0 ){

*return* **true**;

}

*return* **false**;

}

*void* SSE\_Add(Vector4 \*a, Vector4 \*b, Vector4 \*res){

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(a));

*\_\_asm\_\_ volatile*("mov %0, %%rbx":: "m"(b));

*\_\_asm\_\_ volatile*("movaps (%rax), %xmm0");

*\_\_asm\_\_ volatile*("movaps (%rbx), %xmm1");

*\_\_asm\_\_ volatile*("addps %xmm1, %xmm0");

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(res));

*\_\_asm\_\_ volatile*("movaps %xmm0, (%rax)");

}

*void* SSE\_Sub(Vector4 \*a, Vector4 \*b, Vector4 \*res){

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(a));

*\_\_asm\_\_ volatile*("mov %0, %%rbx":: "m"(b));

*\_\_asm\_\_ volatile*("movaps (%rax), %xmm0");

*\_\_asm\_\_ volatile*("movaps (%rbx), %xmm1");

*\_\_asm\_\_ volatile*("subps %xmm1, %xmm0");

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(res));

*\_\_asm\_\_ volatile*("movaps %xmm0, (%rax)");

}

*void* SSE\_Mul(Vector4 \*a, Vector4 \*b, Vector4 \*res){

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(a));

*\_\_asm\_\_ volatile*("mov %0, %%rbx":: "m"(b));

*\_\_asm\_\_ volatile*("movaps (%rax), %xmm0");

*\_\_asm\_\_ volatile*("movaps (%rbx), %xmm1");

*\_\_asm\_\_ volatile*("mulps %xmm1, %xmm0");

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(res));

*\_\_asm\_\_ volatile*("movaps %xmm0, (%rax)");

}

*void* SSE\_Max(Vector4 \*a, Vector4 \*b, Vector4 \*res){

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(a));

*\_\_asm\_\_ volatile*("mov %0, %%rbx":: "m"(b));

*\_\_asm\_\_ volatile*("movaps (%rax), %xmm0");

*\_\_asm\_\_ volatile*("movaps (%rbx), %xmm1");

*\_\_asm\_\_ volatile*("maxps %xmm1, %xmm0");

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(res));

*\_\_asm\_\_ volatile*("movaps %xmm0, (%rax)");

}

*void* SSE\_Div(Vector4 \*a, Vector4 \*b, Vector4 \*res){

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(a));

*\_\_asm\_\_ volatile*("mov %0, %%rbx":: "m"(b));

*\_\_asm\_\_ volatile*("movaps (%rax), %xmm0");

*\_\_asm\_\_ volatile*("movaps (%rbx), %xmm1");

*\_\_asm\_\_ volatile*("divps %xmm1, %xmm0");

*\_\_asm\_\_ volatile*("mov %0, %%rax":: "m"(res));

*\_\_asm\_\_ volatile*("movaps %xmm0, (%rax)");

}

*void* absSum(Vector4\* a, Vector4\* b, Vector4\* res){

Vector4 null = {0};

Vector4 absA = {0};

Vector4 absB = {0};

Vector4 negv = {0};

SSE\_Sub(&null, a, &negv);

SSE\_Max(a, &negv, &absA);

SSE\_Sub(&null, b, &negv);

SSE\_Max(b, &negv, &absB);

SSE\_Add(&absA, &absB, res);

}

*void* VecArray(*float* \*a, *float* \*b, *float* \*res, operation op){

Vector4 \*xx, \*yy, \*ss;

xx = (Vector4\*)a;

yy = (Vector4\*)b;

ss = (Vector4\*)res;

*for*(*int* i = 0; i < **N** / 4; ++i, ++xx, ++yy, ++ss){

*switch* (op){

*case* absum:{

absSum(xx, yy, ss);

*break*;

}

*case* sum:{

SSE\_Add(xx, yy, ss);

*break*;

}

*case* sub:{

SSE\_Sub(xx, yy, ss);

*break*;

}

*case* mul:{

SSE\_Mul(xx, yy, ss);

*break*;

}

*default*:

*break*;

}

}

}

*void* VecArrayDiv(*float* \*a, *float* scalar, *float* \*res){

Vector4 \*xx, \*ss;

xx = (Vector4\*)a;

Vector4 yy = {scalar, scalar, scalar, scalar};

ss = (Vector4\*)res;

*for*(*int* i = 0; i < **N** / 4; ++i, ++xx, ++ss){

SSE\_Div(xx, &yy, ss);

}

}

*float* VecSum(*float* \*a){

Vector4 sum = {0, 0, 0, 0};

Vector4 \*xx;

xx = (Vector4\*)a;

*for*(*int* i = 0; i < **N** / 4; ++i, xx++){

SSE\_Add(xx, &sum, &sum);

}

*return* sum.x + sum.y + sum.z + sum.w;

}

*float* findMax(*float*\* arr){

*float* max = 1;

*for* (*int* i = 0; i < **N**; i ++){

*if* (arr[i] > max){

max = arr[i];

}

}

*return* max;

}

*float*\*\* allocateMatrix(*int* n){

*float* \*\*A = (*float* \*\*)malloc(n \* *sizeof*(*float* \*));

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

A[i] = (*float* \*)malloc(n \* *sizeof*(*float*));

}

*return* A;

}

*void* freeMatrix(*float*\*\* A){

*for* (*int* i = 0; i < **N**; i++) {

free(A[i]);

}

free(A);

}

*float*\*\* transpose(*float*\*\* matrix){

*float*\*\* A = allocateMatrix(**N**);

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++) {

A[i][j] = matrix[j][i];

}

}

*return* A;

}

*float* calcA1(*float*\*\* A){

*float* \*result = (*float*\* )calloc(**N**, *sizeof*(*float*));

*for* (*int* j = 0; j < **N**; ++j) {

result[j] = 0;

}

*for*(*int* i = 0; i < **N**; i++){

VecArray(A[i], result, result, sum);

}

*float* max = findMax(result);

free(result);

*return* max;

}

*float* calcAinf(*float*\*\* A){

*float*\*\* TA;

TA = transpose(A);

*float* max = calcA1(TA);

freeMatrix(TA);

*return* max;

}

*void* sumMatrices(*float*\*\* a, *float*\*\* b, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

VecArray(a[i], b[i], c[i], sum);

}

}

*void* subMatrices(*float*\*\* a, *float*\*\* b, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

VecArray(a[i], b[i], c[i], sub);

}

}

*void* scalarDivMatrices(*float*\*\* a, *float* scalar, *float*\*\* c){

*for* (*int* i = 0; i < **N**; i++) {

VecArrayDiv(a[i], scalar, c[i]);

}

}

*void* printMatrix(*float*\*\* A){

printf("\n");

*for* (*int* i = 0; i < **N**; ++i){

printf("%d: ", i);

*for* (*int* j = 0; j < **N**; ++j){

printf("%f ", A[i][j]);

}

printf("\n");

}

printf("\n");

}

*void* mulMatrices(*float*\*\* a, *float*\*\* TA, *float*\*\* c){

*float* \*product = (*float*\* )calloc(**N**, *sizeof*(*float*));

*float* cur = 0;

*for* (*int* i = 0; i < **N**; i++) {

*for* (*int* j = 0; j < **N**; j++) {

VecArray(a[i], TA[j], product, mul);

cur = VecSum(product);

c[j][i] = cur;

}

}

free(product);

}

*float*\*\* generateE(){

*float*\*\* E = allocateMatrix(**N**);

*for* (*int* i = 0; i < **N**; i++){

*for* (*int* j = 0; j < **N**; j++){

*if* (i == j){

E[i][j] = 1;

} *else* {

E[i][j] = 0;

}

}

}

*return* E;}

*float*\*\* invertMatrix(*float*\*\* A){

*float*\*\* sum = allocateMatrix(**N**);

*float*\*\* E = generateE();

*float*\*\* B = allocateMatrix(**N**);

*float*\*\* AT = transpose(A);

scalarDivMatrices(AT,calcA1(A)\*calcAinf(A), B);

freeMatrix(AT);

*float*\*\* BA = allocateMatrix(**N**);

*float*\*\* BT = transpose(B);

mulMatrices(A, BT, BA);

freeMatrix(B);

*float*\*\* R = allocateMatrix(**N**);

subMatrices(E, BA, R);

freeMatrix(BA);

*float*\*\* R\_prev = allocateMatrix(**N**);

*float*\*\* RT = transpose(R);

mulMatrices(R, RT, R\_prev);

sumMatrices(E, R, sum);

freeMatrix(E);

*float*\*\* R\_cur = allocateMatrix(**N**);

*for* (*int* i = 3; i < **M**; i ++){

*if* (i % 2 == 1){

sumMatrices(sum, R\_prev, sum);

mulMatrices(RT, R\_prev, R\_cur);

} *else* {

sumMatrices(sum, R\_cur, sum);

mulMatrices(RT, R\_cur, R\_prev);

}

}

*float*\*\* result = allocateMatrix(**N**);

mulMatrices(sum, BT, result);

freeMatrix(sum);

freeMatrix(BT);

freeMatrix(R);

freeMatrix(RT);

freeMatrix(R\_prev);

freeMatrix(R\_cur);

*return* result;

}

*int* main() {

*float* \*\*A = allocateMatrix(**N**);

*float* \*\*A2;

srand(time(**NULL**));

*for* (*int* i = 0; i < **N**; i++){

*for* (*int* j = 0; j < **N**; j++){

A[i][j] = rand() % 6;

}

}

clock\_t start = clock();

A2 = invertMatrix(A);

printf("time taken is : %lf sec \n ", (*double*)(clock() - start)

/ **CLOCKS\_PER\_SEC**);

freeMatrix(A);

freeMatrix(A2);

*return* 0;

}

**Вариант #3**

#include "time.h"

#include <stdio.h>

#include <stdlib.h>

#include <cblas.h>

#define N 2048

#define M 10

*float*\* allocateMatrix(*int* n){

*float* \*A = (*float* \*)calloc(n\*n, *sizeof*(*float*));

*return* A;

}

*void* freeMatrix(*float*\* A){

free(A);

}

*float*\* transpose(*float*\* matrix, *float*\* A){

*for* (*int* i = 0; i < N; i++) {

*for* (*int* j = 0; j < N; j++) {

A[i\*N+j] = matrix[j\*N+i];

}

}

*return* A;

}

*float* calcA1(*float*\* A){

*float* max = 0, sum = 0;

*for*(*int* i = 0; i < N; i++){

*for*(*int* j = 0; j < N; j++){

sum += abs(A[j \* N + i]);

}

*if* (sum > max){

max = sum;

}

sum = 0;

}

*return* max;

}

*float* calcAinf(*float*\* A){

*float*\* TA = allocateMatrix(N);

transpose(A, TA);

*float* max = calcA1(TA);

freeMatrix(TA);

*return* max;

}

*float*\* generateE(){

*float*\* E = allocateMatrix(N);

*for* (*int* i = 0; i < N; i++){

*for* (*int* j = 0; j < N; j++){

*if* (i == j){

E[j \* N + i] = 1;

} *else* {

E[j \* N + i] = 0;

}

}

}

*return* E;

}

*float*\* invertMatrix(*float*\* A){

*float*\* sum = generateE();

*float*\* E = generateE();

*float*\* B = allocateMatrix(N);

*// calculates B*

cblas\_sgemm(CblasRowMajor, CblasTrans, CblasNoTrans, N, N, N,

1/calcA1(A)\*calcAinf(A), A, N, E, N, 0, B, N);

*float*\* BA = allocateMatrix(N);

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0,

A, N, B, N, 0, BA, N);

*float*\* R = allocateMatrix(N);

*float*\* BAneg = allocateMatrix(N);

*// R = E - BA*

cblas\_saxpy(N \* N, -1.0, BA, 1, BAneg, 1);

cblas\_scopy(N \* N, BAneg, 1, R, 1);

freeMatrix(BA);

freeMatrix(BAneg);

*float*\* R\_prev = allocateMatrix(N);

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0,

R, N, R, N, 0, R\_prev, N);

*float*\* R\_cur = generateE();

cblas\_saxpy(N \* N, 1.0, R\_prev, 1, sum, 1);

*for* (*int* i = 2; i < M; i ++){

*if* (i % 2 == 1){

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N,

N, 1.0, R\_prev, N, R, N, 0, R\_cur, N);

cblas\_saxpy(N \* N, 1.0, R\_prev, 1, sum, 1);

} *else* {

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N,

N, 1.0, R\_cur, N, R, N, 0, R\_prev, N);

cblas\_saxpy(N \* N, 1.0, R\_cur, 1, sum, 1);

}

}

*float*\* result = allocateMatrix(N);

cblas\_sgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, N, N, N, 1.0,

sum, N, B, N, 0, result, N);

freeMatrix(E);

freeMatrix(sum);

freeMatrix(B);

freeMatrix(R);

freeMatrix(R\_prev);

freeMatrix(R\_cur);

*return* result;

}

*int* main() {

*float* \*A = allocateMatrix(N);

*float* \*A2;

srand(time(NULL));

*for* (*int* i = 0; i < N; i++){

*for* (*int* j = 0; j < N; j++){

A[j \* N + i] = rand() % 11;

}

}

clock\_t start = clock();

A2 = invertMatrix(A);

printf("time taken is : %lf sec \n ", (*double*)(clock() - start)

/ CLOCKS\_PER\_SEC);

freeMatrix(A);

freeMatrix(A2);

*return* 0;

}

*return* 0;

}

**Результаты измерений**

|  |  |
| --- | --- |
| **Вариант программы** | **Время, с** |
| 1 | 421.039442 |
| 2 | 179.467761 |
| 3 | 11.683195 |

**ЗАКЛЮЧЕНИЕ**

В ходе работы были изучены SIMD-расширения архитектуры x86/x86-64 и их использование в коде. Получены навыки написания ассемблерных вставок в языке Си, а также изучена работу оптимизированной библиотеки линейной алгебры BLAS.  
Как было выяснено, лучше всего пользоваться готовой библиотекой, хотя ручная векторизация и может дать ощутимый прирост в производительности. Единственный недостаток библиотеки BLAS составляет большое количество разных параметров, из-за которых код становится трудно понимаемым.