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

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 _____ Мешков М.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

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.
4. Выполнить скомпилированную программу под управлением Sampler’a с внешним зацикливанием и получить отчет по результатам профилирования.
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 выполнения.

1. Под управлением SAMPLER была выполнена тестовая программа test_cyc.c (см. исходный код в приложении А) — см. табл 1, программа sampler-repeat вызывалась с параметрами 1000 100.

Таблица 1 — Результаты профилирования программы test_cyc.c

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**	**std**
13	15	2120.523	1	2120.523	8969.393
15	17	4179.852	1	4179.852	7738.553

исх	прием	общее время	кол-во проходов	среднее время	std
17	19	19063.516	1	19063.516	12431.153
19	21	31853.645	1	31853.645	14184.452
21	24	1019.018	1	1019.018	7134.970
24	27	2330.402	1	2330.402	8274.105
27	30	13714.654	1	13714.654	10058.991
30	33	27219.321	1	27219.321	12584.246
33	39	1579.459	1	1579.459	7533.704
39	45	2579.047	1	2579.047	9279.160
45	51	15747.850	1	15747.850	11776.924
51	57	29007.087	1	29007.087	11771.923

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 (см. исходный код в приложении Б) — см. табл 2, программа sampler-repeat вызывалась с параметрами 10 1.

Таблица 2 — Результаты профилирования программы test_sub.c

исх	прием	общее время	кол-во проходов	среднее время	std
30	32	22901882.389	1	22901882.389	634629.070
32	34	44482707.667	1	44482707.667	589528.990
34	36	109285745.333	1	109285745.333	2687556.558
36	38	219456180.833	1	219456180.833	3038777.875

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. Исходный код программы с контрольными точками см. в приложении В.

4. Под управлением SAMPLER была выполнена программа из лаб. работы 1 (см. исходный код в приложении В) — см. табл 3, программа sampler-repeat вызывалась с параметрами 1000 5.

Таблица 3 — Результаты профилирования программы из лаб. работы 1

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**	**std**
41	20	182.466	1	182.466	7679.626
20	26	849.388	7	121.341	7328.258
26	30	1005.088	7	143.584	7895.997
30	30	14983.617	247	60.662	7475.214
30	34	423.797	7	60.542	7046.518
34	20	1140.601	6	190.100	8019.699
34	36	59.628	1	59.628	6543.735

Видно, что большую часть времени работы программы исполняется цикл на 27-31 строках. Цикл на 27-31 строках был оптимизирован (в нем было уменьшено количество делений и умножений) — см. исходный код в приложении Г (строки 27-30). Можно было полностью избавиться от умножений в этом цикле (на каждой итерации добавлять delta_x,

предварительно умноженный на 2), однако это бы привело к потере точности из-за все большего накопления ошибки на каждой итерации.

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

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

исх	**прием**	**общее время**	**кол-во проходов**	**среднее время**	**std**
40	20	84.774	1	84.774	9517.118
20	26	-367.210	7	-52.459	6886.560
26	29	1298.729	7	185.533	7598.311
29	29	9020.693	247	36.521	7382.230
29	33	74.334	7	10.619	7259.119
33	20	1619.697	6	269.949	7926.139
33	35	-81.145	1	-81.145	6459.122

Видно что время исполнения после оптимизации уменьшилось.

Выводы.

В результате выполнения данной лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER. Для программы из лаб. работы 1 было проведено измерение времени работы, найдено узкое место, проблемный участок кода был оптимизирован — удалось добиться ускорения выполнения программы.

ПРИЛОЖЕНИЕ А. ПРОГРАММА TEST_CYC

```
#include "sampler.h"

#ifndef SIZE
#define SIZE 10000
#endif

int main(int argc, char **argv)
{
    sampler_init(&argc, argv);

    int i, tmp, dim[SIZE];

    SAMPLE;
    for (i=0; i<SIZE/10; i++) { tmp=dim[0]; dim[0]=dim[i];
dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/5; i++) { tmp=dim[0]; dim[0]=dim[i];
dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/2; i++) { tmp=dim[0]; dim[0]=dim[i];
dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE; i++) { tmp=dim[0]; dim[0]=dim[i];
dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/10; i++)
        { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/5; i++)
        { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/2; i++)
        { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE; i++)
        { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; }
    SAMPLE;
    for (i=0; i<SIZE/10; i++)
        { tmp=dim[0];
```

```
            dim[0]=dim[i];
            dim[i]=tmp;
        }
    SAMPLE;
    for (i=0; i<SIZE/5; i++)
        { tmp=dim[0];
            dim[0]=dim[i];
            dim[i]=tmp;
        }
    SAMPLE;
    for (i=0; i<SIZE/2; i++)
        { tmp=dim[0];
            dim[0]=dim[i];
            dim[i]=tmp;
        }
    SAMPLE;
    for (i=0; i<SIZE; i++)
        { tmp=dim[0];
            dim[0]=dim[i];
            dim[i]=tmp;
        }
    SAMPLE;
}
```

ПРИЛОЖЕНИЕ Б. ПРОГРАММА TEST_SUB

```
#include "sampler.h"

#ifndef SIZE
#define SIZE 10000
#endif

void TestLoop(int nTimes)
{
   static int TestDim[SIZE];
   int tmp;
   int iLoop;

   while (nTimes > 0) {
      --nTimes;

      iLoop = SIZE;
      while (iLoop > 0) {
         --iLoop;
         tmp = TestDim[0];
         TestDim[0] = TestDim[nTimes];
         TestDim[nTimes] = tmp;
      }
   }
}

int main(int argc, char **argv)
{
   sampler_init(&argc, argv);
```

```
    SAMPLE;
    TestLoop(SIZE / 10);
    SAMPLE;
    TestLoop(SIZE / 5);
    SAMPLE;
    TestLoop(SIZE / 2);
    SAMPLE;
    TestLoop(SIZE / 1);
    SAMPLE;
}
```

ПРИЛОЖЕНИЕ В. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ДО ОПТИМИЗАЦИИ

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "sampler.h"
 4
 5  double fx(double x) {
 6      return 1.0 / x;
 7  }
 8
 9  void simps(double lower, double upper, double tol, double *sum) {
10      int pieces = 2;
11      double
12          delta_x = (upper-lower)/pieces,
13          odd_sum = fx(lower+delta_x),
14          even_sum = 0.0,
15          end_sum = fx(lower) + fx(upper);
16      *sum = (end_sum + 4.0 * odd_sum)*delta_x/3.0;
17      // printf("%5d %f\n", pieces, *sum);
18      double sum1;
19      do {
20          SAMPLE;
21          pieces *= 2;
22          sum1 = *sum;
23          delta_x = (upper-lower)/pieces;
24          even_sum = even_sum + odd_sum;
25          odd_sum = 0.0;
26          SAMPLE;
27          for (int i = 1; i <= pieces/2; i++) {
28              double x = lower+delta_x*(2*i-1);
```

```
29            odd_sum += fx(x);
30            SAMPLE;
31        }
32        *sum = (end_sum + 4.0*odd_sum + 2.0*even_sum)*delta_x/3.0;
33        // printf("%5d %f\n", pieces, *sum);
34        SAMPLE;
35    } while (*sum != sum1 && fabs(*sum-sum1) >= fabs(tol**sum));
36    SAMPLE;
37 }
38
39 int main(int argc, char **argv) {
40    sampler_init(&argc, argv);
41    SAMPLE;
42    const double tol = 1.0e-6;
43    double
44        lower = 1.0,
45        upper = 9.0,
46        sum;
47    simps(lower, upper, tol, &sum);
48    // printf("\narea=%f\n", sum);
49 }
```

ПРИЛОЖЕНИЕ Г. ПРОГРАММА ИЗ ЛАБ. РАБОТЫ 1 ПОСЛЕ ОПТИМИЗАЦИИ

```
 1  #include <stdio.h>
 2  #include <math.h>
 3  #include "sampler.h"
 4
 5  double fx(double x) {
 6      return 1.0 / x;
 7  }
 8
 9  void simps(double lower, double upper, double tol, double *sum) {
10      int pieces = 2;
11      double
12          delta_x = (upper-lower)/pieces,
13          odd_sum = fx(lower+delta_x),
14          even_sum = 0.0,
15          end_sum = fx(lower) + fx(upper);
16      *sum = (end_sum + 4.0 * odd_sum)*delta_x/3.0;
17      // printf("%5d %f\n", pieces, *sum);
18      double sum1;
19      do {
20          SAMPLE;
21          pieces *= 2;
22          sum1 = *sum;
23          delta_x = (upper-lower)/pieces;
24          even_sum = even_sum + odd_sum;
25          odd_sum = 0.0;
26          SAMPLE;
27          for (int i = 1; i < pieces; i+=2) {
28              odd_sum += fx(lower+delta_x*i);
```

```
29            SAMPLE;
30        }
31        *sum = (end_sum + 4.0*odd_sum + 2.0*even_sum)*delta_x/3.0;
32        // printf("%5d %f\n", pieces, *sum);
33        SAMPLE;
34    } while (*sum != sum1 && fabs(*sum-sum1) >= fabs(tol**sum));
35    SAMPLE;
36 }
37
38 int main(int argc, char **argv) {
39    sampler_init(&argc, argv);
40    SAMPLE;
41    const double tol = 1.0e-6;
42    double
43        lower = 1.0,
44        upper = 9.0,
45        sum;
46    simps(lower, upper, tol, &sum);
47    // printf("\narea=%f\n", sum);
48 }
```