به نام خدا

برای عملیات برداری در زبان c++ کتابخانه‌ای با نام <immintrin.h> تعریف شده است. داکیومتشن این کتابخانه در سایت مایکروسافت قابل مشاهده است ( لینک روبرو) : [x86 intrinsics](https://learn.microsoft.com/en-us/cpp/intrinsics/x86-intrinsics-list?view=msvc-170)

ما برنامه‌ی نمونه ای به این منظور نوشته‌ایم :

#include <iostream>

#include <immintrin.h>

#include <chrono>

#include <random>

using namespace std;

void addArrays\_serial(float\* a, float\* b, float\* c, int n) {

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

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

}

}

void addArrays\_vectorize2(float\* a, float\* b, float\* c, int n) {

int i;

\_\_m256 vec\_a, vec\_b, vec\_c;

for (i = 0; i < n - 7; i += 8) {

vec\_a = \_mm256\_load\_ps(&a[i]);

vec\_b = \_mm256\_load\_ps(&b[i]);

vec\_c = \_mm256\_add\_ps(vec\_a, vec\_b);

\_mm256\_store\_ps(&c[i], vec\_c);

}

for (; i < n; i++) {

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

}

}

void addArrays\_vectorize1(float\* a, float\* b, float\* c, int n) {

int i;

\_\_m128 vec\_a, vec\_b, vec\_c;

for (i = 0; i < n - 7; i += 4) {

vec\_a = \_mm\_load\_ps(&a[i]);

vec\_b = \_mm\_load\_ps(&b[i]);

vec\_c = \_mm\_add\_ps(vec\_a, vec\_b);

\_mm\_store\_ps(&c[i], vec\_c);

}

for (; i < n; i++) {

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

}

}

void print (float \*a,int n){

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

cout<<a[i]<<' ';

}

}

void set\_random(float \*a , int n){

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<float> dis(0.0f, 1.0f); // Range: [0.0, 1.0)

// Populate the array with random numbers

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

a[i] = dis(gen);

}

}

int main()

{

int size\_n=100000;

float a[size\_n], b[size\_n], c[size\_n];

set\_random(a,size\_n);

set\_random(b,size\_n);

const int numIterations = 1000;

// Measure performance of serial implementation

auto start\_c1 = chrono::high\_resolution\_clock::now();

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

addArrays\_serial(a, b, c, size\_n);

}

auto end\_c1 = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::milli> duration\_serial = end\_c1 - start\_c1;

cout << " Serial (1 single-precision floating-point value ) implementation time: " << duration\_serial.count() << " ms" << endl<< endl;

که با توجه به مدل پردازنده کامپیوتر مورد استفاده (intel core i7 7500u) این پردازنده توان عملیات برداری تا ۲۵۶ بیت را دارا ست. در این برنامه ما زمان جمع دو بردار را به صورت سریال و به صورت برداری با سایز ۱۲۸ بیت و همچنین به صورت برداری با سایز ۲۵۶ بیت را باهم مقایسه کردیم به این نکته باید توجه داشت که برای کامپایل شدن این برنامه نیاز است تا تنظیماتی را بر کامپایلر خود اعمال کنید. اگر از کدبلاکس استفاده میکنید به این شرح است:

1. پروژه خود را در Code::Blocks باز کنید.
2. به منوی "Project" بروید و "Build Options" را انتخاب کنید.
3. در پنجره "Build Options"، بر روی تب "Compiler Settings" کلیک کنید.
4. در جعبه "Other Options"، پرچم کامپایلر -mavx یا -mavx2 را بر اساس دستورات خاصی که می‌خواهید فعال کنید، اضافه کنید. به عنوان مثال، برای فعال کردن دستورات AVX2، شما باید -mavx2 را اضافه کنید.( برخی از پرچم ها به صورت پیش فرض در کد بلاکس وجود دارد که میتوانید فقط ان پرچم را با روشن کردن گزینه آن فعال کنید)
5. بر روی "OK" کلیک کرده و پنجره "Build Options" را ببندید.
6. پروژه خود را دوباره بسازید تا پرچم‌های جدید کامپایلر اعمال شود.

// Measure performance of vectorized implementation

auto start\_c2 = chrono::high\_resolution\_clock::now();

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

addArrays\_vectorize1(a, b, c, size\_n);

}

auto end\_c2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::milli> duration\_vectorize = end\_c2 - start\_c2;

cout << "Vectorized\_1 (4 single-precision floating-point values) implementation time: " << duration\_vectorize.count() << " ms" << endl<< endl;

// Measure performance of vectorized implementation

auto start\_c3 = chrono::high\_resolution\_clock::now();

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

addArrays\_vectorize2(a, b, c, size\_n);

}

auto end\_c3 = chrono::high\_resolution\_clock::now();

chrono::duration<double, std::milli> duration\_vectorize2 = end\_c3 - start\_c3;

cout << "Vectorized\_2 (8 single-precision floating-point values) implementation time: " << duration\_vectorize2.count() << " ms" << endl;

return 0;

}

با استفاده از این سه تابع به زمان های زیر رسیدیم:

|  |  |
| --- | --- |
| program | time |
| serial | 328.047 ms |
| Vectorized\_128bit | 203.084 ms |
| Vectorized\_256bit | 93.721 ms |

همانطور که مشخص است با برداری سازی به تسریع خوبی رسیده ایم.