Описание задачи

Пусть у нас имеется некоторый вектор $x \in R^n$ и пусть x_i - $i-ая$ компонента вектора x. Необходимо вычислить

$$a = \prod_{i=1}^{n} x_i$$

Предварительные изменения исходного кода

- Увеличено число элементов вектора до 1 000 000 000
- Изменён способ генерации чисел, чтобы при их перемножении мы не выходили за пределы double или не получали в результате ноль

  v.data[i] = 1.0 + ((double)rand() - 0.5) / RAND_MAX / 1e9;
- Добавлены функции, измеряющие время и печатающие результат работы программы
- Сменены 32 битные целые числа на 64 битные

Результат работы исходного кода

CMakeLists.txt

```
project(optimization)
cmake_minimum_required(VERSION 3.9)

find_package(OpenMP)
if (OPENMP_FOUND)
    set (CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
    set (CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
    set (CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} ${OpenMP_EXE_LINKER_FLAGS}")
endif()

add_executable(test_multiply main.c)
```

```
moskalenkoviktor@moskalenkoviktor-Aspire-E5-571G:~/Документы/Study/6_1/optimization/build$ cmake ..
-- The C compiler identification is GNU 7.2.0
-- The CXX compiler identification is GNU 7.2.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moskalenkoviktor/Документы/Study/6_1/optimization/build
```

Result — результат перемножения элементов вектора
Time — время выполнения умножения

```
Result - 1.648737
Time - 4.221450 sec
```

Оптимизация 1 (оптимизации компилятора)

Меняем **Debug** на **RelWithDebInfo**

Результаты

```
moskalenkoviktor@moskalenkoviktor-Aspire-E5-571G:~/Документы/Study/6_1/optimization/build$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo ..
-- The C compiler identification is GNU 7.2.0
-- The CXX compiler identification is GNU 7.2.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moskalenkoviktor/Документы/Study/6_1/optimization/build
```

```
Result - 1.648711
Time - 1.973709 sec
```

Оптимизация 2 (смена компилятора)

Меняем компилятор с **GNU** на **Intel**

Результаты

```
moskalenkoviktor@moskalenkoviktor-Aspire-E5-571G:~/Документы/Study/6_1/optimization/build$ cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DCMAKE_C_C
OMPILER=icc -DCMAKE_CXX_COMPILER=icpc ..
-- The C compiler identification is Intel 19.0.0.20181018
-- The CXX compiler identification is Intel 19.0.0.20181018
-- Check for working C compiler: /home/moskalenkoviktor/intel/bin/icc
-- Check for working C compiler: /home/moskalenkoviktor/intel/bin/icc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /home/moskalenkoviktor/intel/bin/icpc
-- Check for working CXX compiler: /home/moskalenkoviktor/intel/bin/icpc -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -qopenmp
-- Found OpenMP_CXX: -qopenmp
-- Configuring done
-- Generating done
-- Build files have been written to: /home/moskalenkoviktor/Документы/Study/6_1/optimization/build
```

```
Result - 1.648706
Time - 0.975949 sec
```

Попытка оптимизации 1 (проверка векторизации)

Проверяем векторизацию основного цикла программы

Поиск проблемы

Заходим на https://godbolt.org/, вставляем наш исходный код и выбираем компилятор x86-64 icc 19.0.0

```
19  void multiply(my_vector *v, double *res)
20  {
21      long i;
22      *res = 1;
23      for (i = 0; i < my_vector_len(v); i++) {
24          *res *= v->data[i];
25      }
26  }
```

Результаты

```
167         inc       rdi                                             #23.21
168         cmp       rdi, r8                                         #23.21
169         jb        ..B3.12       # Prob 82%                        #23.21
170 ..B3.14:                        # Preds ..B3.12 ..B3.10
171         movhpd    xmm0, QWORD PTR .L_2il0floatpacket.5[rip]       #24.10
172 ..B3.15:                        # Preds ..B3.15 ..B3.14
173         mulpd     xmm0, XMMWORD PTR [rax+r8*8]                    #24.10
174         mulpd     xmm0, XMMWORD PTR [16+rax+r8*8]                 #24.10
175         mulpd     xmm0, XMMWORD PTR [32+rax+r8*8]                 #24.10
176         mulpd     xmm0, XMMWORD PTR [48+rax+r8*8]                 #24.10
177         add       r8, 8                                           #23.21
178         cmp       r8, rcx                                         #23.21
179         jb        ..B3.15       # Prob 82%                        #23.21
180         movaps    xmm1, xmm0                                      #24.10
181         unpckhpd  xmm1, xmm0                                      #24.10
182         mulsd     xmm0, xmm1                                      #24.10
183 ..B3.17:                        # Preds ..B3.16 ..B3.29
184         cmp       rcx, rdx                                        #23.21
185         jae       ..B3.21       # Prob 9%                         #23.21
186 ..B3.19:                        # Preds ..B3.17 ..B3.19
```

Смотрим во что скомпилировалась наша функцию, видим что уже присутствуют векторные инструкции **mulpd**. Компилятор сам справился с векторизацией.

Попытка оптимизации 2 (проверка в VTune)

Поиск проблемы

Открываем **Intel VTune Amplifier**, выбираем наш исполняемый файл. Запускаем микроархитектурный анализ, смотрим на функцию **multiply**:

µPipe

Retiring:	**6.5%**	**of Pipeline Slots**
Front-End Bound:	**1.0%**	**of Pipeline Slots**
Bad Speculation:	**0.0%**	**of Pipeline Slots**
Back-End Bound:	**92.5%**	**of Pipeline Slots**
Memory Bound:	**65.4%**	**of Pipeline Slots**
L1 Bound:	**10.0%**	**of Clockticks**
DTLB Overhead:	2.0%	of Clockticks
Loads Blocked by Store Forwarding:	0.0%	of Clockticks
Lock Latency:	0.0%	of Clockticks
Split Loads:	0.0%	of Clockticks
4K Aliasing:	0.0%	of Clockticks
FB Full:	0.0%	of Clockticks
L2 Bound:	10.0%	of Clockticks
L3 Bound:	**6.7%**	**of Clockticks**
Contested Accesses:	0.0%	of Clockticks
Data Sharing:	0.0%	of Clockticks
L3 Latency:	100.0%	of Clockticks
SQ Full:	4.0%	of Clockticks
DRAM Bound:	**31.4%**	**of Clockticks**
Memory Bandwidth:	32.1%	of Clockticks
Memory Latency:	**56.2%**	**of Clockticks**
Store Bound:	**0.0%**	**of Clockticks**
Core Bound:	**27.1%**	**of Pipeline Slots**
Divider:	0.0%	of Clockticks

Спускаемся через самые большие цифры к **Memory Latency**. Проверим пропускную способность DRAM в бенчмарке:

```
moskalenkoviktor@moskalenkoviktor-Aspire-E5-571G:~$ mbw 4096 -t1
Long uses 8 bytes. Allocating 2*536870912 elements = 8589934592 bytes of memory.
Getting down to business... Doing 10 runs per test.
0       Method: DUMB    Elapsed: 0.52710        MiB: 4096.00000 Copy: 7770.777 MiB/s
1       Method: DUMB    Elapsed: 0.52906        MiB: 4096.00000 Copy: 7741.975 MiB/s
2       Method: DUMB    Elapsed: 0.52811        MiB: 4096.00000 Copy: 7755.901 MiB/s
3       Method: DUMB    Elapsed: 0.52565        MiB: 4096.00000 Copy: 7792.198 MiB/s
4       Method: DUMB    Elapsed: 0.55435        MiB: 4096.00000 Copy: 7388.874 MiB/s
5       Method: DUMB    Elapsed: 0.52971        MiB: 4096.00000 Copy: 7732.547 MiB/s
6       Method: DUMB    Elapsed: 0.52949        MiB: 4096.00000 Copy: 7735.687 MiB/s
7       Method: DUMB    Elapsed: 0.52661        MiB: 4096.00000 Copy: 7778.052 MiB/s
8       Method: DUMB    Elapsed: 0.52905        MiB: 4096.00000 Copy: 7742.223 MiB/s
9       Method: DUMB    Elapsed: 0.52677        MiB: 4096.00000 Copy: 7775.631 MiB/s
AVG     Method: DUMB    Elapsed: 0.53059        MiB: 4096.00000 Copy: 7719.685 MiB/s
```

Скорость передачи данных в нашей задаче:

$$\frac{8 * 1000000000\,байт}{0.975949\,сек} \approx 7817.411\,\frac{Мбайт}{сек}$$

Результаты

Мы упёрлись в пропускную способность оперативной памяти

Оптимизация 3 (параллелизм)

Добавим прагму перед циклом и запустим программу в два потока

```
void multiply(my_vector *v, double *res)
{
    long i;
    double tmp_res = 1;
    #pragma omp parallel for reduction(* : tmp_res)
    for (i = 0; i < my_vector_len(v); i++) {
        tmp_res *= v->data[i];
    }
    *res = tmp_res;
}
```

Результаты

```
Result - 1.648704
Time - 0.573526 sec
```

Получили ускорение в 1.7 раз