

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»

Студентка гр. 8304

Николаева М. А.

Преподаватель

Кириянчиков 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/libsampler -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.

Были выполнены под управлением монитора SAMPLER_v2 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	2878.651	1	2878.651
15	17	5660.858	1	5660.858
17	19	22950.709	1	22950.709
19	21	30330.478	1	30330.478
21	24	2619.210	1	2619.210
24	27	7182.633	1	7182.633
27	30	21143.421	1	21143.421
30	33	32671.630	1	32671.630
33	39	2729.829	1	2729.829
39	45	6527.264	1	6527.264
45	51	16821.247	1	16821.247
51	57	34012.077	1	34012.077

Рисунок 1 – Результат работы SAMPLER для программы test_cyc.c

исх	прием	общее время	кол-во проходов	среднее время
30	32	28784993.156	1	28784993.156
32	34	56977840.711	1	56977840.711
34	36	142577719.133	1	142577719.133
36	38	284360479.033	1	284360479.033

Рисунок 2 - Результат работы SAMPLER для программы test_sub.c

Была выполнена под управлением монитора SAMPLER_v2 программа из лабораторной работы №1. Результат измерений полного времени выполнения функции *romb* представлен на рисунке 3. Исходный код этой программы представлен в Приложении А.

исх	прием	общее время	кол-во проходов	среднее время
90	92	1132.756	1	1132.756

Рисунок 3 - Результат работы SAMPLER_v2 для измерения полного времени выполнения функции *romb*

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
108	18	22.878	1	22.878
18	35	70.250	1	70.250
35	38	0.997	1	0.997
38	47	34.325	6	5.721
47	50	86.758	6	14.460
50	54	1438.738	63	22.837
54	56	17.494	6	2.916
54	50	183.814	57	3.225
56	61	22.278	6	3.713
61	64	22.645	6	3.774
64	69	353.857	21	16.850
69	71	51.039	6	8.506
69	64	40.507	15	2.700
71	73	104.396	6	17.399
73	94	14.639	3	4.880
73	76	10.221	3	3.407
94	38	12.240	5	2.448
94	97	9.692	1	9.692
76	78	12.291	3	4.097
78	90	34.117	2	17.059
78	82	13.489	1	13.489
90	92	30.831	3	10.277
92	94	3.510	3	1.170
82	90	2.187	1	2.187
97	110	28.853	1	28.853

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

В итоге общее время выполнения – 2622,02 мкс. Разницу в 1489,27 мкс с измерением для полного времени можно объяснить большим количеством измерений (меток SAMPLE).

Как видно из результатов измерения времени выполнения функциональных участков – наиболее затратными фрагментами являются циклы *for* (строчки 50-54 и

64-69). Для решения этой проблемы вычисления в строках 51-53 были объединены в одну строку, деление было заменено побитовым сдвигом, вызов функции был убран. В строках 65-68 в вычислениях доступ к элементам массива был заменен адресной арифметикой.

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
106	18	25.433	1	25.433
18	35	62.454	1	62.454
35	38	0.022	1	0.022
38	47	27.131	6	4.522
47	50	30.111	6	5.019
50	52	966.395	63	15.340
52	54	54.951	6	9.158
52	50	94.288	57	1.654
54	59	15.751	6	2.625
59	62	41.492	6	6.915
62	67	374.342	21	17.826
67	69	72.899	6	12.150
67	62	11.358	15	0.757
69	71	19.250	6	3.208
71	92	20.871	3	6.957
71	74	11.538	3	3.846
92	38	11.351	5	2.270
92	95	9.366	1	9.366
74	76	17.910	3	5.970
76	88	30.638	2	15.319
76	80	13.570	1	13.570
88	90	15.609	3	5.203
90	92	4.349	3	1.450
80	88	0.248	1	0.248
95	108	29.982	1	29.982

В результате внесённых изменений общее время выполнения – 1961,31 мкс. Удалось добиться сокращения времени выполнения на 660,7 мкс (25%).

Выводы.

В ходе выполнения лабораторной работы были изучены возможности измерения динамических характеристик программ с помощью профилировщиков на примере профилировщика SAMPLER_v2. Для программы, взятой из первой лабораторной работы, было выполнено измерение времени работы, с последующим

выявлением узкого места и его устранения. В результате удалось добиться более быстрого выполнения программы.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
1 #include <stdio.h>
2 #include <math.h>
3 #include "sampler.h"
4
5 typedef int bool;
6 #define true 1
7 #define false 0
8
9
10 float fx (float x)
11 {
12     return (1.0 / x);
13 }
14
15
16 float romb (float lower, float upper, float tol)
17 {
18     int nx [16];
19     float t [136];
20
21     bool done = false;
22     bool error = false;
23     int pieces = 1;
24     nx[1] = 1;
25     float delta_x = (upper - lower) / pieces;
26     float c = (fx(lower) + fx(upper)) * 0.5;
27     t[1] = delta_x * c;
28     int n = 1;
29     int nn = 2;
30     float sum = c;
31
32     float fotom,x;
33     int l,i,j,k,nt,ntra;
34     do
35     {
36         n = n+1;
37         fotom = 4;
38         nx[n] = nn;
39         pieces = pieces * 2;
40         l = pieces - 1;
41         delta_x = (upper - lower) / pieces;
42
43         int ll = (l+1)/2;
44         for(int ii = 1; ii <= ll; ii++)
45         {
46             i = ii * 2 - 1;
47             x = lower + i * delta_x;
48             sum = sum + fx(x);
49         }
50
51         t[nn] = delta_x * sum;
```

```

52
53     ntra = nx[n-1];
54     k = n-1;
55
56     for(int m = 1; m <= k; m++)
57     {
58         j = nn+m;
59         nt = nx[n - 1] + m - 1;
60         t[j] = (fotom * t[j - 1] - t[nt]) / (fotom-1.0);
61         fotom = fotom * 4;
62     }
63
64     if (n > 4)
65     {
66         if (t[nn + 1] != 0.0) {
67             if ((fabs(t[ntra+1]-t[nn+1])<=fabs(t[nn+1]*tol))
68                 || (fabs(t[nn-1]-t[j])<=fabs(t[j]*tol)))
69             {
70                 done = true;
71             } else
72                 if (n>15) {
73                     done = true;
74                     error = true;
75                 }
76         }
77     }
78     nn = j+1;
79 } while (!done);
80
81 return (t[j]);
82 }
83
84 int main(int argc, char **argv)
85 {
86     sampler_init(&argc, argv);
87     const float tol = 1.0E-4;
88     float lower = 1.0;
89     float upper = 9.0;
90     SAMPLE;
91     float sum = romb(lower,upper,tol);
92     SAMPLE;
93     return 0;
94 }

```


ПРИЛОЖЕНИЕ Б.

КОД ПРОГРАММЫ С РАЗДЕЛЕНИЕМ НА ФУ

```
1 #include <stdio.h>
2 #include <math.h>
3 #include "sampler.h"
4
5 typedef int bool;
6 #define true 1
7 #define false 0
8
9
10 float fx (float x)
11 {
12     return (1.0 / x);
13 }
14
15
16 float romb (float lower, float upper, float tol)
17 {
18     SAMPLE;
19     int nx [16];
20     float t [136];
21
22     bool done = false;
23     bool error = false;
24     int pieces = 1;
25     nx[1] = 1;
26     float delta_x = (upper - lower) / pieces;
27     float c = (fx(lower) + fx(upper)) * 0.5;
28     t[1] = delta_x * c;
29     int n = 1;
30     int nn = 2;
31     float sum = c;
32
33     float fotom,x;
34     int l,i,j,k,nt,ntra;
35     SAMPLE;
36     do
37     {
38         SAMPLE;
39         n = n+1;
40         fotom = 4;
41         nx[n] = nn;
42         pieces = pieces * 2;
43         l = pieces - 1;
44         delta_x = (upper - lower) / pieces;
45
46         int ll = (l+1)/2;
47         SAMPLE;
48         for(int ii = 1; ii <= ll; ii++)
```

```

49     {
50     SAMPLE;
51     i = ii * 2 - 1;
52     x = lower + i * delta_x;
53     sum = sum + fx(x);
54     SAMPLE;
55     }
56     SAMPLE;
57     t[nn] = delta_x * sum;
58
59     ntra = nx[n-1];
60     k = n-1;
61     SAMPLE;
62     for(int m = 1; m <= k; m++)
63     {
64         SAMPLE;
65         j = nn+m;
66         nt = nx[n - 1] + m - 1;
67         t[j] = (fotom * t[j - 1] - t[nt]) / (fotom-1.0);
68         fotom = fotom * 4;
69         SAMPLE;
70     }
71     SAMPLE;
72
73     SAMPLE;
74     if (n > 4)
75     {
76         SAMPLE;
77         if (t[nn + 1] != 0.0) {
78             SAMPLE;
79             if ((fabs(t[ntra+1]-t[nn+1])<=fabs(t[nn+1]*tol))
80                 || (fabs(t[nn-1]-t[j])<=fabs(t[j]*tol)))
81             {
82                 SAMPLE;
83                 done = true;
84             } else
85             if (n>15) {
86                 SAMPLE;
87                 done = true;
88                 error = true;
89             }
90             SAMPLE;
91         }
92         SAMPLE;
93     }
94     SAMPLE;
95     nn = j+1;
96 } while (!done);
97 SAMPLE;
98
99 return (t[j]);
100 }
101

```

```
102 int main(int argc, char **argv)
103 {
104     sampler_init(&argc, argv);
105     const float tol = 1.0E-4;
106     float lower = 1.0;
107     float upper = 9.0;
108     SAMPLE;
109     float sum = romb(lower,upper,tol);
110     SAMPLE;
111     return 0;
112 }
```

ПРИЛОЖЕНИЕ В.

КОД ПРОГРАММЫ С ОПТИМИЗАЦИЯМИ

```
1 #include <stdio.h>
2 #include <math.h>
3 #include "sampler.h"
4
5 typedef int bool;
6 #define true 1
7 #define false 0
8
9
10 float fx (float x)
11 {
12     return (1.0 / x);
13 }
14
15
16 float romb (float lower, float upper, float tol)
17 {
18     SAMPLE;
19     int nx [16];
20     float t [136];
21
22     bool done = false;
23     bool error = false;
24     int pieces = 1;
25     nx[1] = 1;
26     float delta_x = (upper - lower) / pieces;
27     float c = (fx(lower) + fx(upper)) * 0.5;
28     t[1] = delta_x * c;
29     int n = 1;
30     int nn = 2;
31     float sum = c;
32
33     float fotom,x;
34     int l,i,j,k,nt,ntra;
35     SAMPLE;
36     do
37     {
38         SAMPLE;
39         n = n+1;
40         fotom = 4;
41         nx[n] = nn;
42         pieces = pieces * 2;
43         l = pieces - 1;
44         delta_x = (upper - lower) / pieces;
45
46         int ll = (l+1) >> 1;
47         SAMPLE;
48         for(int ii = 1; ii <= ll; ++ii)
```

```

49     {
50     SAMPLE;
51     sum += 1.0 / (lower + ((ii << 1) - 1) * delta_x);
52     SAMPLE;
53     }
54     SAMPLE;
55     t[nn] = delta_x * sum;
56
57     ntra = nx[n-1];
58     k = n-1;
59     SAMPLE;
60     for(int m = 1; m <= k; ++m)
61     {
62         SAMPLE;
63         j = nn+m;
64         nt = (*(nx + n - 1)) + m - 1;
65         t[j] = (fotom * (*(t + j - 1)) - (*(t+nt))) / (fotom-
1.0);
66         fotom = fotom * 4;
67         SAMPLE;
68     }
69     SAMPLE;
70
71     SAMPLE;
72     if (n > 4)
73     {
74         SAMPLE;
75         if (t[nn + 1] != 0.0) {
76             SAMPLE;
77             if ((fabs(t[ntra+1]-t[nn+1])<=fabs(t[nn+1]*tol))
78             || (fabs(t[nn-1]-t[j])<=fabs(t[j]*tol)))
79             {
80                 SAMPLE;
81                 done = true;
82             } else
83             if (n>15) {
84                 SAMPLE;
85                 done = true;
86                 error = true;
87             }
88             SAMPLE;
89         }
90         SAMPLE;
91     }
92     SAMPLE;
93     nn = j+1;
94 } while (!done);
95 SAMPLE;
96
97 return (t[j]);
98 }
99
100 int main(int argc, char **argv)

```

```
101 {  
102     sampler_init(&argc, argv);  
103     const float tol = 1.0E-4;  
104     float lower = 1.0;  
105     float upper = 9.0;  
106     SAMPLE;  
107     float sum = romb(lower,upper,tol);  
108     SAMPLE;  
109     return 0;  
110 }
```