

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Качество и метрология программного обеспечения»**

**Тема: «Измерение характеристик динамической сложности программ с  
помощью профилировщика SAMPLER\_v2»**

Студент гр. 8304

Рыжиков А.В.

Преподаватель

Кирияничков В. А.

Санкт-Петербург

2022

## **Цель работы.**

Изучить возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER.

## **Задание.**

- 1) Выполнить под управлением SAMPLER тестовые программы test\_cyc.c и test\_sub.c и привести отчет по результатам их выполнения с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.
- 2) Разработанную в лаб. работе 1 программу, реализующую заданный вычислительный алгоритм, разбить на функциональные участки (ФУ) и расставить на их границах контрольные точки (КТ) для выполнения с помощью ПИМ SAMPLER измерений и получения профиля выполнения программы, представляющего времени выполнения и количество выполнений каждого ФУ.
- 3) Скомпилировать полученную программу. При компиляции добавить путь к sampler.h в набор путей поиска включаемых файлов (Isampler/lib sampler при компиляции, если архив был распакован в текущий каталог), при линковке добавить путь к libsampler.a в набор путей поиска библиотек и подключить её (флаги -LSampler/build/lib sampler -lsampler при линковке).
- 4) Выполнить скомпилированную программу под управлением Sampler'a с внешним заикливанием и получить отчет по результатам профилирования. Заикливание можно выполнять при помощи программы sampler-repeat. Использование программы приведено в разделе 4 документа «Описание работы с ПИМ SAMPLER\_v2». Число повторов зависит от сложности самой программы; имеет смысл начальное число запусков взять равным 10 и увеличивать его в 5–10 раз до тех пор, пока среднее время выполнения участков не стабилизируется, или на запуски станет уходить слишком много

времени, или на результаты станет уходить слишком много дискового пространства.

- 5) Проанализировать полученный отчет и выявить "узкие места", приводящие к ухудшению производительности программы.
- 6) Ввести в программу усовершенствования для повышения производительности, получить новые профили, добавить их в отчет и объяснить полученные результаты.

### **Ход работы.**

1. Под управлением SAMPLER была выполнена тестовая программа test\_cyc.c, программа sampler-repeat вызывалась с параметрами 1000 100.

<b>исх</b>	<b>прием</b>	<b>общее время</b>	<b>кол-во проходов</b>	<b>среднее время</b>
13	15	3713.947	1	3713.947
15	17	7408.111	1	7408.111
17	19	18293.988	1	18293.988
19	21	36594.598	1	36594.598
21	24	3657.444	1	3657.444
24	27	7405.383	1	7405.383
27	30	18032.603	1	18032.603
30	33	36127.186	1	36127.186
33	39	3632.168	1	3632.168
39	45	7430.824	1	7430.824
45	51	17904.643	1	17904.643
51	57	35309.666	1	35309.666

Таблица 1 — Результаты профилирования программы test\_cyc.c

Программа test\_cyc содержит несколько циклов, выполняющих перестановку элементов одного статического массива. Первый цикл обрабатывает первую 1/10 элементов массива, следующий 2/10, третий 5/10, а четвёртый обрабатывает массив целиком. Затем такая же последовательность повторяется ещё дважды с различным представлением тела цикла. Несмотря на то, что циклы выполняют одни и те же действия, их производительность различается из-за того, что первые 4 цикла обеспечивают попадание массива в кэш процессора и поэтому работают медленнее чем оставшиеся 8 циклов. Видно, что среднеквадратичное отклонение велико по сравнению

со средним значением времени из-за того, что исполнение цикла происходит достаточно быстро и это время сравнимо с погрешностью.

2. Под управлением SAMPLER была выполнена тестовая программа test\_sub.c, программа sampler-repeat вызывалась с параметрами 100 10.

исх	прием	общее время	кол-во проходов	среднее время
30	32	21274948.467	1	21274948.467
32	34	42879579.844	1	42879579.844
34	36	111845085.706	1	111845085.706
36	38	238436592.022	1	238436592.022

Таблица 2 — Результаты профилирования программы test\_sub.c

Программа test\_sub содержит цикл, вынесенный в функцию. По результатам видно что время исполнения цикла в функции зависит от переданного аргумента nTimes — при увеличении nTimes в 2 раза примерно в 2 раза увеличивается время исполнения. Видно, что среднеквадратичное отклонение невелико по сравнению со средним значением времени из-за того, что исполнение цикла происходит достаточно долго и это время гораздо больше погрешности.

3. Разработанная в лаб. работе 1 программа была разбита на функциональные участки (ФУ) и были расставлены на их границах контрольные точки (КТ) для выполнения с помощью ПИМ SAMPLER измерений и получения профиля выполнения программы, представляющего времени выполнения и количество выполнений каждого ФУ. Предварительно программа из первой лабораторной была несколько изменена с связи с особенностями работы компилятора gcc, а также увеличен размер обрабатываемого массива с 5 до 1000.

4. Под управлением SAMPLER была выполнена программа из лаб. работы 1 (см. исходный код в приложении А) — см. табл 3, программа sampler-repeat вызывалась с параметрами 1000 100.

исх	прием	общее время	кол-во проходов	среднее время
38	42	2735.913	1	2735.913
42	46	283.036	1	283.036
46	46	21258.752	999	21.280
46	48	32.768	1	32.768
48	12	24.645	1	24.645
12	21	34.604	1	34.604
21	21	1	999	1
21	29	37.738	1	37.738
29	32	45.078	1	45.078
32	32	6512.305	999	6.519
32	68	55.327	1	55.327
68	72	462.682	1	462.682

Таблица 3 — Результаты профилирования программы из лаб. работы 1

Большую часть времени работы программы занимает заполнение массива случайными значениями (42 – 46, 46 – 46) и обработка данных (строки 12 – 21, 21 – 21). Оптимизирован был фрагмент кода, отвечающий за заполнения массива значениями (42 – 46, 46 – 46) и обработку данных (12 – 22), за место копирования значений элементов массива и использования новых переменных происходит обращение к массиву напрямую, также была удалена строка номер 26 (21 – 29), так переменная `suu`, нигде более в расчётах не используется (данный недочёт был в исходном коде на языке Паскаль).

5. Под управлением SAMPLER была выполнена оптимизированная программа из лаб. работы 1 (см. исходный код в приложении В) — см. табл 4, программа `sampler-repeat` вызывалась с параметрами 1000 100.

исх	прием	общее время	кол-во проходов	среднее время
38	42	2671.955	1	2671.955
42	46	32.005	1	32.005
46	46	4815.023	999	4.820
46	48	24.447	1	24.447
48	12	24.648	1	24.648
12	21	32.229	1	32.229
21	21	9166.982	999	9.176
21	29	32.759	1	32.759
29	32	28.632	1	28.632
32	32	6969.899	999	6.977
32	68	51.993	1	51.993
68	72	528.034	1	528.034

Таблица 4 — Результаты профилирования оптимизированной программы из лаб. работы 1

Среднее время оптимизированных фрагментов – заполнение массива (строчки 42 – 46, 46 – 46) уменьшилось с 21258.752 мкс до 4815.023 мкс и обработка массива (строчки 12 – 21, 21 – 21) с 10153.590 мкс до 9166.982 мкс. Стоит отметить, что оптимизация фрагментов кода, влияющих на скорость выполнения программы соответствующей выданному варианту крайне затруднительно, так как «пространство» для оптимизации крайне мало, так как нет фрагментов вызывающих сторонних математических функции обработки данных.

### **Выводы.**

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER. Для программы, взятой из первой лабораторной работы, было выполнено измерение времени работы, с последующим выявлением узких мест и их оптимизация.

## ПРИЛОЖЕНИЕ А. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ДО ОПТИМИЗАЦИИ

```
#include <stdio.h>
#include <stdlib.h>
#include "sampler.h"

#ifdef n
#define n 1000
#endif

void linfit2(float *x, float *y, float *y_calc) {
    float sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0, sum_y2 = 0;
    float xi, yi, sxy, sxx, syy;
    SAMPLE;
    for (int i = 0; i < n; ++i) {
        xi = x[i];
        yi = y[i];
        sum_x += xi;
        sum_y += yi;
        sum_xy += xi * yi;
        sum_x2 += xi * xi;
        sum_y2 += yi * yi;
        SAMPLE;
    }

    sxx = sum_x2 - sum_x * sum_x / n;
    sxy = sum_xy - sum_x * sum_y / n;
    syy = sum_y2 - sum_y * sum_y / n;
    float b = sxy / sxx;
    float a = ((sum_x2 * sum_y - sum_x * sum_xy) / n) / sxx;
    SAMPLE;
    for (int i = 0; i < n; ++i) {
        y_calc[i] = a + b * x[i];
        SAMPLE;
    }
}

int main(int argc, char **argv) {
    sampler_init(&argc, argv);
    SAMPLE;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    float *y_calc = (float*) malloc(n * sizeof(float));
    SAMPLE;
    for (int i = 0; i < n; i++) {
        x[i] = (float) (rand() % 100) / 100;
        y[i] = (float) (rand() % 100) / 100;
        SAMPLE;
    }
    SAMPLE;
    linfit2(x, y, y_calc);
    //SAMPLE;
    //printf("x: ");
    //for (int i = 0; i < n; ++i) {
```

```

// printf("%f ;", x[i]);
// SAMPLE;
//}
//SAMPLE;
//printf("\ny: ");
//for (int i = 0; i < n; ++i) {
// printf("%f ;", y[i]);
// SAMPLE;
//}
//SAMPLE;
//printf("\ny_calc: ");
//for (int i = 0; i < n; ++i) {
// printf("%f ;", y_calc[i]);
// SAMPLE;
//}
SAMPLE;
free(x);
free(y);
free(y_calc);
SAMPLE;
}

```



## ПРИЛОЖЕНИЕ В. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ПОСЛЕ ОПТИМИЗАЦИИ

```
#include <stdio.h>
#include <stdlib.h>
#include "sampler.h"

#ifndef n
#define n 1000
#endif

void linfit2(float *x, float *y, float *y_calc) {
    float sum_x = 0, sum_y = 0, sum_xy = 0, sum_x2 = 0, sum_y2 = 0;
    float sxy, sxx, syy;
    SAMPLE;
    for (int i = 0; i < n; ++i) {

        sum_x += x[i];
        sum_y += y[i];
        sum_xy += x[i] * y[i];
        sum_x2 += x[i] * x[i];
        sum_y2 += y[i] * y[i];
        SAMPLE;
    }

    sxx = sum_x2 - sum_x * sum_x / n;
    sxy = sum_xy - sum_x * sum_y / n;

    float b = sxy / sxx;
    float a = ((sum_x2 * sum_y - sum_x * sum_xy) / n) / sxx;
    SAMPLE;
    for (int i = 0; i < n; ++i) {
        *(y_calc + i) = a + b * x[i];
        SAMPLE;
    }
}

int main(int argc, char **argv) {
    sampler_init(&argc, argv);
    SAMPLE;
    float *x = (float*) malloc(n * sizeof(float));
    float *y = (float*) malloc(n * sizeof(float));
    float *y_calc = (float*) malloc(n * sizeof(float));
    SAMPLE;
    for (int i = 0; i < n; i++) {
        x[i] = 0.5;
        y[i] = 0.5;
        SAMPLE;
    }
    SAMPLE;
    linfit2(x, y, y_calc);
    //SAMPLE;
    //printf("x: ");
    //for (int i = 0; i < n; ++i) {
```

```

// printf("%f ;", x[i]);
// SAMPLE;
//}
//SAMPLE;
//printf("\ny: ");
//for (int i = 0; i < n; ++i) {
// printf("%f ;", y[i]);
// SAMPLE;
//}
//SAMPLE;
//printf("\ny_calc: ");
//for (int i = 0; i < n; ++i) {
// printf("%f ;", y_calc[i]);
// SAMPLE;
//}
SAMPLE;
free(x);
free(y);
free(y_calc);
SAMPLE;
}

```