Федеральное государственное бюджетное образовательное учреждения высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра АЭС

Лабораторная работа № 4

По дисциплине: «Архитектура вычислительных систем»

Выполнил: студент III курса

ИВТ, гр. ИП-712

Рещиков А.Е.

Проверил: /\*\*/

Токмашева Е.И.

Новосибирск 2019

[**Saxpy blas функция**](#_z1jbiray4jiq) **3**

[Листинг программы saxpy blas](#_p5a12k54wvua) 3

[Оценить предельные размеры векторов, которые возможно перемножить на одном ПК](#_do7h5opaq7hl) 4

[Построить график зависимости времени выполнения программы от объема входных данных](#_usvyxum17un5) 4

[**Dgemm blas функция**](#_j1v3x9phyl6k) **5**

[Листинг программы dgemm blas](#_lzoks2qag5mh) 5

[Оценить предельные размеры матриц, которые можно перемножить на одном ПК](#_cp8fls2gehnk) 6

[Построить график зависимости времени выполнения программы от объемов входных данных](#_r7lpid15wpau) 7

# 

# 

# 

# Saxpy blas функция

\*ЗАДАНИЕ\*

## Листинг программы saxpy blas

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <algorithm>

#include <iomanip>

#include <random>

#include <x86intrin.h>

static const int64\_t proc\_tc = 2200000000;

double\_t a = 1.0;

std::vector <double\_t> sumary;

int64\_t saxpy\_blas(std::vector <int64\_t> &X, std::vector <int64\_t> &Y);

int main()

{

std::mt19937 gen(time(nullptr));

std::uniform\_int\_distribution <int64\_t> uid(0, 100000);

int64\_t v\_size = 0;

std::cout << "Vector size - ";

std::cin >> v\_size;

std::cout << "A - ";

//std::cin >> a;

sumary.resize(v\_size, 0);

std::vector <int64\_t> X(v\_size);

std::vector <int64\_t> Y(v\_size);

std::generate(X.begin(), X.end(), std::rand);

std::generate(Y.begin(), Y.end(), std::rand);

/\*std::cout << "X\n";

for(auto &i : X)

std::cout << i << "\n";

std::cout << "\nY\n";

for(auto &i : Y)

std::cout << i << "\n";

\*/saxpy\_blas(X,Y);

/\*std::cout << "\nSumary vector\n";

for(auto &i : sumary)

std::cout << i << " ";

\*/return 0;

}

int64\_t saxpy\_blas(std::vector <int64\_t> &X, std::vector <int64\_t> &Y)

{

int64\_t start = \_\_rdtsc();

for(auto iterator = 0 ; iterator < sumary.size(); iterator++)

sumary[iterator] = X[iterator] \* a + Y[iterator];

std::cout << "\n\nTime(TSC) - " << std::fixed << std::setprecision(10) << (double\_t)(\_\_rdtsc() - start) / proc\_tc;

return 0;

}

## Оценить предельные размеры векторов, которые возможно перемножить на одном ПК

Предельные размеры векторов зависят от объёма оперативной памяти.

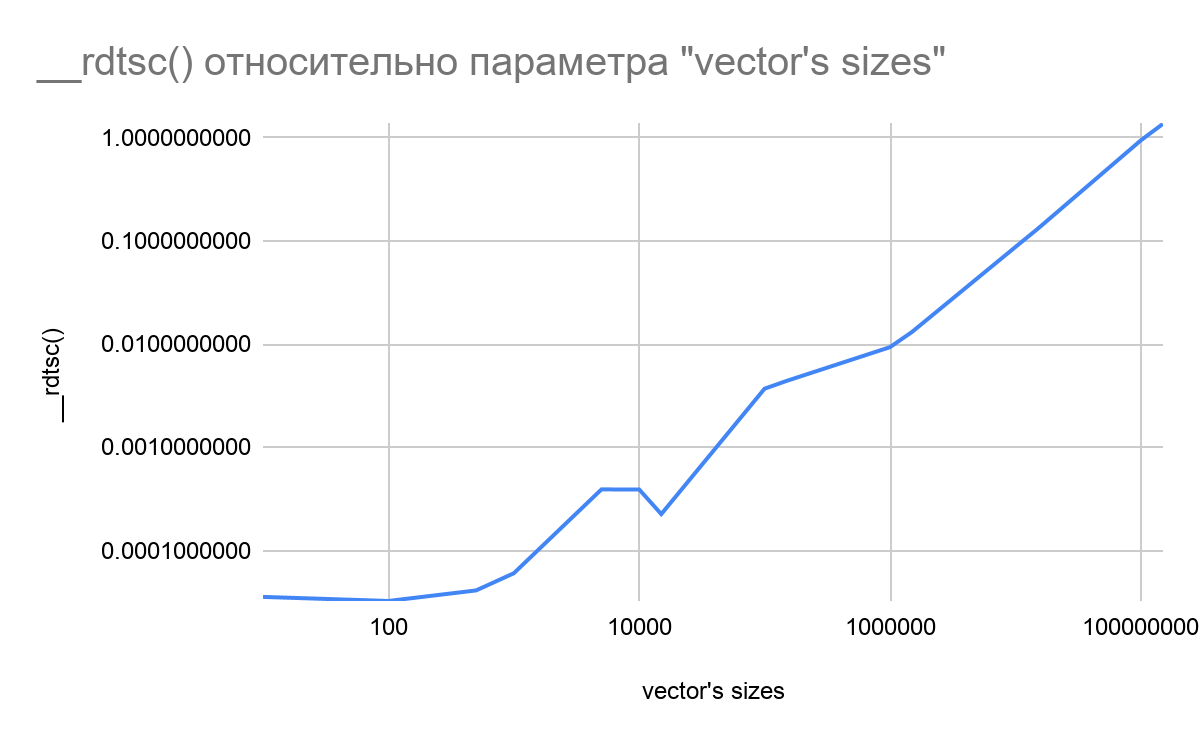
Предельный размер вектора можно вычислить по формуле: , где

* V - размер оперативной памяти в байтах.
* С - память занятая системой и работающей программой, которая не может быть перемещена в файл подкачки. В этой параметре хранится память отдаваемая под IDE(в моем случае) и оболочку системы. Число равно числу порядка 500-800, в связи с чем возьмем среднее значение равное 650 byte.
* S - размер хранимого вектором элемента (int64\_t, 8 byte).

В моём компьютере 6 GB RAM, что равно 6442450944 байт. Подставим известные числа в формулу:

(6442450944 - 650) / 8 =~ 805306286 - предельный размер одного вектора, т.к их 3, два входных и итоговый, то оперативной памяти хватит на 3 вектора длинной в 268 435 428 переменных в каждом.

## Построить график зависимости времени выполнения программы от объема входных данных



# 

# Dgemm blas функция

\*задание\*

## Листинг программы dgemm blas

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <iomanip>

#include <random>

#include <x86intrin.h>

static const int64\_t proc\_tc = 2200000000;

double\_t dgemm\_blas(std::vector<std::vector<double\_t>> &first, std::vector<std::vector<double\_t>> &last, int64\_t f\_col, int64\_t s\_str);

int main()

{

std::mt19937 gen(time(nullptr));

std::uniform\_real\_distribution<double\_t> uid(0.0, 10.0);

int64\_t f\_str = 0, f\_col = 0, s\_str = 0, s\_col = 0;

std::cout << "Matrix size, string for first - ";

std::cin >> f\_str;

std::cout << "Matrix size, column for first - ";

std::cin >> f\_col;

std::cout << "Matrix size, string for second - ";

std::cin >> s\_str;

std::cout << "Matrix size, column for second - ";

std::cin >> s\_col;

std::vector<std::vector<double\_t>> first(f\_str, std::vector<double\_t>(f\_col, 0));

std::vector<std::vector<double\_t>> last(s\_str, std::vector<double\_t>(s\_col, 0));

//std::cout << "First\n";

for (auto i = 0; i < first.size(); i++)

{

for (auto j = 0; j < first[i].size(); j++)

{

first[i][j] = uid(gen);

//std::cout << std::fixed << std::setprecision(2) << std::setfill('0') << std::setw(5) << first[i][j] << "\t";

}

//std::cout << "\n";

}

//std::cout << "Second\n";

for (auto i = 0; i < last.size(); i++)

{

for (auto j = 0; j < last[i].size(); j++)

{

last[i][j] = uid(gen);

//std::cout << std::fixed << std::setprecision(2) << std::setfill('0') << std::setw(5) << last[i][j] << "\t";

}

//std::cout << "\n";

}

if (f\_col != s\_str)

{

std::cout << "Matrix cannot multi\n";

return 0;

}

else

dgemm\_blas(first, last, f\_col, s\_str);

return 0;

}

double\_t dgemm\_blas(std::vector<std::vector<double\_t>> &first, std::vector<std::vector<double\_t>> &last, int64\_t f\_col, int64\_t s\_str)

{

int64\_t start\_time =\_\_rdtsc();

std::vector<std::vector<double\_t>> multi(s\_str, std::vector<double\_t>(f\_col, 0));

//std::cout << "Multi matrix\n";

for(auto & i : multi)

{

for(auto j = 0; j < i.size(); j++)

{

for(auto k = 0; k < f\_col; k++)

i[j] += first[j][k] \* last[k][j];

}

}

/\*for (auto i = 0; i < multi.size(); i++)

{

for (auto j = 0; j < multi[i].size(); j++)

std::cout << std::fixed << std::setprecision(2) << std::setfill('0') << std::setw(5) << multi[i][j] << "\t";

std::cout << "\n";

}\*/

std::cout << "\nTime of matrix multi - " << std::fixed << std::setprecision(10) << (double\_t)(\_\_rdtsc() - start\_time) / proc\_tc;

return 0;

}

## Оценить предельные размеры матриц, которые можно перемножить на одном ПК

Стандартная функция перемножения 2 матриц достаточно проста, однако алгоритм перемножения имеет 3 цикла, 2 из которых вложены в основной, что обеспечивает алгоритму dgemm blas кубическую трудоемкость O(n^3), что на казалось бы даже маленьких значениях вроде 10 000 занимает большой промежуток времени, что доказывает график. Так же фактор, что исполнение программы производится на одном ядре процессора еще более утяжеляет процесс, сводя скорость программы к очень низкой.

2200 МГц × 1 ядро × 4×10-6 = 8,8 млрд операций в секунду.

8 000 000 000 - количество итераций для n = 2000, в каждой итерации помимо действия перемножения происходят изменения счетчиков, которые имеют также 8 000 000 000 разных значения в цикле итераций.

В результате этого и других факторов процессор попросту не успевает вычитывать такие масштабное количество операций.

## Построить график зависимости времени выполнения программы от объемов входных данных

