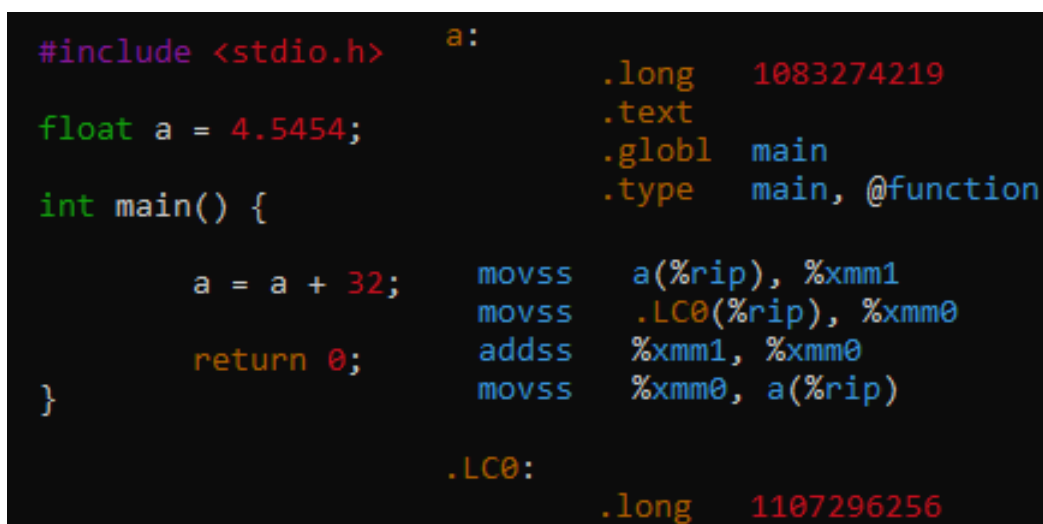


Лабораторная работа 4

Калашников Михаил, Б03-205

Числа с плавающей точкой (Asm)

1. Посмотрим как в листинге выглядит суммирование даблов



```
#include <stdio.h>    a:
float a = 4.5454;      .long    1083274219
int main() {          .text
                        .globl   main
                        .type    main, @function
    a = a + 32;        movss    a(%rip), %xmm1
                        movss    .LC0(%rip), %xmm0
    return 0;          addss    %xmm1, %xmm0
                        movss    %xmm0, a(%rip)
}

                        .LC0:
                        .long    1107296256
```

Рис. 1

Суммирование происходит с помощью операнда addss.

2. Теперь напишем программу, которая находит среднее из 16 чисел.
Посмотрим на ее листинги при разных оптимизациях.

```

float a[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
float m = 0;

int main() {
    for (int i = 0; i < 16; i++) {
        m += a[i];
    }
    m /= 16;
    return 0;
}

main:                                     -O0
.LFB0:
.cfi_startproc
endbr64
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
movl    $0, -4(%rbp)
jmp     .L2

.L2:
movl    -4(%rbp), %eax
cltq
leaq    0(, %rax, 4), %rdx
leaq    a(%rip), %rax
movss   m(%rip), %xmm0
addss   %xmm1, %xmm0
shufps  $85, %xmm1, %xmm2
addss   %xmm2, %xmm0
movaps  %xmm1, %xmm2
unpckhps %xmm1, %xmm2
shufps  $255, %xmm1, %xmm1
addss   %xmm2, %xmm0
addss   %xmm1, %xmm0
movaps  16+ a(%rip), %xmm1
movaps  %xmm1, %xmm2
addss   %xmm1, %xmm0
shufps  $85, %xmm1, %xmm2
addss   %xmm2, %xmm0
movaps  %xmm1, %xmm2
unpckhps %xmm1, %xmm2
shufps  $255, %xmm1, %xmm1
addss   %xmm2, %xmm0
addss   %xmm1, %xmm0
movaps  32+ a(%rip), %xmm1
movaps  %xmm1, %xmm2
addss   %xmm1, %xmm0
shufps  $85, %xmm1, %xmm2
addss   %xmm2, %xmm0
movaps  %xmm1, %xmm2
unpckhps %xmm2, %xmm1
shufps  $255, %xmm2, %xmm2
addss   %xmm1, %xmm0
addss   %xmm2, %xmm0
mulss  160(%rip), %xmm0
movss  %xmm0, m(%rip)

.LC0:
.long 1031798784

main:                                     -O1
.LFB0:
.cfi_startproc
endbr64
movss   m(%rip), %xmm0
leaq    a(%rip), %rax
leaq    64(%rax), %rdx
.L2:
addss   (%rax), %xmm0
addq    $4, %rax
cmpq    %rdx, %rax
jne     .L2
mulss   160(%rip), %xmm0
movss   %xmm0, m(%rip)
movl    $0, %eax
ret

.LC0:
.long 1031798784

main:                                     -O2
.LFB0:
.cfi_startproc
endbr64
leaq    a(%rip), %rax
movss   m(%rip), %xmm0
leaq    64(%rax), %rdx
.p2align 4,,10
.p2align 3
.L2:
addss   (%rax), %xmm0
addq    $4, %rax
cmpq    %rdx, %rax
jne     .L2
mulss   160(%rip), %xmm0
xorl    %eax, %eax
movss   %xmm0, m(%rip)
ret

main:                                     -O3
.LFB0:
.cfi_startproc
endbr64
movaps  a(%rip), %xmm1
movss   m(%rip), %xmm0
xorl    %eax, %eax
addss   %xmm1, %xmm0
movaps  %xmm1, %xmm2
shufps  $85, %xmm1, %xmm2
addss   %xmm2, %xmm0
movaps  %xmm1, %xmm2
unpckhps %xmm1, %xmm2
shufps  $255, %xmm1, %xmm1
addss   %xmm2, %xmm0
addss   %xmm1, %xmm0
movaps  16+ a(%rip), %xmm1
movaps  %xmm1, %xmm2
addss   %xmm1, %xmm0
shufps  $85, %xmm1, %xmm2
addss   %xmm2, %xmm0
movaps  %xmm1, %xmm2
unpckhps %xmm2, %xmm1
shufps  $255, %xmm2, %xmm2
addss   %xmm1, %xmm0
addss   %xmm2, %xmm0
mulss  160(%rip), %xmm0
movss   %xmm0, m(%rip)

```

Рис. 2

При -O0 – -O2 понятно что происходит, а вот при -O3 начинается жёсть с записывание флоатов в пакеты. Этим мы сейчас и займем-ся.

3. Распараллелим задачу. Разобьем массив на 4 штучки и просуммируем их как пакеты с помощью addps. Оно работает.

```

#include <stdio.h>

float a[] = { 0, 1, 2, 3 },
        b[] = { 4, 5, 6, 7 },
        c[] = { 8, 9, 10, 11 },
        d[] = { 12, 13, 14, 15 };

float m = 0;

int main() {

    asm(
        "movups a(%rip), %xmm0\n\t"
        "movups b(%rip), %xmm1\n\t"
        "movups c(%rip), %xmm2\n\t"
        "movups d(%rip), %xmm3\n\t"
        "addps %xmm3, %xmm0\n\t"
        "addps %xmm2, %xmm0\n\t"
        "addps %xmm1, %xmm0\n\t"
        "movups %xmm0, a(%rip)"
    );

    for (int i = 0; i < 4; i++) {
        m += a[i];
    }

    m /= 16;

    printf("%f\n", m);

    return 0;
}

sanya@LAPTOP-TSG87PQU:~/miumiu$ ./a.out
7.500000

```

Рис. 3

4. Теперь очередь конвейера операций. Будем раскидывать данные по корзинам и суммировать корзины в надежде уравнять количество операций для процессора и математического сопроцессора.

```
t = clock();

for (unsigned int i = 0; i < N_elements; i++) {
    bins[i % N_bins] += a[i];
}

for (unsigned int i = 0; i < N_bins; i++) {
    mean += bins[i];
}

mean /= N_elements;

t = clock() - t;
```

Рис. 4

В начале попробуем определить оптимальное количество корзин. Построим график зависимости времени поиска среднего в массиве из ста миллионов случайных флоатов от количества корзин.

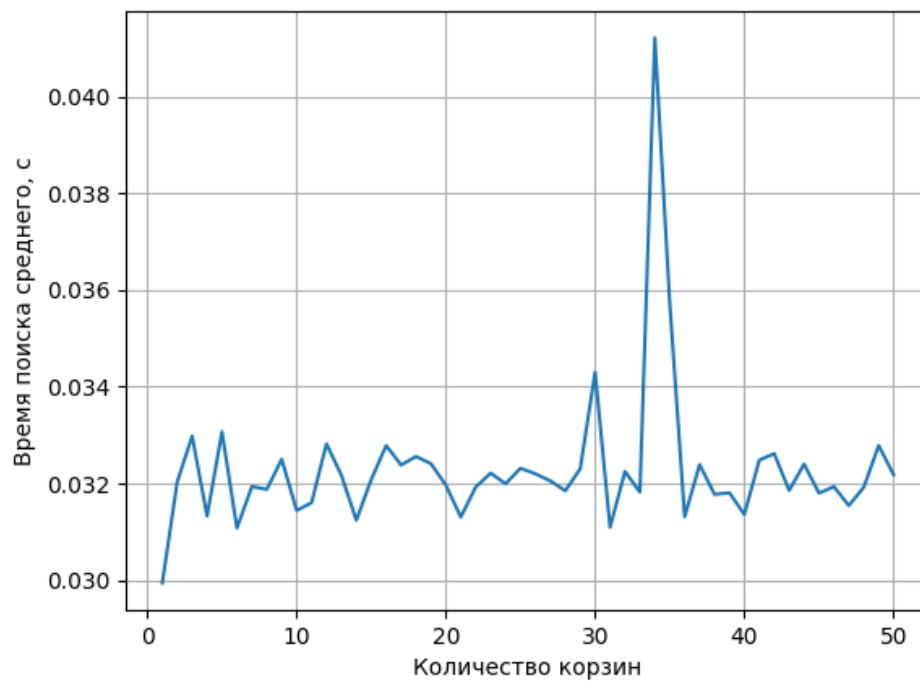


Рис. 5

Из графика можно сделать вывод что время от количества корзин почти не зависит. В дальнейшем будем использовать 4 корзины. Построим график зависимости времени поиска от количества элементов в массиве.

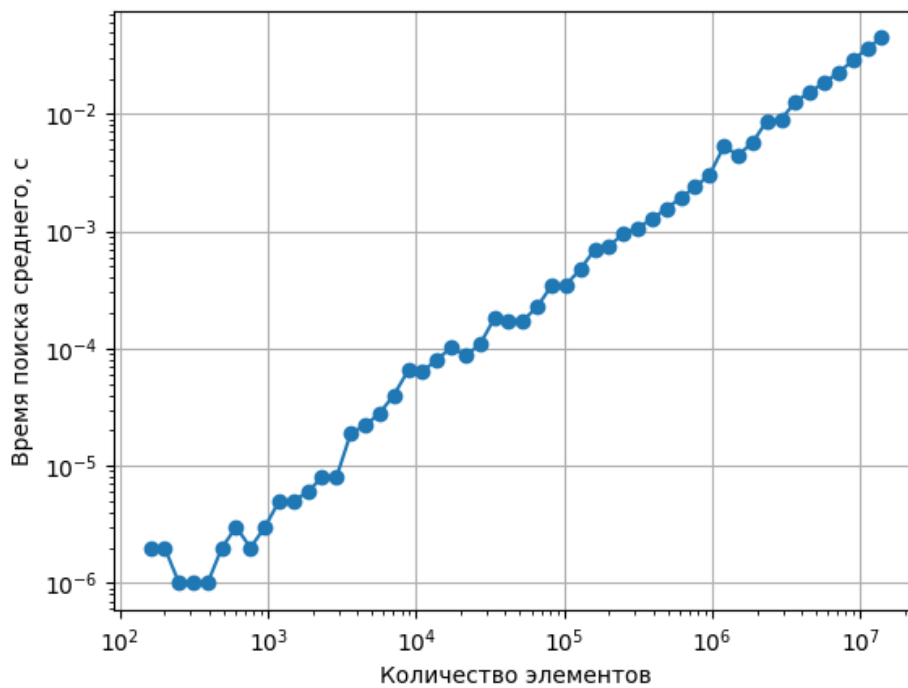


Рис. 6

Наложим время поиска обычным алгоритмом, но с включенной оптимизацией. Можно видеть что обогнать их не удалось. :(

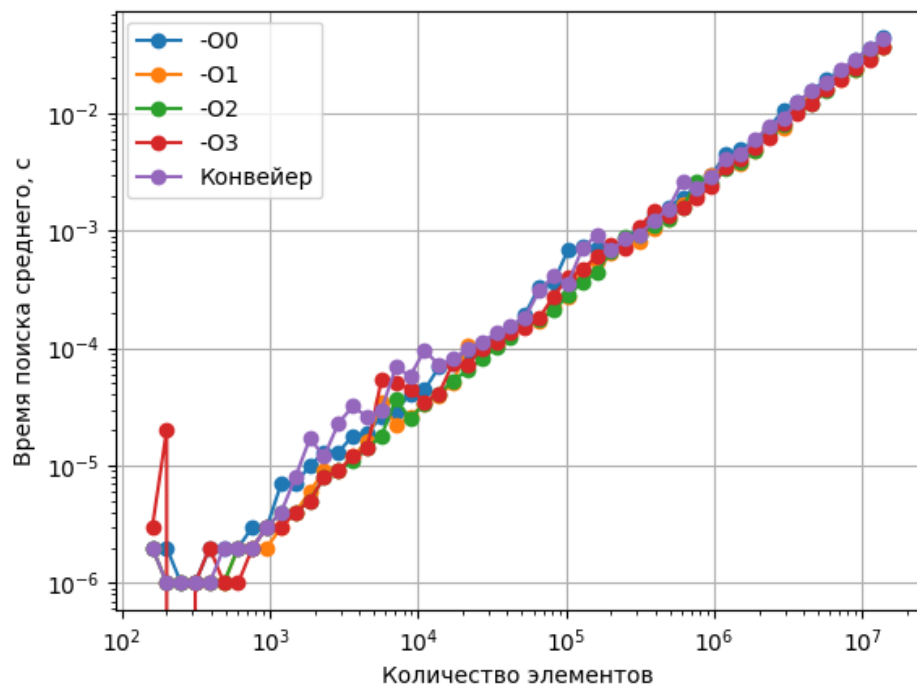


Рис. 7

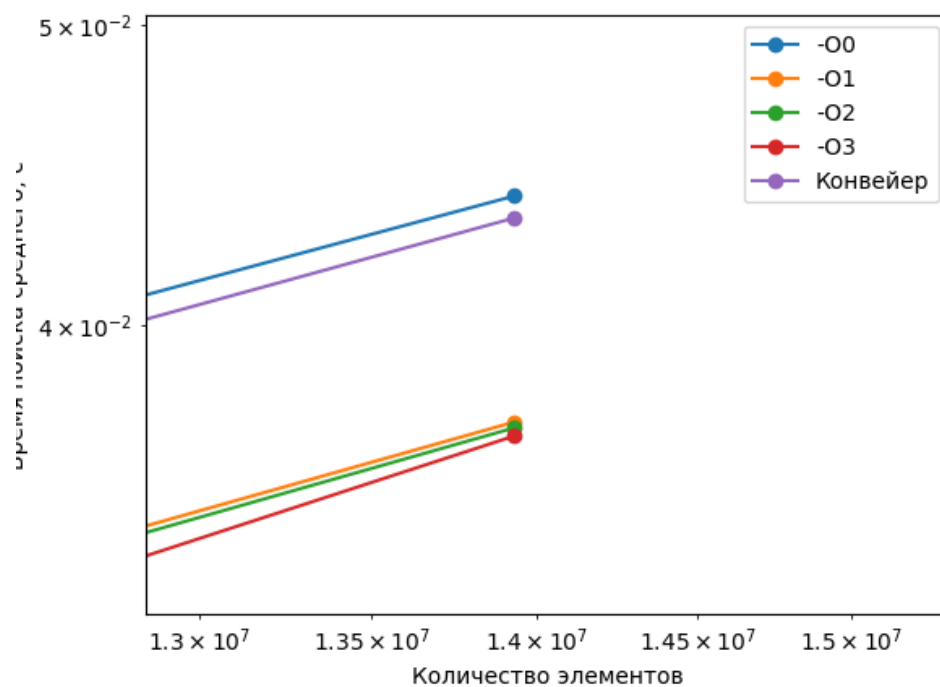


Рис. 8

5. Теперь попробуем оптимизировать численное решение ограниченной задачи двух тел. Я вынес все вычисления в функцию `iterate`.


```
void iterate() {  
    dist = sqrt(rx * rx + ry * ry);  
    a = mu / dist / dist / dist;  
    ax = rx * a;  
    ay = ry * a;  
    vx = vx + ax * dt;  
    vy = vy + ay * dt;  
    rx = rx + vx * dt;  
    ry = ry + vy * dt;  
}
```

Рис. 9

Я выбрал полностью переписать эту функцию на ассемблере для максимальной оптимизации. Получилось вот это:

```

void iterate_asm() {
    asm(
        "movsd rx(%rip), %xmm0\n\t"
        "movsd ry(%rip), %xmm1\n\t"

        "movsd %xmm0, %xmm2\n\t"
        "movlhps %xmm1, %xmm2\n\t" // %xmm2 - rx, ry

        "mulsd %xmm0, %xmm0\n\t"
        "mulsd %xmm1, %xmm1\n\t"
        "addsd %xmm1, %xmm0\n\t"
        "sqrtsd %xmm0, %xmm0\n\t" // %xmm0 - dist

        "movsd mu(%rip), %xmm1\n\t"
        "divsd %xmm0, %xmm1\n\t"
        "divsd %xmm0, %xmm1\n\t"
        "divsd %xmm0, %xmm1\n\t"
        "movlhps %xmm1, %xmm1\n\t" // %xmm1 - a / dist, a / dist
        "movsd %xmm1, a(%rip)\n\t" // %xmm0 - dt, dt

        "movsd dt(%rip), %xmm0\n\t"
        "movlhps %xmm0, %xmm0\n\t"

        "movups %xmm2, %xmm4\n\t" // %xmm4 - rx, ry
        "mulpd %xmm1, %xmm2\n\t"
        "mulpd %xmm0, %xmm2\n\t" // %xmm2 - ax * dt, ay * dt

        "movsd vx(%rip), %xmm3\n\t"
        "movhpd vy(%rip), %xmm3\n\t"
        "addpd %xmm2, %xmm3\n\t" // %xmm3 - vx, vy

        "movsd %xmm3, vx(%rip)\n\t"
        "movhpd %xmm3, vy(%rip)\n\t"

        "mulpd %xmm0, %xmm3\n\t"
        "addpd %xmm3, %xmm4\n\t" // %xmm4 - rx, ry

        "movsd %xmm4, rx(%rip)\n\t"
        "movhpd %xmm4, ry(%rip)"
    );
}

```

Рис. 10

Странно, но оно работает. Да еще и обгоняет оптимизации. Правда при -O2 моя функция работать перестала и я ее закомментил.

asm: Time is 136300 ns rx = 6.96641e+10 C++: Time is 181400 ns rx = 6.96641e+10 -O0	asm: Time is 90800 ns rx = 6.96641e+10 C++: Time is 106100 ns rx = 6.96641e+10 -O1	asm: Time is 200 ns rx = 1.496e+11 C++: Time is 100100 ns rx = 6.96641e+10 -O2
--	---	---

Рис. 11