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

Студентка гр. 8304 Сани З.Б.

Преподаватель Кирьянчиков В. А.

Санкт-Петербург
2022

Цель работы.

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

Ход выполнения.

Были выполнены под управлением монитора SAMPLER тестовые программы test_cyc.cpp и test_sub.cpp. Результаты представлены на рисунках 1 и 2.

исх	прием	общее время	кол-во проходов	среднее время
13	15	11050.568	1	11050.568
15	17	22977.411	1	22977.411
17	19	56520.118	1	56520.118
19	21	113167.289	1	113167.289
21	24	11970.644	1	11970.644
24	27	22760.383	1	22760.383
27	30	56508.136	1	56508.136
30	33	113069.113	1	113069.113
33	39	10710.835	1	10710.835
39	45	27217.595	1	27217.595
45	51	54850.554	1	54850.554
51	57	110380.831	1	110380.831

Рисунок 1 – Результат работы SAMPLER для программы test_cyc.cpp

исх	прием	общее время	кол-во проходов	среднее время
30	32	30014470.300	1	30014470.300
32	34	64816604.700	1	64816604.700
34	36	133547045.000	1	133547045.000
36	38	271665139.550	1	271665139.550

Рисунок 2 – результат работы SAMPLER для программы test_sub.cpp

Программа test_cyc содержит несколько циклов, выполняющих перестановку элементов массива. В каждый следующий цикл передаётся большее число элементов для перестановки, поэтому время обработки различается. Так же причиной является то, что первые 4 цикла обеспечивают попадание массива в кэш процессора и обрабатываются быстрее, чем остальные.

Программа test_sub содержит функцию, обрабатывающую те же циклы. С помощью результатов профилировщика можно убедиться, что время исполнения

цикла в функции не будет заметно отличаться от времени исполнения цикла вне её при том же объёме работы.

Была выполнена под управлением монитора SAMPLER программа вычисления функции ошибок распределения Гаусса (вар.1) из лабораторной работы №1. Результат измерений для полного времени выполнения функции представлен на рисунке 3. Исходный код этой программы представлен в Приложении А.

исх	прием	общее время	кол-во проходов	среднее время
52	60	7855.365	1	7855.365

Рисунок 3 – Результат работы SAMPLER для измерения полного времени выполнения функции

Было выполнено разбиение программы из лабораторной работы №1 на функциональные участки. Исходный код программы, разбитый на функциональные участки, представлен в приложении Б.

Полученные с помощью программы SAMPLER результаты представлены на рисунке 4.

исх	прием	общее время	кол-во проходов	среднее время
8	11	65.609	1	65.609
11	24	32.294	1	32.294
24	26	7.294	1	7.294
26	28	9.568	1	9.568
28	30	7693.787	1	7693.787

Рисунок 4 – Результат работы SAMPLER для измерения полного времени выполнения функции sort, разбитой на функциональные участки

В итоге общее время выполнения – 7808,552 мкс.

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным по среднему времени фрагментом являются вычисления в строках 28-30. Для оптимизации исключим переменную sum, а её вычисления в строке 27 подставим вместо переменной sum в строке 29. Также удалим переменную t3, значение которой совпадает с переменной t2. Переменная sum была удалена в строках 45, 47, а её выражение подставлено в строку 49

Была выполнена проверка изменённой программы. Результат представлен на рисунке 5. Исходный код модифицированной программы представлен в Приложении В.

исх	прием	общее время	кол-во проходов	среднее время
8	11	6.399	1	6.399
11	23	25.566	1	25.566
23	25	7.270	1	7.270
25	27	6690.805	1	6690.805

Рисунок 5 – Результат работы SAMPLER для измерения полного времени выполнения функции sort после внесения изменений

В результате внесённых изменений общее время выполнения – 6730,04 мкс.

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 узкого места и его устранения – в результате чего удалось добиться более быстрого выполнения функции ошибок распределения Гаусса.

ПРИЛОЖЕНИЯ

Приложение А. Исходный код программы erfd

```
#include <stdio.h>
#include <math.h>

double x,er,ec;

double erf(double x){
    const double sqrtpi = 1.7724538;
    const double t2          = 0.66666667;
    const double t3          = 0.66666667;
     const double     t4           = 0.07619048;
     const double     t5           = 0.01693122;
     const double     t6           = 3.078403E-3;
     const double     t7           = 4.736005E-4;
     const double     t8           = 6.314673E-5;
     const double     t9           = 7.429027E-6;
     const double     t10          = 7.820028E-7;
     const double     t11          = 7.447646E-8;
     const double     t12          = 6.476214E-9;

    double x2,sum, erf;

    x2 = x*x;
    sum = t5+x2*(t6+x2*(t7+x2*(t8+x2*(t9+x2*(t10+x2*(t11+x2*t12))))));
    erf = 2.0*exp(-x2)/sqrtpi*(x*(1+x2*(t2+x2*(t3+x2*(t4+x2*sum)))));
    return erf;
}

double erfc(double x){
    const double sqrtpi           = 1.7724538;

    double    x2,v,sum, erfc;
    x2 = x*x;
    v = 1.0/(2.0*x2);
    sum = v/(1+8*v/(1+9*v/(1+10*v/(1+11*v/(1+12*v)))));
    sum = v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*sum)))));
    erfc = 1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*sum)));
    return erfc;
}

int main()
{
    x = 1.00;

    if (x == 0.0){
      er = 0.0;
      ec = 1.0;
    }
    else{
      if (x < 1.5){
        er = erf(x);
        ec = 1.0 - er;
      } else {
        ec = erfc(x);
        er = 1.0 - ec;
      }
    }
```

```
57.
58.     return 0;
59. }
```

Приложение Б. Исходный код программы erfd с разделением на ФУ

```
1. #include <stdio.h>
2. #include <math.h>
3. #include "sampler.h"
4.
5. double x,er,ec;
6.
7. double erf(double x){
8.     SAMPLE;
9.     double x2,sum, erf;
10.
11.     SAMPLE;
12.     const double sqrtpi = 1.7724538;
13.     const double t2         = 0.66666667;
14.     const double t3         = 0.66666667;
15.     const double     t4           = 0.07619048;
16.     const double     t5           = 0.01693122;
17.     const double     t6           = 3.078403E-3;
18.     const double     t7           = 4.736005E-4;
19.     const double     t8           = 6.314673E-5;
20.     const double     t9           = 7.429027E-6;
21.     const double     t10          = 7.820028E-7;
22.     const double     t11          = 7.447646E-8;
23.     const double     t12          = 6.476214E-9;
24.     SAMPLE;
25.     x2 = x*x;
26.     SAMPLE;
27.     sum = t5+x2*(t6+x2*(t7+x2*(t8+x2*(t9+x2*(t10+x2*(t11+x2*t12))))));
28.     SAMPLE;
29.     erf = 2.0*exp(-x2)/sqrtpi*(x*(1+x2*(t2+x2*(t3+x2*(t4+x2*sum)))));
30.     SAMPLE;
31.     return erf;
32. }
33.
34. double erfc(double x){
35.     SAMPLE;
36.     double    x2,v,sum, erfc;
37.     SAMPLE;
38.     const double sqrtpi           = 1.7724538;
39.
40.     SAMPLE;
41.     x2 = x*x;
42.     SAMPLE;
43.     v = 1.0/(2.0*x2);
44.     SAMPLE;
45.     sum = v/(1+8*v/(1+9*v/(1+10*v/(1+11*v/(1+12*v)))));
46.     SAMPLE;
47.     sum = v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*sum)))));
48.     SAMPLE;
49.     erfc = 1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*sum)));
50.     SAMPLE;
51.     return erfc;
52. }
53.
54. int main(int argc, char **argv)
55. {
56.     sampler_init(&argc, argv);
57.
58.     x = 1.00;
```

```
59.
60.     if (x == 0.0){
61.       er = 0.0;
62.       ec = 1.0;
63.     }
64.     else{
65.       if (x < 1.5){
66.         er = erf(x);
67.         ec = 1.0 - er;
68.       } else {
69.         ec = erfc(x);
70.         er = 1.0 - ec;
71.       }
72.     }
73.
74.     return 0;
75. }
```

Приложение B. Исходный код программы erfd с оптимизациями

```
1. #include <stdio.h>
2. #include <math.h>
3. #include "sampler.h"
4.
5. double x,er,ec;
6.
7. double erf(double x){
8.     SAMPLE;
9.     double x2, erf;
10.
11.     SAMPLE;
12.     const double sqrtpi = 1.7724538;
13.     const double t2           = 0.66666667;
14.     const double       t4           = 0.07619048;
15.     const double       t5           = 0.01693122;
16.     const double       t6           = 3.078403E-3;
17.     const double       t7           = 4.736005E-4;
18.     const double       t8           = 6.314673E-5;
19.     const double       t9           = 7.429027E-6;
20.     const double       t10          = 7.820028E-7;
21.     const double       t11          = 7.447646E-8;
22.     const double       t12          = 6.476214E-9;
23.     SAMPLE;
24.     x2 = x*x;
25.     SAMPLE;
26.     erf = 2.0*exp(-
x2)/sqrtpi*(x*(1+x2*(t2+x2*(t2+x2*(t4+x2*(t5+x2*(t6+x2*(t7+x2*(t8+x2*(t9+x2*(t10+x2*(t11+x2*
t12))))))))))));
27.     SAMPLE;
28.     return erf;
29. }
30.
31. double erfc(double x){
32.     SAMPLE;
33.     double      x2,v, erfc;
34.     SAMPLE;
35.     const double sqrtpi             = 1.7724538;
36.
37.     SAMPLE;
38.     x2 = x*x;
39.     SAMPLE;
40.     v = 1.0/(2.0*x2);
41.     SAMPLE;
42.     erfc =
1.0/(exp(x2)*x*sqrtpi*(1+v/(1+2*v/(1+3*v/(1+4*v/(1+5*v/(1+6*v/(1+7*v/(1+8*v/(1+9*v/(1+10*v/(
1+11*v/(1+12*v))))))))))));
43.     SAMPLE;
44.     return erfc;
45. }
46.
47. int main(int argc, char **argv)
48. {
49.     sampler_init(&argc, argv);
50.
51.     x = 1.00;
52.
53.     if (x == 0.0){
54.       er = 0.0;
55.       ec = 1.0;
56.     }
57.     else{
58.       if (x < 1.5){
59.          er = erf(x);
60.          ec = 1.0 - er;
61.       } else {
62.          ec = erfc(x);
63.          er = 1.0 - ec;
64.       }
```

```
65.     }
66.
67.     return 0;
68. }
```