ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ВТ

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

«Оценка характеристик персонального компьютера» по дисциплине «Архитектура вычислительных систем»

Выполнил: студент гр. АММ2-24

Ириков Евгений Алексеевич

Проверил: к.т.н., доцент Кафедры ВТ Перышкова Евгения Николаевна

Новосибирск 2024

Содержание

Постановка задачи

Задание: *Реализовать программу для оценки производительности процессора (benchmark).*

1. Написать программу(ы) (benchmark) на языке C/C++/C# для оценки производительности процессора. В качестве набора типовых задач использовать либо минимум 3 функции выполняющих математические вычисления, либо одну функцию по работе с матрицами и векторами данных с несколькими типами данных.
2. С помощью системного таймера или с помощью процессорного регистра счетчика TSC реализовать оценку в секундах среднего времени испытания каждой типовой задачи. Оценить точность и погрешность (абсолютную и относительную) измерения времени.
3. Результаты испытаний в самой программе (или с помощью скрипта) сохранить в файл в формате CSV.
4. ***** Оценить среднее время испытания каждой типовой задачи с разным типом входных данных (целочисленные, с одинарной и двойной точностью).
5. ****** Оценить среднее время испытания каждой типовой задачи с оптимизирующими преобразования исходного кода компилятором (ключи –O1, O2, O3 и др.).
6. ******* Оценить и постараться минимизировать накладные расходы(время на вызов функций, влияние загрузки системы и т.п.) при испытании, то есть добиться максимальной точности измерений.
7. Построить сводную диаграмму производительности в зависимости от задач и выбранных исходных параметров испытаний. Оценить среднее быстродействие (производительность) для равновероятного использования типовых задач.

Выполнение работы

В качестве целевого языка для написания тестов производительности был выбран язык C++. В роли типовых задач были выбраны 3 функции нахождения exp, cos и ln с определённой точностью используя ряд Тэйлора.

За само тестирование отвечали 3 функции void Clock_exp(), void Clock_cos(), void Clock_ln(). Каждая соответствовала своей типовой задаче. Последовательность действий функций:

1. Получение определённой константы на вход
2. Начало тестирования
3. Засечь время старта испытаний с помощью системного таймера.
4. Непосредственное выполнение испытаний.
5. Засечь время окончания испытаний.
6. Объединение времени выполнений всех испытаний для получения времени выполнения теста
7. Сохранить результат для данного запуска.
8. Расчёт абсолютной и относительной погрешностей.

После выполнения тестов для получения модели процессора использовались команды bash, изученные в предыдущей лабораторной работе, а также функция popen(), позволяющая получить результат работы bash команд не используя bash-скрипт.

Затем все необходимые данные сохраняются в csv файл используя базовую библиотеку для работы с файлами:

1. создаётся объект класса ofstream
2. объект связывается с файлом “bench result.csv”
3. устанавливается режим для открытия и записи в конец файла
4. все данные записываются в определённом стандарте scv файлов

Чтобы оценить среднее время испытания каждой типовой задачи с разным типом входных данных и ключами оптимизации программа выполнялась несколько раз.

Результат работы

Запуск программы

```
ilihon@ilihon:~/Ilihon/5sem/AVS/lab2$ ./main
Start tests...
All tests done
Saving results...
Results saved
ilihon@ilihon:~/Ilihon/5sem/AVS/lab2$
```

Просмотр scv файла

bench result.csv U 1 course\Computing systems\2 lab\bench result.csv

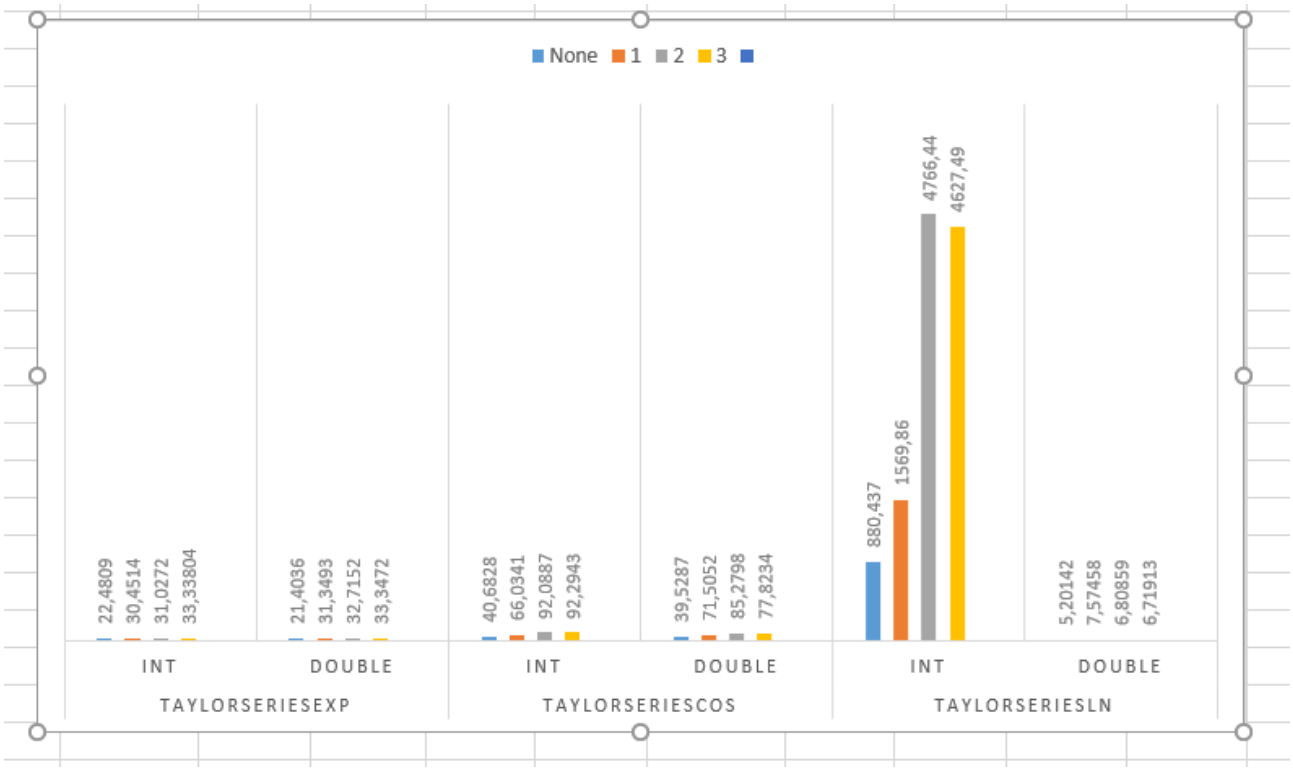
```
PModel,Task,OpType,Opt,LNum,InsCount,Timer,AvTime,AbsErr,RelErr,TaskPerf
12th Gen Intel(R) Core(TM),TaylorSeriesExp,int,None,10,100000,Clock,0.044771,0.0002888,0.64506,22.4809
12th Gen Intel(R) Core(TM),TaylorSeriesCos,int,None,10,100000,Clock,0.024762,0.0001816,0.733382,40.6828
12th Gen Intel(R) Core(TM),TaylorSeriesLn,int,None,10,100000,Clock,0.001198,6.22e-05,5.19199,880.437
12th Gen Intel(R) Core(TM),TaylorSeriesExp,int,1,10,100000,Clock,0.034272,0.0014328,4.18067,30.4514
12th Gen Intel(R) Core(TM),TaylorSeriesCos,int,1,10,100000,Clock,0.015191,4.73e-05,0.311369,66.0341
12th Gen Intel(R) Core(TM),TaylorSeriesLn,int,1,10,100000,Clock,0.00062,1.7e-05,2.74194,1569.86
12th Gen Intel(R) Core(TM),TaylorSeriesExp,int,2,10,100000,Clock,0.033138,0.0009082,2.74066,31.0272
12th Gen Intel(R) Core(TM),TaylorSeriesCos,int,2,10,100000,Clock,0.010959,9.99e-05,0.91158,92.0887
12th Gen Intel(R) Core(TM),TaylorSeriesLn,int,2,10,100000,Clock,0.000219,9.2e-06,4.20091,4766.44
12th Gen Intel(R) Core(TM),TaylorSeriesExp,int,3,10,100000,Clock,0.029632,0.0003257,1.09915,33.3804
12th Gen Intel(R) Core(TM),TaylorSeriesCos,int,3,10,100000,Clock,0.01146,0.0006251,5.45462,92.2943
12th Gen Intel(R) Core(TM),TaylorSeriesLn,int,3,10,100000,Clock,0.000222,5.9e-06,2.65766,4627.49
12th Gen Intel(R) Core(TM),TaylorSeriesExp,double,None,10,100000,Clock,0.045454,0.0012672,2.78787,21.4036
12th Gen Intel(R) Core(TM),TaylorSeriesCos,double,None,10,100000,Clock,0.024905,0.0003931,1.5784,39.5287
12th Gen Intel(R) Core(TM),TaylorSeriesLn,double,None,10,100000,Clock,0.186284,0.0059712,3.20543,5.20142
12th Gen Intel(R) Core(TM),TaylorSeriesExp,double,1,10,100000,Clock,0.031557,0.0003416,1.08249,31.3493
12th Gen Intel(R) Core(TM),TaylorSeriesCos,double,1,10,100000,Clock,0.013803,0.000182,1.31855,71.5052
12th Gen Intel(R) Core(TM),TaylorSeriesLn,double,1,10,100000,Clock,0.139002,0.0069815,5.02259,7.57458
12th Gen Intel(R) Core(TM),TaylorSeriesExp,double,2,10,100000,Clock,0.03086,0.0002932,0.950097,32.7152
12th Gen Intel(R) Core(TM),TaylorSeriesCos,double,2,10,100000,Clock,0.012321,0.0005949,4.82834,85.2798
12th Gen Intel(R) Core(TM),TaylorSeriesLn,double,2,10,100000,Clock,0.135584,0.0112892,8.32635,6.80859
12th Gen Intel(R) Core(TM),TaylorSeriesExp,double,3,10,100000,Clock,0.030771,0.0007835,2.54623,33.3472
12th Gen Intel(R) Core(TM),TaylorSeriesCos,double,3,10,100000,Clock,0.01758,0.0047304,26.9078,77.8234
12th Gen Intel(R) Core(TM),TaylorSeriesLn,double,3,10,100000,Clock,0.147112,0.0017167,1.16693,6.71913
```

Импорт scv файла в таблицу

	TaylorSeriesExp							TaylorSeriesCos							TaylorSeriesLn						
		int			double			int			double				int			double			
	Opt	AbsErr	RelErr	TaskPerf	AbsErr	RelErr	TaskPerf	AbsErr	RelErr	TaskPerf	AbsErr	RelErr	TaskPerf	AbsErr	RelErr	TaskPerf	AbsErr	RelErr	TaskPerf	
	None	0.0002888	0.64506	22.4809	0.0012672	2.78787	21.4036	0.0001816	0.733382	40.6828	0.0003931	01.015784	39.5287	6.22e-05	5.19199	880.437	0.0059712	3.20543	5.20142	
	1	0.0014328	4.18067	30.4514	0.0003416	1.08249	31.3493	4.73e-05	0.311369	66.0341	0.000182	1.31855	71.5052	1.7e-05	2.74194	1569.86	0.0069815	5.02259	7.57458	
	2	0.0009082	2.74066	31.0272	0.0002932	0.950097	32.7152	9.99e-05	0.91158	92.0887	0.0005949	4.82834	85.2798	9.2e-06	4.20091	4766.44	0.0112892	8.32635	6.80859	
	3	0.0003257	1.09915	33.3804	0.0007835	2.54623	33.3472	0.0006251	5.45462	92.2943	0.0047304	26.9078	77.8234	5.9e-06	2.65766	4627.49	0.0017167	1.16693	6.71913	

Pmodel	Task	OpType	Opt	LNum	InsCount	Timer	AvTime	AbsErr	RelErr	TaskPerf	
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	int	None	10	100000	Clock	0,044771	0.0002888	0.64506	22.4809	22,34
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	int	None	10	100000	Clock	0,024762	0.0001816	0.733382	40.6828	40,38
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	int	None	10	100000	Clock	0,001198	6.22e-05	5.19199	880.437	834,72
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	int	1	10	100000	Clock	0,034272	0.0014328	4.18067	30.4514	29,18
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	int	1	10	100000	Clock	0,015191	4.73e-05	0.311369	66.0341	65,83
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	int	1	10	100000	Clock	0,00062	1.7e-05	2.74194	1569.86	1612,90
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	int	2	10	100000	Clock	0,033138	0.0009082	2.74066	31.0272	30,18
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	int	2	10	100000	Clock	0,010959	9.99e-05	0.91158	92.0887	91,25
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	int	2	10	100000	Clock	0,000219	9.2e-06	4.20091	4766.44	4566,21
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	int	3	10	100000	Clock	0,029632	0.0003257	1.09915	33.3804	33,75
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	int	3	10	100000	Clock	0,01146	0.0006251	5.45462	92.2943	87,26
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	int	3	10	100000	Clock	0,000222	5.9e-06	2.65766	4627.49	4504,50
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	double	None	10	100000	Clock	0,045454	0.0012672	2.78787	21.4036	22,00
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	double	None	10	100000	Clock	0,024905	0.0003931	01.015784	39.5287	40,15
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	double	None	10	100000	Clock	0,186284	0.0059712	3.20543	5.20142	5,37
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	double	1	10	100000	Clock	0,031557	0.0003416	1.08249	31.3493	31,69
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	double	1	10	100000	Clock	0,013803	0.000182	1.31855	71.5052	72,45
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	double	1	10	100000	Clock	0,139002	0.0069815	5.02259	7.57458	7,19
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	double	2	10	100000	Clock	0,03086	0.0002932	0.950097	32.7152	32,40
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	double	2	10	100000	Clock	0,012321	0.0005949	4.82834	85.2798	81,16
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	double	2	10	100000	Clock	0,135584	0.0112892	8.32635	6.80859	7,38
12th Gen Intel(R) Core(TM)	TaylorSeriesExp	double	3	10	100000	Clock	0,030771	0.0007835	2.54623	33.3472	32,50
12th Gen Intel(R) Core(TM)	TaylorSeriesCos	double	3	10	100000	Clock	0,01758	0.0047304	26.9078	77.8234	56,88
12th Gen Intel(R) Core(TM)	TaylorSeriesLn	double	3	10	100000	Clock	0,147112	0.0017167	1.16693	6.71913	6,80

Сводная таблица с результатами бенчмарки для четырёх уровней оптимизации.



Гистограмма производительности.

Можно отметить следующие интересные особенности:

- Во всех случаях типовых задач, использование ключей оптимизации увеличивает производительность в 1,5 и более раз.
- В TaylorSeriesLn тестах заметная разница в производительности с входными данными типа int и типа double . Результаты прогонов с разными флагами оптимизации сильно отличаются в случае int и слабо в случае double.
- Во всех остальных случаях разница в производительности между разными ключами оптимизации не значительна.

Приложение

```
#include <iostream>
#include <time.h>
#include <fstream>
#include <math.h>

#define eps 0.00000000000001
#define N 100000
#define M 10

using namespace std;

double exp_(const int x)
{
	double s = 1;
	double n = 1;
	double a = 1;
	while (1)
	{
		a = a * x / n;
		if (fabs(a) <= eps) break;
		s = s + a;
		n++;
	}
	return s;
}

double cos_(const int x)
{
	double s = 0;
	double n = 1;
	double a = 1;
	while (1)
	{
		s = s + a;
		a = a * (-1)*x*x/((2 * n - 1) * (2 * n));
		if (fabs(a) <= eps) break;
		n++;
	}
	return s;
}

double ln_(const int x)
{
	double s = 0;
	double n = 1;
	double a = (x-1)/x;
	while (1)
	{
		s = s + a;
		a = a * n * (x - 1)/((n + 1)*x);
		if (fabs(a) <= eps) break;
		n++;
	}
	return s;
}

void Clock_exp(const int x, double &absolute, double &relative, double &result);
void Clock_cos(const int x, double &absolute, double &relative, double &result);
void Clock_ln(const int x, double &absolute, double &relative, double &result);

int main()
{
	const int x =5;
	cout.precision(15);

	double absolute_exp = 0;
	double relative_exp = 0;
	double result_exp = 0;
	double absolute_cos = 0;
	double relative_cos = 0;
	double result_cos = 0;
```

```
        double absolute_ln = 0;
        double relative_ln = 0;
        double result_ln = 0;
        cout << "Start tests..." << endl;
        Clock_exp(x, absolute_exp, relative_exp, result_exp);
        Clock_cos(x, absolute_cos, relative_cos, result_cos);
        Clock_ln(x, absolute_ln, relative_ln, result_ln);
        cout << "All tests done" << endl;

        FILE *process;
    char CPU_name[1024];

    process = popen(" lscpu | grep 'Имя модели' | awk '{$1=$2=NULL; print $0}' | tr -d '\n' | sed -e
's/^[[:space:]]*//' ", "r");

    if (process != NULL) {
        while (!feof(process)) {
            fgets(CPU_name, sizeof(CPU_name), process);
        }

        pclose(process);
    }
        cout << "Saving results..." << endl;

    ofstream benchmark_output;
        benchmark_output.open("bench result.csv", ios_base::app);
        benchmark_output                                                                    <<
"PModel;Task;OpType;Opt;LNum;InsCount;Timer;AvTime;AbsErr;RelErr;TaskPerf" << endl;

        benchmark_output << CPU_name << ";TaylorSeriesExp;int;None;" << M << ";" << N << ";Clock;"
<<
        result_exp << ";" << absolute_exp << ";" << relative_exp << ";" << 1/ result_exp << endl;
        benchmark_output << CPU_name << ";TaylorSeriesCos;int;None;" << M << ";" << N << ";Clock;"
<<
        result_cos << ";" << absolute_cos << ";" << relative_cos << ";" << 1/ result_cos << endl;
        benchmark_output << CPU_name << ";TaylorSeriesLn;int;None;" << M << ";" << N << ";Clock;" <<
        result_ln << ";" << absolute_ln << ";" << relative_ln << ";" << 1/ result_ln << endl;

        benchmark_output.close();

        cout << "Results saved" << endl;

        return 0;
}

void Clock_exp(const int x, double &absolute, double &relative, double &result) {
        unsigned int start, stop;
        for (int i = 0; i < M; i++) {
                start = (double)clock();
                for (int j = 0; j < N; j++)
                        exp_(x);
                stop = (double)clock();
                result += stop - start;
        }
        absolute = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC);
        relative = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC) / ((double)(stop -
start) / CLOCKS_PER_SEC) * 100;
        result = result / M / CLOCKS_PER_SEC;
}

void Clock_cos(const int x, double &absolute, double &relative, double &result) {
        unsigned int start, stop;
        for (int i = 0; i < M; i++) {
                start = (double)clock();
                for (int j = 0; j < N; j++)
                        cos_(x);
                stop = (double)clock();
                result += stop - start;
        }
        absolute = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC);
        relative = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC) / ((double)(stop -
start) / CLOCKS_PER_SEC) * 100;
        result = result / M / CLOCKS_PER_SEC;
```

```
}

void Clock_ln(const int x, double &absolute, double &relative, double &result) {
	unsigned int start, stop;
	for (int i = 0; i < M; i++) {
		start = (double)clock();
		for (int j = 0; j < N; j++)
			ln_(x);
		stop = (double)clock();
		result += stop - start;
	}
	absolute = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC);
	relative = fabs((result / M - (double)(stop - start)) / CLOCKS_PER_SEC) / ((double)(stop -
start) / CLOCKS_PER_SEC) * 100;
	result = result / M / CLOCKS_PER_SEC;
}
```