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 _____ Кирьянчиков 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 времена выполнения и количество выполнений каждого ФУ.
3) Скомпилировать полученную программу. При компиляции добавить путь к sampler.h в набор путей поиска включаемых файлов (`Isampler/libsampler` 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/libsampler -lsampler` при линковке).
4) Выполнить скомпилированную программу под управлением Sampler’a с внешним зацикливанием и получить отчет по результатам профилирования. Зацикливание 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.

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**
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 времена 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 10 000.

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
36	40	4203.227	1	4203.227
40	45	217060.141	1	217060.141
45	12	33.535	1	33.535
12	22	76859.288	1	76859.288
22	28	21.234	1	21.234
28	47	45550.586	1	45550.586
47	52	7629877.712	1	7629877.712
52	57	7201618.330	1	7201618.330
57	62	7038043.473	1	7038043.473

Таблица 3 — Результаты профилирования программы из лаб. работы 1

Большую часть времени работы программы занимает заполнение массива рандомными значениями (40 – 45) и вывод данных (строчки 47 – 52, 52 – 57, 57 – 62). Оптимизирован был фрагмент кода, отвечающий за обработку днных 12 – 22, за место использования новых переменных, копирующих значение элементов массива, происходит обращение к массиву напрямую, также была удалена строка номер 25 (22 – 28), так переменная syy, 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
36	40	4140.793	1	4140.793
40	45	213350.583	1	213350.583
45	12	22.621	1	22.621
12	22	63146.167	1	63146.167
22	28	21.130	1	21.130
28	47	44259.915	1	44259.915
47	52	7412339.886	1	7412339.886
52	57	7375522.713	1	7375522.713
57	62	6690019.526	1	6690019.526

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

Среднеее время оптимизированного фрагмента уменьшилось с 76 859 мкс до 63146 мкс, что несущественно по сравнению с фрагментами кода, которые выполняются основную часть врменени выполнения программы. 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, а оптимизация методов rand, malloc и printf из стандартной библиотеки крайне сомнительна.

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 – в результате существенно уменьшить время исполнения программы не удалось ввиду отсутствия того, что основное время выполнения отводится к функциям вывода и выделения/освобождления памяти стандартной библиотеки, а обработка данных не содержит затратных вычислительных операций.

ПРИЛОЖЕНИЕ А. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ДО ОПТИМИЗАЦИИ

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

#ifndef n
#define n 10000
#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];
   }
}

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;
   linfit2(x, y, y_calc);
   SAMPLE;
   printf("x: ");
   for (int i = 0; i < n; ++i) {
      printf("%f ;", x[i]);
   }
   SAMPLE;
```

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

ПРИЛОЖЕНИЕ В. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ПОСЛЕ ОПТИМИЗАЦИИ

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

#ifndef n
#define n 10000
#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 += y[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];
   }
}

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;
   linfit2(x, y, y_calc);
   SAMPLE;
   printf("x: ");
   for (int i = 0; i < n; ++i) {
      printf("%f ;", x[i]);
   }
   SAMPLE;
   printf("\ny: ");
```

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