به نام خدا

برای عملیات برداری در زبان c++ کتابخانه‌ای با نام <immintrin.h> تعریف شده است. داکیومنتشن این کتابخانه در سایت مایکروسافت قابل مشاهده است (لینک روبرو) : intrinsics ۸۶x

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:

۱. پروژه خود را در Code::Blocks باز کنید.

۲. به منوی "Project" بروید و "Build Options" را انتخاب کنید.

۳. در پنجره"Build Options" ، بر روی تب "Compiler Settings" کلیک کنید.

٤. در جعبه"Other Options" ، پرچم کامپایلر -mavx یا -mavx2را بر اساس دستورات خاصی که می‌خواهید فعال کنید، اضافه کنید. به عنوان مثال، برای فعال کردن دستوراتAVX2 ، شما باید -mavx2را 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)

٥. بر روی "OK" کلیک کرده و پنجره "Build Options" را ببندید.

٦. 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:

time	program
ms ۳۲۸.۰۴۷	serial
ms ۲۰۳.۰۸۴	Vectorized_128bit
ms ۹۳.۷۲۱	Vectorized_256bit

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