Московский Физико-Технический Институт

Отчет по эксперименту

Низкоуровневая оптимизация параллельных алгоритмов

Выполнил:
Студент 1 курса ФРКТ
Группа Б01-302
Хальфин Бахтияр

Долгопрудный
2024

Ключевые слова

ПАРАЛЛЕЛИЗМ, ВЕКТОРИЗАЦИЯ, ОПТИМИЗАЦИЯ, SIMD, AVX, AVX2, INTINSIC ФУНКЦИИ, МНОЖЕСТВО МАНДЕЛЬБРОТА, ФРАКТАЛЫ

Цель работы: оптимизировать функцию, обрабатывающую большое количество значений, и сравнить производительность разных реализаций. В качестве оптимизируемой функции используется функция расчета множества Мандельброта.

Оборудование: Персональный компьютер (ПК) с центральным процессором (ЦП), поддерживающим как минимум AVX2 инструкции; монитор; клавиатура и мышь.

Программное обеспечение: Операционая система (ОС) Linux; компилятор GCC; графическая библиотека SDL2 с расширением SDL2_ttf; инструмент для сборки проекта GNU Make; программа визуализирующая множество Мандельброта

Полученные результаты:

- В зависимости от степени оптимизации отношение скорости примитивной реализации к векторизированной лежит в диапазоне от 2.76 (-O0) до 6.44(-Os).
- Флаги оптимизации ускорили примитивную реализацию в 2.75 раз. Наименьшая скорость при -O0, наибольшая при -O1.
- Флаги оптимизации ускорили векторизированную реализацию в 5.44 раз. Наименьшая скорость при -O0, наибольшая при -O1.
- Наилучшая производительность программы наблюдается с флагом -O1.

Введение

Теоретические сведения

Параллельные вычисления - это тип вычислений, в котором множество вычислений или процессов выполняются одновременно.

Векторизация (в параллельных вычислениях) — вид распараллеливания программы, при котором однопоточные приложения, выполняющие одну операцию в каждый момент времени, модифицируются для выполнения нескольких однотипных операций одновременно.

Скалярные операции, обрабатывающие по паре операндов, заменяются на операции над векторами, обрабатывающие несколько элементов вектора в каждый момент времени.

Например, фрагмент программы, который поэлементно перемножает два массива может быть векторизирован следующим образом.

```
for (i = 0; i < 1024; i++)
    C[i] = A[i] * B[i];
```

```
for (i = 0; i < 1024; i+=4)
    C[i:i+3] = A[i:i+3] * B[i:i+3];
```

Запись $C[i:i+3]$ означает вектор из 4 элементов — от $C[i]$ до $C[i+3]$ включительно, а под * понимается операция поэлементного умножения векторов. Векторный процессор в данном примере сможет выполнить 4 скалярные операции при помощи одной векторной.

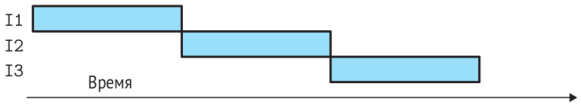


Рис. 1: Неконвейерная реализация

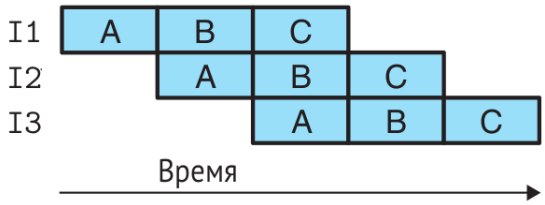


Рис. 2: Конвейерная реализация

SIMD (англ. single instruction, multiple data — одиночный поток команд, множественный поток данных) — принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных.

Короткие SIMD инструкции (64 или 128 бит) стали появляться в 1990-х годах. В 2010 году компания Intel представила SIMD-расширение AVX в процессорах архитектуры Sandy Bridge.

Intrinsic (англ. внутренний) функции - это функции, реализация которой специально обрабатывается компилятором. Как правило, она может заменять последовательность автоматически генерируемых инструкций для вызова оригинальной функции, подобно *inline* функции. В отличие от *inline* функции, компилятор обладает глубокими знаниями о *intrinsic* функции и поэтому может лучше интегрировать и оптимизировать ее для конкретной ситуации.

Intrinsic функции часто используются для векторизации и параллелизации. Компиляторы для C и C++ преобразуют *intinsic* непосредственно в *SIMD* инструкции.

Множество Мандельброта

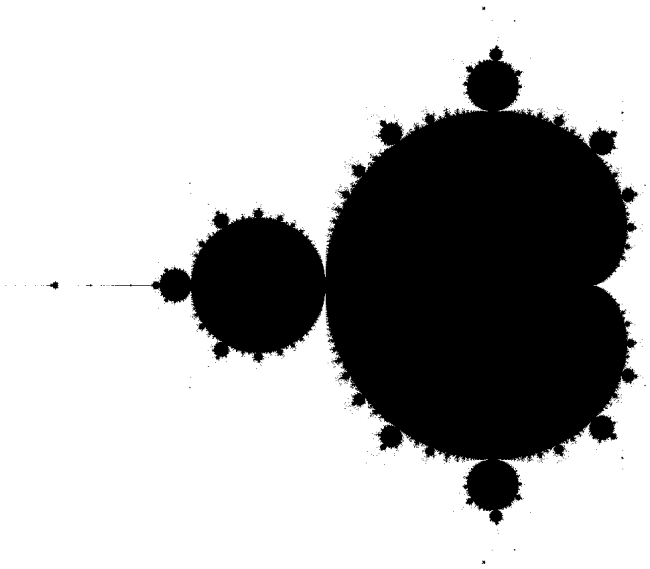


Рис. 3: Пример визуализации множества Мандельброта

Множество Мандельброта — множество точек c на комплексной плоскости, которое задается рекуррентным соотношением $Z_n = Z_n^2 + c$, где $Z_0 = 0$.

Иначе говоря, это множество таких , для которых существует такое действительное R, что неравенство $|z_n| < R$ выполняется при всех натуральных n.

Множество Мандельброта является одним из самых известных фракталов, в том числе за пределами математики, благодаря своим цветным визуализациям

Построение множества

Переформулируем соотношение, описанное выше. Заменим Z_n на x_n и y_n и получим значения координат комплексной плоскости (x, y):

$$\begin{aligned} x_{n+1} &= x_n^2 - y_n^2 + x_0 \\ y_{n+1} &= 2x_n y_n + y_0 \end{aligned} \tag{1}$$

Очевидно, что как только модуль Z_n окажется больше 2, все последующие модули последовательности станут стремиться к бесконечности. В случае $|c| > 2$ это можно доказать с помощью метода математической индукции. При $|\mathrm{c}| > 2$ точка с заведомо не принадлежит множеству Мандельброта, что можно вывести методом математической индукции, используя равенство $Z_0 = 0$.

Виды реализаций вычислений множества Мандельброта

Примитивный

В данной реализации выражения (1) просто переведены в код на C, каждый пиксель обрабатывается отдельно

```
FOR EACH pixel on the screen (x0, y0)
    x := x0
    y := y0

    i := 0
    FOR i TO MAX_ITERATION_NUMBER
        IF x*x + y*y > 4 THEN
            BREAK
        END IF

        x = x*x - y*y + x0
        y = 2*x*y + y0
        i++
    ENDLOOP

    PAINT(x0, y0, i)
```

С векторизацией

В данной реализации используются AVX2 инструкции, одновременной обрабатываются 8 пикселей.

```
FOR EACH 8 pixels on the screen (x0, y0)
    x := x0
    y := y0
    i := _mm256_setzero_si256();
    FOR i TO MAX_ITERATION_NUMBER
        x2       := _mm256_mul_ps(x,  x)
        y2       := _mm256_mul_ps(y,  y)
        xy       := _mm256_mul_ps(x,  y)
        radius2  := _mm256_add_ps(x2, y2)

        cmp_mask := _mm256_cmp_ps(radius2, MAX_RADIUS_2_256,
                                  _CMP_LT_OQ)
        IF (_mm256_testz_ps(cmp_mask, cmp_mask)) THEN
            BREAK
        END IF

        x = _mm256_add_ps(x0, _mm256_sub_ps(x2, y2))
        y = _mm256_add_ps(y0, _mm256_add_ps(xy, xy))

        iterations = _mm256_sub_epi32(iterations,
                     _mm256_castps_si256(cmp_mask))
```

ENDLOOP

PAINT(x0 , y0 , i)

Формально алгоритм остался таким же, но вместо чисел вектора из 8 чисел.

Экспериментальная установка

Характеристики системы, на которой снимались значения:

OS	Linux Mint 21.3 x86_64
Kernel	6.1.0-1036-oem
CPU	AMD Ryzen 7 5700U

Методика измерений

Разрешение	1600x900
Количество вызовов	100
Макс. Число итераций	256

Таблица 1: Параметры программы

Количество тактов вычисляется с помощью функции clock() из библиотеки time.h.

Вычисляется разница тактов перед многократным вызовом функции и после.

Измерения проводятся для с флагами оптимизации -O0, -O1, -O2, -O3, -Os.

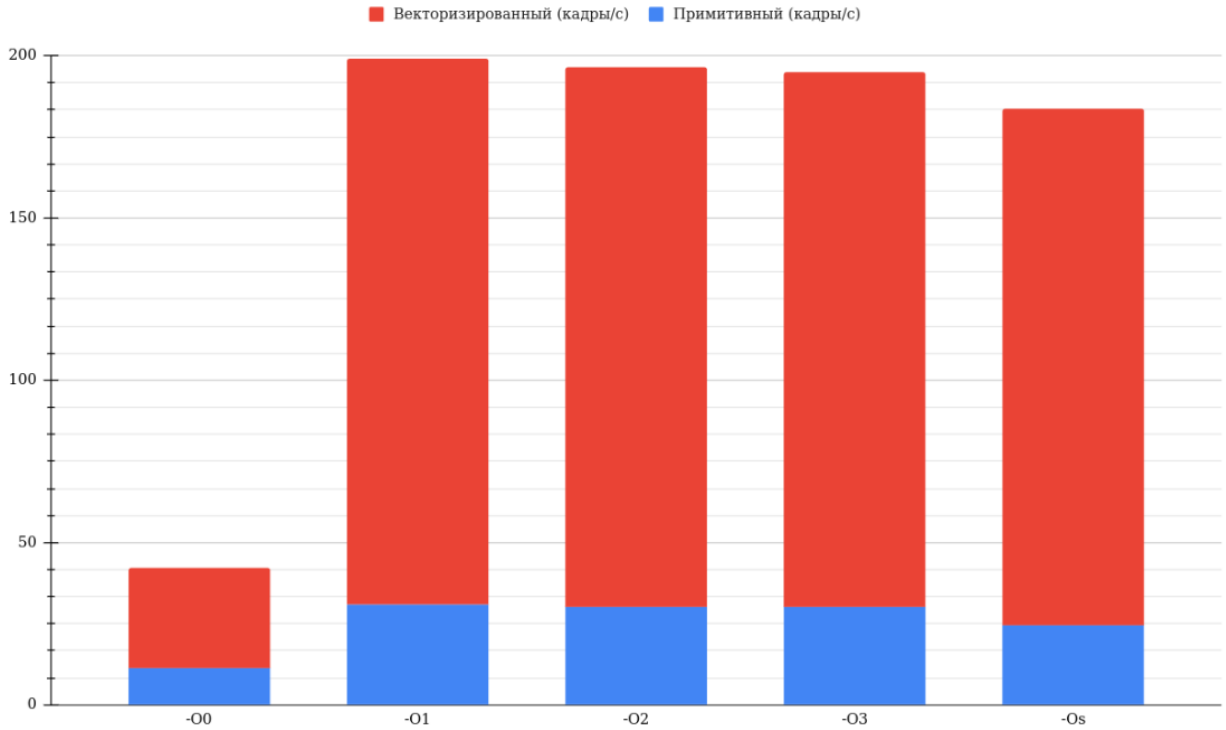
Влиянием температуры процессора, кэширования, пыли в компьютере, космических лучей мы пренебрегаем.

Результаты измерений

	-O0	-O1	-O2	-O3	-Os
П* (такты)	8934761	3249166	3309362	3310215	4047199
В* (такты)	3233748	593930	601007	607176	628752
П* (такты / вызов)	34901,41016	12692,05469	12927,19531	12930,52734	15809,37109
В* (такты/вызов)	12631,82813	2320,039063	2347,683594	2371,78125	2456,0625
П* (кадры/с)	11,19224118	30,77712865	30,21730473	30,20951811	24,70844651
В* (кадры/с)	30,92386915	168,3700099	166,3874131	164,6968918	159,0452197
П*/В*	2,7629738	5,470621117	5,506361823	5,451821218	6,436876543

Таблица 2: Количество тактов для каждой конфигурации

П* - примитивный, В* - векторизированный



Список литературы

1. Р. Брайант, Д. О'Халларон "Компьютерные системы. Архитектура и программирование".
2. Intrinsic function - https://en.wikipedia.org/wiki/Intrinsic_function
3. Mandelbrot set - https://en.wikipedia.org/wiki/Mandelbrot_set