Федеральноегосударственное бюджетное образовательное учреждение высшего образования. «Национально-исследовательский университет «Московскийэнергетический институт»
Кафедра ВМСС

Лабораторная работа №5 по курсу «Вычислительные системы»

Тема: Исследование SIMD-расширений процессора

Выполнил: студент группы А-08-19 Балашов С.А.
Проверил: Карпов А.В.

Москва, 2023 г.

Цель работы: изучение SIMD-расширений архитектуры x86 и их применение в программах на языке C.

Домашняя подготовка:
Написать программу скалярного произведения векторов с использованием встроенных функций SSE на языке C и без их использования (вектор x заполнить по правилу N+i, где N – номер группы, а вектор y – по правилу M+i, где M – номер по журналу). Сравнить время выполнения получившихся программ.
Проанализировать результаты и сделать вывод
Исходные данные представлены в таблице 1.

Таблица 1

Исходные данные

№ Бригады	Матрично-векторная процедура	Обозначения
4	kAx + y	A, B – квадратные матрицы x, y – векторы k, p – скаляры

Листинг А содержит код программы. Результаты выполнения программы представлены в таблице 2.

Таблица 2

Результаты работы программы

Объем массивов L	Без SSE сверху, с SSE внизу
40	L:40 Size of arrays 40 Result: 31180 The elapsed time is 0.000000 seconds Result: 31180 The elapsed time is 0.000000 seconds
400	Size of arrays 400 Result: 22223800 The elapsed time is 0.003000 seconds Result: 22223800 The elapsed time is 0.001000 seconds
4000	L:4000 Size of arrays 4000 Result: 21421438000 The elapsed time is 0.016000 seconds Result: 21421438000 The elapsed time is 0.009000 seconds

40000	L:40000 Size of arrays 40000 Result: 21342134380000 The elapsed time is 0.162000 seconds Result: 21342134380000 The elapsed time is 0.093000 seconds
400000	L:400000 Size of arrays 400000 Result: 21334213343750036 The elapsed time is 1.615000 seconds Result: 21334213343789036 The elapsed time is 0.933000 seconds

Вывод: проанализировав результаты выполнения программы, можно сделать вывод, что ускорение вычислений при использовании функций SSE возрастает примерно в 2 раза, это достигается за счёт распараллеливания данных в регистрах и одновременного выполнения векторных и скалярных операций.

Лабораторное задание:

В соответствии с вариантом задания (см. Таблицу 1) написать программу на языке С, реализующую матрично-векторную процедуру с использованием расширений SSE. Размерность матриц и векторов считать кратным четырем.

Листинг Б содержит код программы. Результаты выполнения программы представлены в таблице 3.

Таблица 3

Результаты работы программы

Объем массивов L	Без SSE сверху, с SSE внизу
40	Size of arrays 40 STANDART FUNCTION Result: 29524000 The elapsed time is 0.006000 seconds FUNCTION WITH SSE2 Result: 29524000 The elapsed time is 0.005000 seconds
400	Size of arrays 400 STANDART FUNCTION Result: 189374800000 The elapsed time is 0.633000 seconds FUNCTION WITH SSE2 Result: 189374800000 The elapsed time is 0.393000 seconds

4000	Size of arrays 4000 STANDART FUNCTION Result: 1802031880000000 The elapsed time is 63.102000 seconds FUNCTION WITH SSE2 Result: 1802031880000000 The elapsed time is 37.842000 seconds

Вывод: проанализировав результаты выполнения программы, можно сделать вывод, что ускорение комплексных вычислений при использовании функций SSE становится заметным при объемах около 400 и возрастает примерно в 2 раза по сравнению с обычным методом вычислений.

Листинг А

```
#include <stdio.h>
// SSE (V2)
#include <emmintrin.h>
#include <time.h>
#define ITERATIONS 1000
#define N 8
#define M 4
#define k (M+N)
// функция без использования SSE
double calc_standart(double* x, double* y, int n)
{
    double s = 0;
    for (int i = 0; i < n; i++)
    {
        s += x[i] * y[i];
    }
    return s;
}
// функция с использованием SSE
double calc_sse(double* x, double* y, int n)
{
    double sum;
    __m128d* xx, * yy;
    __m128d p, s;
    xx = (__m128d*)x;
    yy = (__m128d*)y;
    s = _mm_set_pd(0, 0);
    for (int i = 0; i < n / 2; i++)
    {
        p = _mm_mul_pd(xx[i], yy[i]);   // векторное умножение четырех чисел
        s = _mm_add_pd(s, p);           // векторное сложение четырех чисел
    }
    p = _mm_shuffle_pd(s, s, 1);        // перемещение второго значения в s в младшую позицию в p
    s = _mm_add_pd(s, p);               // скалярное сложение
    _mm_store_sd(&sum, s);              // запись младшего значения в память;
    return sum;
}
int main()
{
    double s;
    const int sizes[] = { 40, 400, 4000, 40000, 400000 };
    for (int w = 0; w < sizeof(sizes) / sizeof(int); w++) {
        int L = sizes[w];
        printf("L:%d\n", L);
        // выделение памяти с выравниванием
        double* x = (double*)_mm_malloc(L * sizeof(double), 16);
        double* y = (double*)_mm_malloc(L * sizeof(double), 16);
        if (x && y) {
            for (int i = 0; i < L; i++)
            {
                x[i] = N + i;
                y[i] = M + i;
            }
        }
        double time_spent = 0.0;
        printf("Size of arrays %d\n\n", L);
        clock_t begin = clock();
        for (int itt1 = 0; itt1 < ITERATIONS; itt1++)
        {
            s = calc_standart(x, y, L);
        }
        printf("\tResult: %.0f\n", s);
        clock_t end = clock();
        time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
        printf("\tThe elapsed time is %f seconds\n", time_spent);
        time_spent = 0.0;
        begin = clock();
        // Using SSE
        for (int itt2 = 0; itt2 < ITERATIONS; itt2++)
        {
            s = calc_sse(x, y, L);
        }
        printf("\tResult: %.0f\n", s);
        end = clock();
        time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
        printf("\tThe elapsed time is %f seconds\n\n", time_spent);
        _mm_free(x);
        _mm_free(y);
    }
    return 0;
}
```

Листинг Б

```
#include <stdio.h>
#include <emmintrin.h>
#include <time.h>
#define N 8
#define M 4
#define k (M+N)
// «обычная» функция
double* inner1(double* x, double* y, double** A, int n)
{
    double* s;
    int i;
    s = (double*)calloc(n, sizeof(double));
    for (i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            s[i] += k * A[i][j] * x[i] + y[i];
        }
    }
    return s;
}
// функция с использованием SSE
double* inner2(double* x, double* y, double** A, int n)
{
    double* sum;
    sum = (double*)calloc(n, sizeof(double));
    int i, j;
    __m128d *xx, *yy, **AA, p, s, kk, tm1, tm2;
    xx = (__m128d*)x;
    yy = (__m128d*)y;
    AA = (__m128d**)A;
    s = _mm_set1_pd(0);
    p = _mm_set1_pd(0);
    kk = _mm_set1_pd(k);
    for (i = 0; i < n; i++)
    {
        tm1 = _mm_set1_pd(0);
        tm2 = _mm_set1_pd(0);
        for (j = 0; j < n/2; j++)
        {
            p = _mm_mul_pd(AA[i][j], xx[j]);
            tm1 = _mm_add_pd(tm1, p);
            tm2 = _mm_add_pd(tm2, yy[j]);
        }
        tm1 = _mm_mul_pd(tm1, kk);
        s = _mm_add_pd(tm1, tm2);
        p = _mm_shuffle_pd(s, s, 1);
        s = _mm_add_pd(s, p);
        _mm_store_sd(&sum[i], s);
    }
    return sum;
}
int main()
{
    const int sizes[] = { 40, 400, 4000};
    for (int w = 0; w < 5; w++) {
        int L = sizes[w];
        double* x, * y, **A, *s, result;
        int i, j;
        // выделение памяти с выравниванием
        x = (double*)_mm_malloc(L * sizeof(double), 16);
        y = (double*)_mm_malloc(L * sizeof(double), 16);
        A = (double**)malloc(L * sizeof(double*));
        for (i = 0; i < L; i++)  // цикл по строкам
            {
                A[i] = (double*)_mm_malloc(L * sizeof(double), 16);
            }
        s = (double*)calloc(L, sizeof(double));
        if (x && y && A) {
            for (i = 0; i < L; i++)
            {
                x[i] = (double)(N + i);
                y[i] = (double)(M + i);
                if (A[i]) {
                    for (j = 0; j < L; j++)
                    {
                        A[i][j] = (double)(N + M + i + j);
                    }
                }
            }
        }
        double time_spent = 0.0;
        printf("Size of arrays %d\n\n", L);
        clock_t begin = clock();
```

```
        // Using x87
        for (int itt1 = 0; itt1 < 1000; itt1++)
        {
            s = inner1(x, y, A, L);
        }
        result = 0;
        printf("STANDART FUNCTION\n");
        printf("\tResult:\n\t");
        for (i = 0; i < L; i++)
        {
            result += s[i];
        }
        printf("%.0f\n", result);
        clock_t end = clock();
        time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
        printf("The elapsed time is %f seconds\n", time_spent);
        time_spent = 0.0;
        begin = clock();
        // Using SSE
        for (int itt2 = 0; itt2 < 1000; itt2++)
        {
            s = inner2(x, y, A, L);
        }
        result = 0;
        printf("\nFUNCTION WITH SSE2\n");
        printf("\tResult:\n\t");
        for (i = 0; i < L; i++)
        {
            result += s[i];
        }
        printf("%.0f\n", result);
        end = clock();
        time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
        printf("The elapsed time is %f seconds\n", time_spent);
        _mm_free(x);
        _mm_free(y);
        for (i = 0; i < L; i++)
        {
            _mm_free(A[i]);
        }
        free(A);
        free(s);
    }
    return 0;
}
```