ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

«Оптимизация доступа к памяти.»

по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. ИП-815
Маслов Роман Сергеевич
Проверил: ст. преп. Кафедры ВС
Токмашева Елизавета Владимировна

Новосибирск 2020

Содержание

Постановка задачи **3**

Выполнение работы **5**

Результат работы **6**

Приложение **8**

Постановка задачи

1. На языке C/C++/C# реализовать функцию DGEMM BLAS последовательное умножение двух

квадратных матриц с элементами типа double. Обеспечить возможность задавать

размерности матриц в качестве аргумента командной строки при запуске программы.

Инициализировать начальные значения матриц случайными числами.

2. Провести серию испытаний и построить график зависимости времени выполнения

программы от объёма входных данных. Например, для квадратных матриц с числом

строк/столбцов 1000, 2000, 3000, ... 10000.

3. Оценить предельные размеры матриц, которые можно перемножить на вашем

вычислительном устройстве.

4. Реализовать дополнительную функцию DGEMM_opt_1, в которой выполняется

оптимизация доступа к памяти, за счет построчного перебора элементов обеих матриц.

5. * Реализовать дополнительную функцию DGEMM_opt_2, в которой выполняется

оптимизация доступа к памяти, за счет блочного перебора элементов матриц. Обеспечить

возможность задавать блока, в качестве аргумента функции.

6. ** Реализовать дополнительную функцию DGEMM_opt_3, в которой выполняется

оптимизация доступа к памяти, за счет векторизации кода.

7. Оценить ускорение умножения для матриц фиксированного размера, например,

1000x1000, 2000x2000, 5000x5000, 10000x10000.

* Для блочного умножения матриц определить размер блока, при котором достигается

максимальное ускорение.

8. С помощью профилировщика для исходной программы и каждого способа оптимизации

доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).

9. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

Выполнение работы

Скрипт run.sh запускает программу dgemm.cpp несколько раз с различными параметрами: размерность, тип оптимизации и блок данных. Функция dgemm выполняет обыкновенное умножение матриц и записывает время выполнения в файл(как и все другие). Функция dgemm_opt_1 в отличии от предыдущей меняет циклы местами таким образом, чтобы элементы в каждой матрице переберались последовательно(без скачков из строчки на строчку). Следующая функция dgemm_opt_3 выполняет умножение небольшими блоками данных(квадратными кусочками), в ходе выполнения работы было выявлено, что оптимальный размер блока составляет размер кэш памяти деленный на размер переменной(в моем случае 1024 / 8 = 128). В другой программе(dgemm_opt_3), скомпилированной с опциями `-ftree-vectorize` и `-ffast-math` для векторизации циклов, выполняющих операции над массивами с плавающей точкой. Векторизация подразумевает выгрузку всех векторов из цикла в оперативную память и одновременное применение к ним одной и той же операции(благодаря архитектуре simd). После сбора статистики с каждой из этих функций с различными размерностями матриц, каждая запускается еще раз с помощью утилиты perf для вычисления числа кэш промахов. Потом по всем полученным данным строятся графики.

Результат работы

Вывод диаграммы иллюстрирующей зависимость времени выполнения от размерности матрицы представлен на рисунке 1.

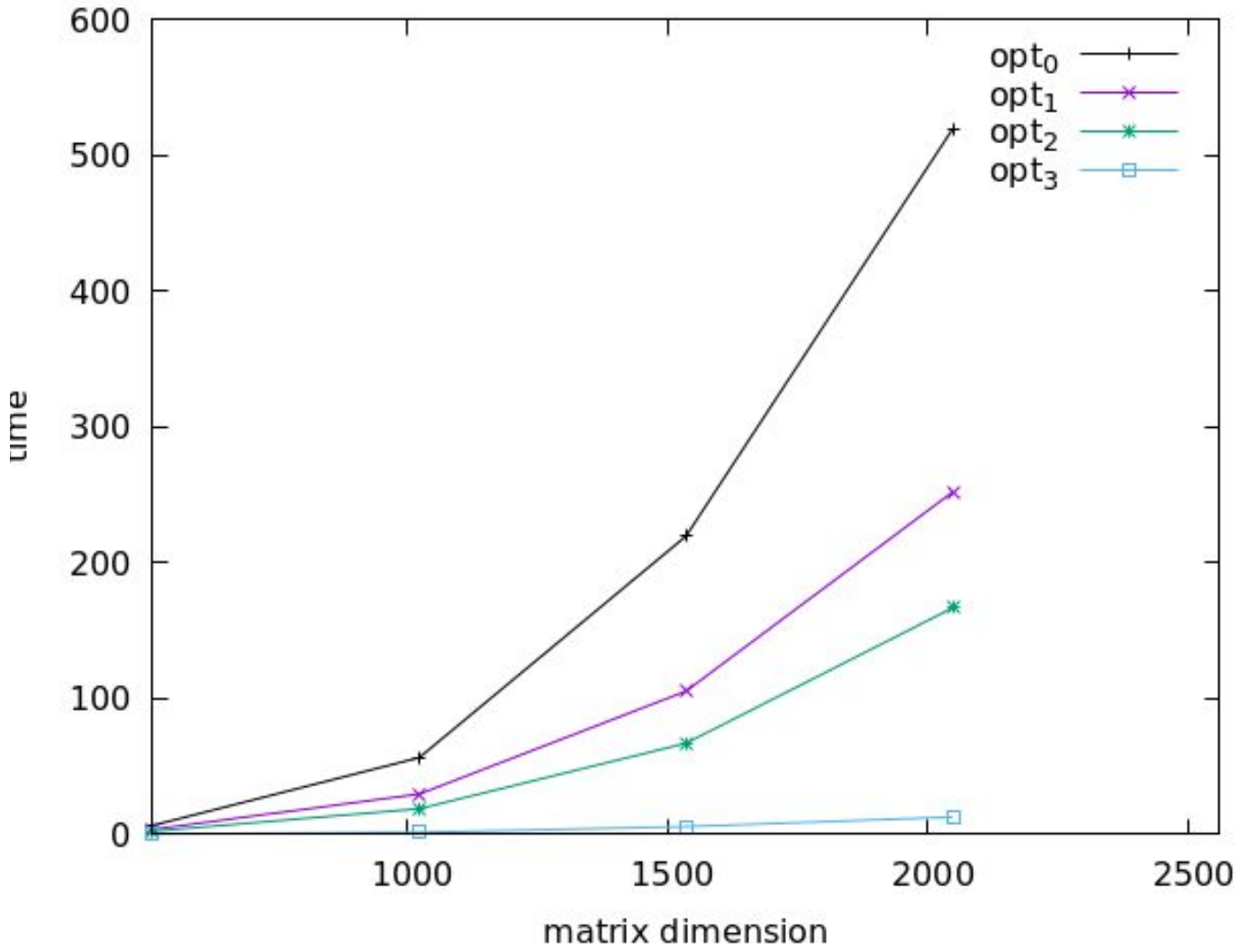

Рисунок 1. Диаграмма иллюстрирующая зависимость времени выполнения от размерности матрицы.

С помощью диаграммы становится видно, что от увеличения размерности матрицы время выполнения растет очень сильно. Также графики показывают различие времени выполнения в зависимости от типа оптимизации, можно сказать, они идут по убыванию(от времени выполнения) от dgemm до dgemm_opt_3.

Вывод диаграммы иллюстрирующей зависимость количества кэш промахов от типа оптимизации рисунке 2.

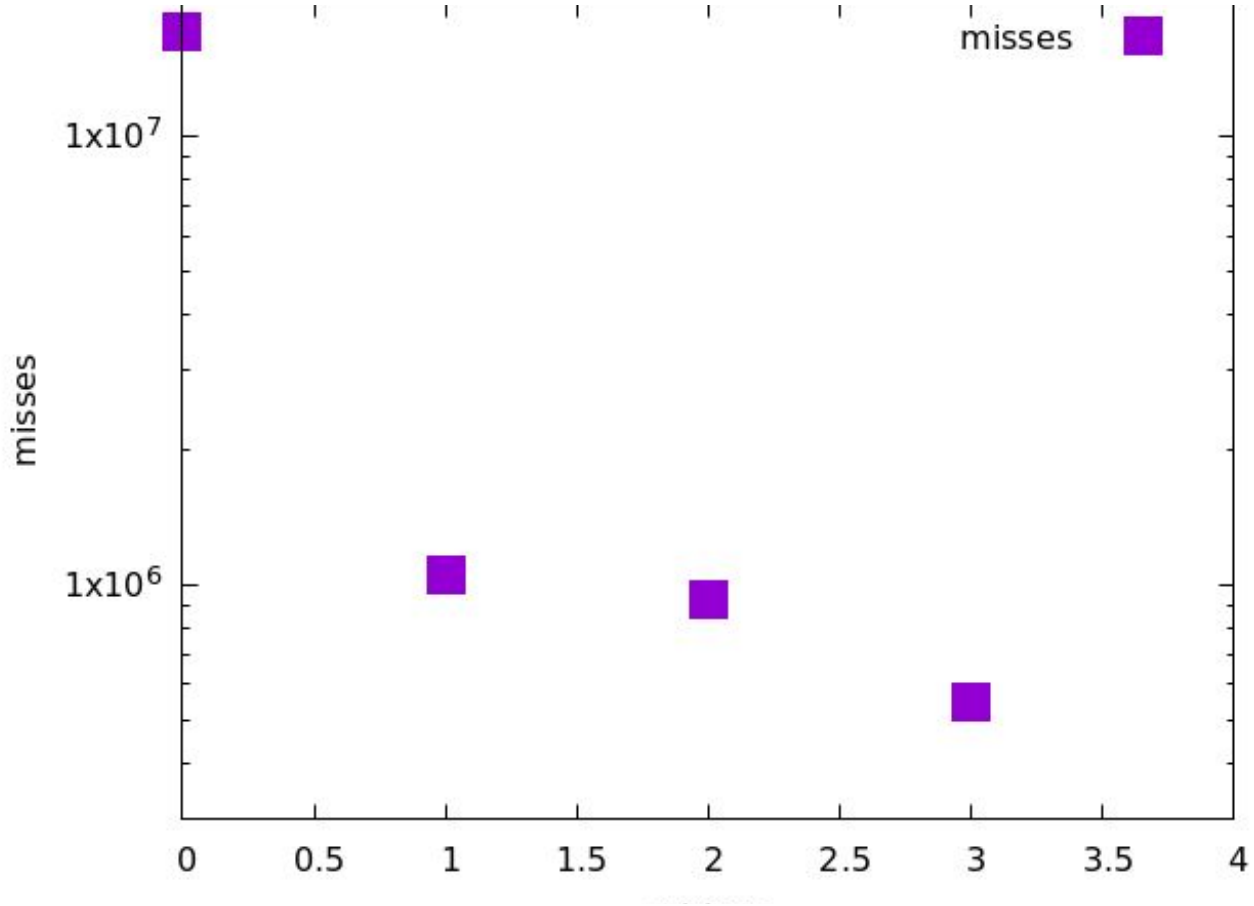

Рисунок 2. Диаграмма иллюстрирующая зависимость количества кэш промахов от типа оптимизации.

С помощью диаграммы становится видно, что без какой либо оптимизации кол-во кэш промахов очень велико, с первой оптимизацией уже на несколько порядков меньше, со второй их примерно столько же сколько и с первой, а с третьей их уже очень мало(примерно в два раза меньше чем на первом уровне оптимизации).

Приложение

"run.sh"

```
#!/bin/bash

bs=128

sudo apt-get install -y gnuplot

g++ -std=c++2a dgemm.cpp -o dgemm
g++ -std=c++2a -O3 dgemm_opt_3.cpp -o dgemm_opt_3

for it in {512..2048..512}
do
	for jt in {0..2..1}
	do
		./dgemm $it $jt $bs
	done
	./dgemm_opt_3 $it
done

sudo perf stat -e cache-misses ./dgemm 512 0 128 2> a
cat a | echo $(egrep -m 1 'cache-misses') | awk '{printf "%s\t0\n", $1}' >> misses
sudo perf stat -e cache-misses ./dgemm 512 1 128 2> a
cat a | echo $(egrep -m 1 'cache-misses') | awk '{printf "%s\t1\n", $1}' >> misses
sudo perf stat -e cache-misses ./dgemm 512 2 128 2> a
cat a | echo $(egrep -m 1 'cache-misses') | awk '{printf "%s\t2\n", $1}' >> misses
sudo perf stat -e cache-misses ./dgemm_opt_3 512 2> a
cat a | echo $(egrep -m 1 'cache-misses') | awk '{printf "%s\t3\n", $1}' >> misses

gnuplot plot
```

"dgemm.cpp"

```
#include <vector>
#include <chrono>
#include <iostream>
#include <random>
#include <string>
#include <cmath>
#include <iomanip>
#include <fstream>

unsigned long dim;
unsigned short opt;
unsigned long bs;

long double dgemm(std::vector<std::vector<double>>& a,
std::vector<std::vector<double>>& b,
		std::vector<std::vector<double>>& c) {
	auto start = std::chrono::system_clock::now();
	for (unsigned long it = 0; it < a.size(); ++it)
		for (unsigned long jt = 0; jt < a.size(); ++jt)
			for (unsigned long et = 0; et < a.size(); ++et)
				c[it][jt] += a[it][et] * b[et][jt];
```

```
	auto end = std::chrono::system_clock::now();
	std::chrono::duration<long double> difference = end - start;
	std::ofstream out("opt_0", std::ios_base::app);
	out << std::fixed << difference.count() << "\t" << dim << std::endl;
	return difference.count();
}

long double dgemm_opt_1(std::vector<std::vector<double>>& a,
std::vector<std::vector<double>>& b,
		std::vector<std::vector<double>>& c) {
	auto start = std::chrono::system_clock::now();
	for (unsigned long it = 0; it < a.size(); ++it)
		for (unsigned long et = 0; et < a.size(); ++et)
			for (unsigned long jt = 0; jt < a.size(); ++jt)
				c[it][jt] += a[it][et] * b[et][jt];
	auto end = std::chrono::system_clock::now();
	std::chrono::duration<long double> difference = end - start;
	std::ofstream out("opt_1", std::ios_base::app);
	out << std::fixed << difference.count() << "\t" << dim << std::endl;
	return difference.count();
}

long double dgemm_opt_2(std::vector<double>& a, std::vector<double>& b,
std::vector<double>& c) {
	auto start = std::chrono::system_clock::now();
	for (unsigned long it = 0; it < dim; it += bs)
		for (unsigned long jt = 0; jt < dim; jt += bs)
			for (unsigned long et = 0; et < dim; et += bs)
				for (unsigned long it0 = 0; it0 < bs; ++it0)
					for (unsigned long et0 = 0; et0 < bs; ++et0)
						for (unsigned long jt0 = 0; jt0 < bs; ++jt0)
							c[(it0 + it) * dim + jt0 + jt] +=
a[(it0 + it) * dim + et0 + et] *
b[(et0 + et) * dim + jt0 + jt];
	auto end = std::chrono::system_clock::now();
	std::chrono::duration<long double> difference = end - start;
	std::ofstream out("opt_2", std::ios_base::app);
	out << std::fixed << difference.count() << "\t" << dim << std::endl;
	return difference.count();
}

int main(int argc, char *argv[]) {
	dim = std::stol(std::string(argv[1]));
	opt = std::stol(std::string(argv[2]));
	bs = std::stol(std::string(argv[3]));
	std::vector<std::vector<double>> a(dim, std::vector<double>(dim)), b(dim,
std::vector<double>(dim)),
		c(dim, std::vector<double>(dim));
	std::vector<double> a0(dim * dim), b0(dim * dim), c0(dim * dim);
	std::random_device rd;
	std::mt19937 rng(rd());
	std::normal_distribution<double> dist;
	for (auto& it : a)
		for (auto& _ : it)
			_ = dist(rng);
	for (auto& it : b)
		for (auto& _ : it)
			_ = dist(rng);
	for (auto& _ : a0)
		_ = dist(rng);
	for (auto& _ : b0)
```

```
            _ = dist(rng);
      long double time;
      if (opt == 0)
            time = dgemm(a, b, c);
      else if (opt == 1)
            time = dgemm_opt_1(a, b, c);
      else if (opt == 2)
            time = dgemm_opt_2(a0, b0, c0);
      std::cout << time << std::endl;
}
```

"dgemm_opt_3.cpp"

```
#include <vector>
#include <chrono>
#include <iostream>
#include <random>
#include <string>
#include <cmath>
#include <iomanip>
#include <fstream>

unsigned long dim;
unsigned short opt;
unsigned long bs;

long double dgemm_opt_3(std::vector<double>& a, std::vector<double>& b,
std::vector<double>& c) {
      auto start = std::chrono::system_clock::now();
      for (unsigned long it = 0; it < dim; ++it)
            for (unsigned long et = 0; et < dim; ++et)
                  for (unsigned long jt = 0; jt < dim; ++jt)
                        c[it * dim + jt] += a[it * dim + et] * b[et * dim + jt];
      auto end = std::chrono::system_clock::now();
      std::chrono::duration<long double> difference = end - start;
      std::ofstream out("opt_3", std::ios_base::app);
      out << std::fixed << difference.count() << "\t" << dim << std::endl;
      return difference.count();
}

int main(int argc, char *argv[]) {
      dim = std::stol(std::string(argv[1]));
      std::vector<double> a0(dim * dim), b0(dim * dim), c0(dim * dim);
      std::random_device rd;
      std::mt19937 rng(rd());
      std::normal_distribution<double> dist;
      for (auto& _ : a0)
            _ = dist(rng);
      for (auto& _ : b0)
            _ = dist(rng);
      long double time;
      time = dgemm_opt_3(a0, b0, c0);
      std::cout << time << std::endl;
}
```

"plot.plg"

```
set term pngcairo
```

```
################################################################################

set xlabel "matrix dimension"
set ylabel "time"

#set y axis urself
ymin = 0
ymax = 600
xmin = 512
xmax = 2560
set yrange [ymin:ymax]
unset autoscale y
set xrange [xmin:xmax]
unset autoscale x

set output "times.png"
plot 'opt_0' u 2:1 t 'opt_0' w lp lc 0, \
      'opt_1' u 2:1 t 'opt_1' w lp lc 1, \
      'opt_2' u 2:1 t 'opt_2' w lp lc 2, \
      'opt_3' u 2:1 t 'opt_3' w lp lc 3

################################################################################

set xlabel "opt type"
set ylabel "misses"

#set y axis urself
ymin = 100000
ymax = 20000000
xmin = 0
xmax = 4
set yrange [ymin:ymax]
unset autoscale y
set xrange [xmin:xmax]
unset autoscale x

set output "cache_misses.png"
plot 'misses' u 2:1 ps 3 pt 5
```