Отчет по лабораторной работе.

Задача оптимизации перемножения элементов вектора.

Выполнила студентка группы 381706-2м(3м),

Реунова Ольга

Нижний Новгород

2018 г

Оглавление

[Описание задачи. Алгоритм. 3](#_Toc532162642)

[Оптимизация 4](#_Toc532162643)

[Hardware: 4](#_Toc532162644)

[Применение оптимизации 4](#_Toc532162645)

[Выводы. 7](#_Toc532162646)

# Описание задачи. Алгоритм.

В данной лабораторной работе необходимо реализовать базовый алгоритм перемножения элементов вектора и оптимизировать его.

Базовый алгоритм:

#include <iostream>

#include <omp.h>

#define LENGTH 20000

typedef struct {

long len;

double \*data;

} my\_vector;

long my\_vector\_len(my\_vector \*v) {

return v->len;

}

void foo\_bar(my\_vector \*v, double \*res)

{

long i;

\*res = 1;

for (i = 0; i < my\_vector\_len(v); i++) {

\*res \*= v->data[i];

}

}

double a[LENGTH];

my\_vector v = { LENGTH, a };

int main() {

double res;

for (i = 0; i < LENGTH; i++)

a[i] = i/2+1;

double start = omp\_get\_wtime();

for (int j=0; j<100;j++)

foo\_bar(&v,&res);

double finish = omp\_get\_wtime();

std::cout << "time: " << (finish - start)/100 << std::endl;

std::cout << "res: " << res << std::endl;

}

# Оптимизация

Для оптимизации алгоритма будем пробовать использовать Intel VTune Amplifier, openmp и code review. Для получения наилучшего результата будем выполнять оптимизацию постепенно, делая анализ результатов.

Заполнение вектора выполняется не рандомно, а по формуле. Такой подход обеспечивает сохранение значений элементов вектора от запуска к запуску.

Время буду замерять с помощью функции omp\_get\_wtime() и усреднять по 100 запускам алгоритма.

## Hardware:

* Процессор: Intel Core i7-4510U CPU @ 2.00GHz - 2.60 GHz
* ОЗУ: 12 Гб
* Ядра – 2, логические процессоры – 4
* Кэши: L1-128 Кб, L2 – 512 Кб, L3 – 4 Мб

## Применение оптимизации

1. Продолжительность работы базового алгоритма: 3.95928e-05
2. На первом этапе была замечена неэффективная работа цикла, выполняющего перемножение. Было принято решение вынести my\_vector\_len(v) из цикла.

void foo\_bar(my\_vector \*v, double \*res)

{

long i, len = my\_vector\_len(v);

\*res = 1;

for (i = 0; i < len; i++) {

\*res \*= v->data[i];

}

}

Время: 3.96138e-05.

Время почти не изменилось, потому что компилятор оказался умным и смог сам выполнить эту оптимизацию.

1. Рассмотрим переменную res - она поступает в функцию из основной программы по указателю и используется внутри цикла. Из-за нее происходят частые обращения к памяти (мы считываем ее значение из памяти, считываем элемент массива, выполняем перемножение, записываем новый результат в память).

От нее можно отказаться: ввести временный буфер tmp для подсчета результата и изменить вид функции.

double foo\_bar(my\_vector \*v)

{

long i, len = my\_vector\_len(v);

double tmp = 1;

for (i = 0; i < len; i++) {

tmp \*= v->data[i];

}

return tmp;

}

Время: 3.60598e-05

1. Можно заметить, что цикл можно разбить на части и выполнять эти части параллельно, после чего перемножить результаты частей (раскрутка цикла) с несколькими буферами. Это поможет эффективнее использовать конвейер вычислений.
   1. Начнем с 2 буферов. Введем два временных буфера. Поскольку у меня четное количество компонент вектора, дополнительных циклов на обработку остатка не требуется.

double foo\_bar(my\_vector \*v)

{

long i, len = my\_vector\_len(v);

double tmp = 1, tmp2 = 1;

for (i = 0; i < lim; i += 2) {

tmp \*= v->data[i];

tmp2 \*= v->data[i + 1];

}

return tmp \* tmp2;

}

Время: 2.61898e-05. Уже неплохо (ускорились в 1.5 раза по сравнению с базой).

* 1. Попробуем раскрутить цикл с использованием 4 буферов. Здесь необходимо обработать остаток вектора (поскольку увеличился шаг обхода вектора).

double foo\_bar(my\_vector \*v)

{

long i, len = my\_vector\_len(v);

double tmp = 1, tmp2 = 1, tmp4 = 1, tmp3 = 1;

long lim = len - 3;

for (i = 0; i < lim; i += 4) {

tmp \*= v->data[i];

tmp2 \*= v->data[i + 1];

tmp3 \*= v->data[i + 2];

tmp4 \*= v->data[i + 3];

}

while (i < len)

{

tmp \*= v->data[i];

i++;

}

return tmp \* tmp2\*tmp3\*tmp4;

}

Время: 1.84009e-05 (ускорение в 2.12 раза)

* 1. Теперь 8. Также добавляем цикл обработки остатка вектора.

double foo\_bar(my\_vector \*v)

{

long i, len = my\_vector\_len(v);

double tmp = 1, tmp2 = 1, tmp3 = 1, tmp4 = 1, tmp5 = 1, tmp6 = 1, tmp7 = 1, tmp8 = 1;

long lim = len - 3;

for (i = 0; i < lim; i += 8) {

tmp \*= v->data[i];

tmp2 \*= v->data[i + 1];

tmp3 \*= v->data[i + 2];

tmp4 \*= v->data[i + 3];

tmp5 \*= v->data[i + 4];

tmp6 \*= v->data[i + 5];

tmp7 \*= v->data[i + 6];

tmp8 \*= v->data[i + 7];

}

while (i < len)

{

tmp \*= v->data[i];

i++;

}

return tmp\*tmp2\*tmp3\*tmp4\*tmp5\*tmp6\*tmp7\*tmp8;

}

Время: 1.27665e-05 (но некоторые запуски даже показывали время 1.02304e-05). Ускорение больше чем в 3 раза.

# Выводы.

Путем оптимизации обращения к памяти и использованием 8 буферов удалось достичь ускорения работы алгоритма по времени в среднем в 3 раза.