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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Измерение характеристик динамической сложности программ с помощью профилировщика SAMPLER_v2»

Студент гр. 8304 _____ Щука А. А.

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'а 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.

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

исх	прием	общее время	кол-во проходов	среднее время
13	15	2724.500	1	2724.500
15	17	5509.600	1	5509.600
17	19	21113.700	1	21113.700
19	21	29048.100	1	29048.100
21	24	2421.700	1	2421.700
24	27	6710.600	1	6710.600
27	30	21085.800	1	21085.800
30	33	27679.600	1	27679.600
33	39	2645.300	1	2645.300
39	45	6723.400	1	6723.400
45	51	16509.200	1	16509.200
51	57	35612.900	1	35612.900

Рисунок 1 – Результат работы SAMPLER для программы test_cyc.cpp

исх	прием	общее время	кол-во проходов	среднее время
30	32	38266799.700	1	38266799.700
32	34	78433060.500	1	78433060.500
34	36	202983276.400	1	202983276.400
36	38	352584030.100	1	352584030.100

Рисунок 2 - Результат работы SAMPLER для программы test_sub.cpp

Была выполнена под управлением монитора SAMPLER программа из лабораторной работы №1. Результат измерений для полного времени выполнения функции Sort представлен на рисунке 3. Исходный код этой программы представлен в Приложении А.

исх	прием	общее время	кол-во проходов	среднее время
47	49	26862.534	1	26862.534

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
15	17	5.789	1	5.789
17	20	44.822	1	44.822
20	23	31.378	7	4.483
23	30	6522.833	581	11.227
23	35	6353.611	622	10.215
30	32	9416.700	1135	8.297
32	35	5354.944	581	9.217
32	30	4467.956	554	8.065
35	23	3474.044	1196	2.905
35	37	53.567	7	7.652
37	20	37.467	6	6.244
37	39	6.867	1	6.867

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

В итоге общее время выполнения – 35769,98 мкс. Разницу в 8907,44 мкс с измерением для полного времени можно объяснить случайной генерацией массива из 200 элементов для сортировки и, следовательно, разным ветвлением в функции сортировки.

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратным фрагментом является тело цикла for (строчки 23-35). Для решения этой проблемы в 26 и 31 строчках операции взятия элемента массива по индексу были заменены адресной арифметикой. Также для уменьшения времени прохождения кода была создана переменная tmp, чтобы вместо четырёх операций сложения выполнялась только одна.

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
15	18	6.800	1	6.800
18	21	28.100	1	28.100
21	24	27.000	7	3.857
24	32	7241.800	581	12.464
24	37	6169.400	622	9.919
32	34	6361.200	1135	5.605
34	37	7338.000	581	12.630
34	32	4750.500	554	8.575
37	24	2434.200	1196	2.035
37	39	70.200	7	10.029
39	21	25.500	6	4.250
39	41	8.600	1	8.600

В результате внесённых изменений общее время выполнения – 34461,3 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 программы.

Также при проведении замеров было обнаружено, что независимо от количества циклов, из-за случайной генерации данных измеренные результаты постоянно изменяются.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "sampler.h"
4
5 void swap(float* x, float* y)
6 {
7    float temp;
8    temp = *x;
9    *x = *y;
10    *y = temp;
11 }
12
13 void shellsort(float arr[], int num)
14 {
15    int i, j, k;
16    for (i = num / 2; i > 0; i = i / 2)
17    {
18        for (j = i; j < num; j++)
19        {
20            for (k = j - i; k >= 0; k = k - i)
21            {
22                if (arr[k + i] >= arr[k])
23                    break;
24                else
25                {
26                    swap(&arr[k], &arr[k + i]);
27                }
28            }
29        }
30    }
31 }
32
33
34 int main(int argc, char **argv)
35 {
36    sampler_init(&argc, argv);
37    const int num = 200;
38    float my_max = 100.0;
39    float arr[num];
40    int k;
41
42    for (k = 0; k < num; k++)
43    {
44        arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
45    }
46
47    SAMPLE;
48    shellsort(arr, num);
49    SAMPLE;
50    return 0;
51 }
```

ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "sampler.h"
4
5 void swap(float* x, float* y)
6 {
7    float temp;
8    temp = *x;
9    *x = *y;
10    *y = temp;
11 }
12
13 void shellsort(float arr[], int num)
14 {
15    SAMPLE;
16    int i, j, k;
17    SAMPLE;
18    for (i = num / 2; i > 0; i = i / 2)
19    {
20       SAMPLE;
21        for (j = i; j < num; j++)
22        {
23            SAMPLE;
24            for (k = j - i; k >= 0; k = k - i)
25            {
26                if (arr[k + i] >= arr[k])
27                    break;
28                else
29                {
30                    SAMPLE;
31                    swap(&arr[k], &arr[k + i]);
32                    SAMPLE;
33                }
34            }
35            SAMPLE;
36        }
37        SAMPLE;
38    }
39    SAMPLE;
40 }
41
42
43 int main(int argc, char **argv)
44 {
45
46    sampler_init(&argc, argv);
47    const int num = 200;
48    float my_max = 100.0;
```

```
49      float arr[num];
50      int k;
51
52      for (k = 0; k < num; k++)
53      {
54          arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
55      }
56
57      shellsort(arr, num);
58      return 0;
59  }
```

ПРИЛОЖЕНИЕ В.

КОД ПРОГРАММЫ С ОПТИМИЗАЦИЯМИ

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "sampler.h"
4
5 void swap(float* x, float* y)
6 {
7    float temp;
8    temp = *x;
9    *x = *y;
10    *y = temp;
11 }
12
13 void shellsort(float arr[], int num)
14 {
15    SAMPLE;
16    int i, j, k;
17    float *tmp;
18    SAMPLE;
19    for (i = num / 2; i > 0; i = i / 2)
20    {
21       SAMPLE;
22        for (j = i; j < num; j++)
23        {
24            SAMPLE;
25            for (k = j - i; k >= 0; k = k - i)
26            {
27                 tmp = arr + k;
28                if (*(tmp + i) >= *(tmp))
29                    break;
30                else
31                {
32                    SAMPLE;
33                    swap(tmp, tmp + i);
34                    SAMPLE;
35                }
36            }
37            SAMPLE;
38        }
39        SAMPLE;
40    }
41    SAMPLE;
42 }
43
44
45 int main(int argc, char **argv)
46 {
47    sampler_init(&argc, argv);
48    const int num = 200;
```

```
49    float my_max = 100.0;
50    float arr[num];
51    int k;
52
53     for (k = 0; k < num; k++)
54    {
55         arr[k] = (float)rand() / (float)(RAND_MAX / my_max);
56     }
57
58     shellsort(arr, num);
59     return 0;
60}
```