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

ОТЧЁТ

по лабораторной работе №3
по дисциплине «Качество и метрология программного обеспечения»
Тема: Измерение характеристик динамической сложности программ с помощью профилировщика SAMPLER

Студент гр. 6304 _____ Корытов П.В.
Преподаватель _____ Кирьянчиков В.А.

Санкт-Петербург
2020

1. Формулировка задания

1. Ознакомиться с документацией на монитор SAMPLER и выполнить под его управлением тестовые программы test_cyc.c и test_sub.c с анализом параметров повторения циклов, структуры описания циклов, способов профилирования процедур и проверкой их влияния на точность и чувствительность профилирования.
2. Скомпилировать и выполнить под управлением SAMPLER'а программу на С, разработанную в 1-ой лабораторной работе.

   Выполнить разбиение программы на функциональные участки и снять профили для двух режимов:

   1. измерение только полного времени выполнения программы;
   2. измерение времен выполнения функциональных участков (ФУ).

   Убедиться, что сумма времен выполнения ФУ соответствует полному времени выполнения программы. *Замечание: Следует внимательно подойти к выбору ФУ для получения хороших результатов профилирования.*
3. Выявить "узкие места", связанные с ухудшением производительности программы, ввести в программу усовершенствования и получить новые профили. Объяснить смысл введенных модификаций программ.

Примечания

1. Для трансляции программ следует использовать компиляторы Turbo C++, ver.3.0 (3.1, 3.2).
2. Для выполнения работы лучше использовать более новую версию монитора Sampler_new и выполнять ее на 32-разрядной машине под управлением ОС не выше Widows XP (при отсутствии реальных средств можно использовать виртуальные).

   В этом случае результаты профилирования будут близки к реальным.

   Если использовать более старую версию монитора Sampler_old, то ее следует запускать под эмулятором DOSBox-0.74. При этом времена, полученные в профилях будут сильно (примерно в 10 раз) завышены из-за накладных затрат эмулятора, но относительные соотношения временных затрат будут корректны.
3. Если автоматически подобрать время коррекции правильно не удается (это видно по большим значениям измерений времени для коротких фраг-

ментов программы, если время коррекции недостаточно, либо по большому количеству нулевых отсчетов, если время коррекции слишком велико), то следует подобрать подходящее время коррекции ручным способом, уменьшая или увеличивая его в нужную сторону.

4. Так как чувствительность SAMPLER'а по времени достаточно высока (на уровне единиц микросекунд), то вводить вспомогательное зацикливание программы обычно не требуется. Но если измеренные времена явно некорректны, следует ввести зацикливание выполнения программы в 10–100 раз. При этом для каждого повторения выполнения программы следует использовать одни и те же исходные данные.
5. Для обеспечения проверки представляемых вами профилей преподавателем необходимо выполнить **нумерацию строк кода программы**, соответствующую нумерации строк, указанной в профиле. У преподавателя нет времени для подсчета номеров строк в отчете каждого студента.

2. Выполнение работы

2.1. Выполнение тестовых программ

Использована новая версия Sampler.

В программах `test_cyc.cpp` и `test_sub.cpp` расставлены контрольные точки. Коды программ в приложении А.

Логи работы Sampler приведены в листингах 1 и 2.

Как видно, результаты правдоподобны — увеличение числа итераций приводит к пропорциональному увеличению времени выполнения программы.

Листинг 1. Результаты профилировки test_cyc

```
-------------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Вер-ть Среднее время(мкс)
-------------------------------------------------------------------------------
 1 :   7  1 :   9               3.63            1   0.00               3.63
-------------------------------------------------------------------------------
 1 :   9  1 :  11               7.26            1   0.00               7.26
-------------------------------------------------------------------------------
 1 :  11  1 :  13              17.32            1   0.00              17.32
-------------------------------------------------------------------------------
 1 :  13  1 :  15              34.36            1   0.00              34.36
-------------------------------------------------------------------------------
 1 :  15  1 :  18               3.91            1   0.00               3.91
-------------------------------------------------------------------------------
 1 :  18  1 :  21               7.26            1   0.00               7.26
-------------------------------------------------------------------------------
 1 :  21  1 :  24              17.04            1   0.00              17.04
-------------------------------------------------------------------------------
 1 :  24  1 :  27              34.08            1   0.00              34.08
-------------------------------------------------------------------------------
 1 :  27  1 :  33               3.63            1   0.00               3.63
-------------------------------------------------------------------------------
 1 :  33  1 :  39               7.26            1   0.00               7.26
-------------------------------------------------------------------------------
 1 :  39  1 :  45              17.32            1   0.00              17.32
-------------------------------------------------------------------------------
 1 :  45  1 :  51              34.36            1   0.00              34.36
-------------------------------------------------------------------------------
```

Листинг 2. Результаты профилировки test_sub

```
-------------------------------------------------------------------------------
Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Вер-ть Среднее время(мкс)
-------------------------------------------------------------------------------
 1 :  29  1 :  31             287.75            1   0.00             287.75
-------------------------------------------------------------------------------
 1 :  31  1 :  33             574.93            1   0.00             574.93
-------------------------------------------------------------------------------
 1 :  33  1 :  35            1442.64            1   0.00            1442.64
-------------------------------------------------------------------------------
 1 :  35  1 :  37            2980.27            1   0.00            2980.27
-------------------------------------------------------------------------------
```

2.2. Профилирование программы из ЛР1

Программа из ЛР1 модифицирована для компиляции под Turbo C++ 3.1 1991-го года выпуска. Для упрощения также убрана петля ввода-вывода. Код программы в приложении Б.

Проведена профилировка процедуры `solve`. Варианты снятия профиля приведены на листингах 3 и 4.

Листинг 3. Установка точек для измерения полного времени

```
116 SAMPLE;
117 error = solve(a, y, coef);
118 SAMPLE;
```

Листинг 4. Установка точек для функциональных участков

```
 83 bool solve(float** a, float* y, float* coef) {
 84     SAMPLE;
 85     float** b = _alloc_matr(RMAX, CMAX);
 86     float det = 0;
 87     SAMPLE;
 88     for (int i = 0; i < RMAX; i++) {
 89         for (int j = 0; j < CMAX; j++) {
 90             b[i][j] = a[i][j];
 91         }
 92     }
 93     SAMPLE;
 94     det = deter(b);
 95     SAMPLE;
 96     if (det == 0) {
 97         printf("ERROR: matrix is singular.");
 98         return true;
 99     }
100     SAMPLE;
101     setup(a, b, coef, y, 0, det);
102     SAMPLE;
103     setup(a, b, coef, y, 1, det);
104     SAMPLE;
105     setup(a, b, coef, y, 2, det);
106     SAMPLE;
107     _free_matr(b, RMAX);
108     SAMPLE;
109     return false;
110 }
```

Соответствующие результаты работы `Sampler` приведены на листингах 5

и 6.

Листинг 5. Результаты профилировки для полного времени

```
20 ------------------------------------------------------------------------------
21 Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Вер-ть Среднее время(мкс)
22 ------------------------------------------------------------------------------
23  1 : 103  1 : 105              121.43             1   0.00             121.43
24 ------------------------------------------------------------------------------
```

Листинг 6. Результаты профилировки функциональных участков

```
20 ------------------------------------------------------------------------------
21 Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Вер-ть Среднее время(мкс)
22 ------------------------------------------------------------------------------
23  1 :  84  1 :  87               12.57             1   0.00              12.57
24 ------------------------------------------------------------------------------
25  1 :  87  1 :  93               12.01             1   0.00              12.01
26 ------------------------------------------------------------------------------
27  1 :  93  1 :  95               12.01             1   0.00              12.01
28 ------------------------------------------------------------------------------
29  1 :  95  1 : 100               11.73             1   0.00              11.73
30 ------------------------------------------------------------------------------
31  1 : 100  1 : 102               34.92             1   0.00              34.92
32 ------------------------------------------------------------------------------
33  1 : 102  1 : 104               23.19             1   0.00              23.19
34 ------------------------------------------------------------------------------
35  1 : 104  1 : 106               12.01             1   0.00              12.01
36 ------------------------------------------------------------------------------
37  1 : 106  1 : 108                1.12             1   0.00               1.12
38 ------------------------------------------------------------------------------
39  1 : 108  1 : 127                0.28             1   0.00               0.28
40 ------------------------------------------------------------------------------
41  1 : 125  1 :  84                0.56             1   0.00               0.56
42 ------------------------------------------------------------------------------
```

Как можно заметить, суммарное время выполнения программы с листинга 4 примерно соответствует времени для листинга 3:

$$12.57+12.01+12.01+11.73+34.92+23.19+12.01+1.12+0.28 = 119.84, \quad (2.1)$$

$$\frac{119.84}{121.43} \approx 98.7\%. \quad (2.2)$$

2.3. Оптимизация программы

В оригинальной программе (Приложение А) в реализации метода Крамера производилось выделение памяти под новую матрицу; после чего на каждом шаге проводилась замена столбца матрицы на вектор с помощью процедуры на листинге 7.

Листинг 7. Оригинальная процедура `setup`

```
73  void setup(float** a, float** b, float* coef, float* y, int j, float det) {
74      for (int i = 0; i < RMAX; i++) {
75          b[i][j] = y[i];
76          if (j > 0) {
77              b[i][j-1] = a[i][j-1];
78          }
79      }
80      coef[j] = deter(b) / det;
81  }
```

В целях оптимизации выделение памяти убрано; вычисление запрограммировано напрямую. Оптимизированный код приведен на листинге 8.

Листинг 8. Оптимизация

```
73  float deter1(float** a, float* y) {
74      return y[0] * (a[1][1] * a[2][2] - a[2][1] * a[1][2])
75           - a[0][1] * (y[1] * a[2][2] - y[2] * a[1][2])
76           + a[0][2] * (y[1] * a[2][1] - y[2] * a[1][1]);
77  }
78
79  float deter2(float** a, float* y) {
80      return a[0][0] * (y[1] * a[2][2] - y[2] * a[1][2])
81           - y[0] * (a[1][0] * a[2][2] - a[2][0] * a[1][2])
82           + a[0][2] * (a[1][0] * y[2] - a[2][0] * y[1]);
83  }
84
85  float deter3(float** a, float* y) {
86      return a[0][0] * (a[1][1] * y[2] - a[2][1] * y[1])
87           - a[0][1] * (a[1][0] * y[2] - a[2][0] * y[1])
88           + y[0] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]);
89  }
```

Расстановка точек в новой программе приведена на листинге 9.

Результат профилирования представлены на листинге 10.

Как видно, время выполнения вычислений сократилось приблизительно в 2 раза.

Листинг 9. Расстановка точек в оптимизированной программе

```
91  bool solve(float** a, float* y, float* coef) {
92      SAMPLE;
93      float det = deter(a);
94      SAMPLE;
95      if (det == 0) {
96          printf("ERROR: matrix is singular.");
97          return true;
98      }
99      SAMPLE;
100     coef[0] = deter1(a, y) / det;
101     SAMPLE;
102     coef[1] = deter2(a, y) / det;
103     SAMPLE;
104     coef[2] = deter3(a, y) /det;
105     SAMPLE;
106     return false;
107 }
```

Листинг 10. Результаты профилировки для полного времени

```
20  ------------------------------------------------------------------------------
21  Исх.Поз. Прием.Поз.  Общее время(мкс)  Кол-во прох. Вер-ть Среднее время(мкс)
22  ------------------------------------------------------------------------------
23   1 :  92  1 :  94              10.90             1   0.00              10.90
24  ------------------------------------------------------------------------------
25   1 :  94  1 :  99              10.90             1   0.00              10.90
26  ------------------------------------------------------------------------------
27   1 :  99  1 : 101              11.45             1   0.00              11.45
28  ------------------------------------------------------------------------------
29   1 : 101  1 : 103              10.62             1   0.00              10.62
30  ------------------------------------------------------------------------------
31   1 : 103  1 : 105              10.90             1   0.00              10.90
32  ------------------------------------------------------------------------------
33   1 : 105  1 : 124               0.56             1   0.00               0.56
34  ------------------------------------------------------------------------------
35   1 : 122  1 :  92               0.56             1   0.00               0.56
36  ------------------------------------------------------------------------------
```

3. Выводы

Произведено вычисление профиля программы на C с помощью профилировщика `Sampler`.

С помощью профилировщика получены следующие утверждения:

- при увеличении числа итераций примерно пропорционально увеличивается время работы программы;
- полное время выполнения процедуры примерно равно сумме времен выполнения функциональных участков процедуры.

Из сохранения этих инвариантов следует, что результаты работы профилировщика, скорее всего, корректны.

Произведена оптимизация программы из ЛР1. Отказ от динамического выделения памяти и отказ от масштабируемости алгоритма повысили время выполнения вычислений приблизительно в 2 раза.

ПРИЛОЖЕНИЕ А

Коды тестовых программ

```
1  #define Size 10000
2  #include "Sampler.h"
3  int i, tmp, dim[Size];
4
5  void main()
6  {
7      SAMPLE;
8      for(i=0;i<Size/10;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp;
       ↪  };
9      SAMPLE;
10     for(i=0;i<Size/5;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp;
       ↪  };
11     SAMPLE;
12     for(i=0;i<Size/2;i++){ tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp;
       ↪  };
13     SAMPLE;
14     for(i=0;i<Size;i++)  { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp;
       ↪  };
15     SAMPLE;
16     for(i=0;i<Size/10;i++)
17     { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
18     SAMPLE;
19     for(i=0;i<Size/5;i++)
20     { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
21     SAMPLE;
22     for(i=0;i<Size/2;i++)
23     { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
24     SAMPLE;
25     for(i=0;i<Size;i++)
26     { tmp=dim[0]; dim[0]=dim[i]; dim[i]=tmp; };
27     SAMPLE;
28     for(i=0;i<Size/10;i++)
```

```
29      { tmp=dim[0];
30          dim[0]=dim[i];
31          dim[i]=tmp;
32      };
33      SAMPLE;
34      for(i=0;i<Size/5;i++)
35      { tmp=dim[0];
36          dim[0]=dim[i];
37          dim[i]=tmp;
38      };
39      SAMPLE;
40      for(i=0;i<Size/2;i++)
41      { tmp=dim[0];
42          dim[0]=dim[i];
43          dim[i]=tmp;
44      };
45      SAMPLE;
46      for(i=0;i<Size;i++)
47      { tmp=dim[0];
48          dim[0]=dim[i];
49          dim[i]=tmp;
50      };
51      SAMPLE;
52  }
```

```
1   const unsigned Size = 1000;
2
3
4   void TestLoop(int nTimes)
5   {
6     static int TestDim[Size];
7     int tmp;
8     int iLoop;
9
10    while (nTimes > 0)
```

```
11    {
12      nTimes --;
13
14      iLoop = Size;
15      while (iLoop > 0)
16      {
17        iLoop -- ;
18        tmp = TestDim[0];
19        TestDim[0] = TestDim[nTimes];
20        TestDim[nTimes] = tmp;
21      }
22    }
23  } /* TestLoop */
24
25
26  void main()
27  {
28     TestLoop(Size / 10); // 100 * 1000  повторений
29     TestLoop(Size / 5);  // 200 * 1000  повторений
30     TestLoop(Size / 2);  // 500 * 1000  повторений
31     TestLoop(Size / 1);  // 1000* 1000  повторений
32  }
```

ПРИЛОЖЕНИЕ Б

Код программы из ЛР1

```
1  #include ”stdio.h”
2  #include ”stdlib.h”
3  #include ”Sampler.h”
4
5  #define RMAX 3
6  #define CMAX 3
7
8  #define bool short int
9  #define true 1
10 #define false 0
11
12 float** _alloc_matr(int a, int b) {
13     float** m = (float**)malloc(a * sizeof(float*));
14     for (int i = 0; i < CMAX; i ++) {
15         m[i] = (float*)malloc(b * sizeof(float));
16     }
17     return m;
18 }
19
20 void _free_matr(float** m, int a) {
21     for (int i = 0; i < a; i ++) {
22         free(m[i]);
23     }
24     free(m);
25 }
26
27
28 /* print out the answers */
29 void print_matr(float** a, float* y) {
30     for (int i = 0; i < RMAX; i++) {
31         for (int j = 0; j < CMAX; j ++) {
32             printf(”%f  ”, a[i][j]);
```

```
33          }
34          printf(": %f\n", y[i]);
35      }
36  }
37
38  /* get the values for n, and arrays a,y */
39  void get_data(float** a, float* y) {
40      /* for (int i = 0; i < RMAX; i++) { */
41      /*     printf("Equation %d\n", i); */
42      /*     for (int j = 0; j < CMAX; j++) { */
43      /*         printf("%d: ", j); */
44      /*         scanf("%f", &a[i][j]); */
45      /*         a[i][j] = */
46      /*     } */
47      /*     printf("C: "); */
48      /*     scanf("%f", &y[i]); */
49      /* } */
50      a[0][0] = 12345;
51      a[0][1] = 23456;
52      a[0][2] = 34567;
53      y[0]    = 45678;
54      a[1][0] = 56789;
55      a[1][1] = 67890;
56      a[1][2] = 78901;
57      y[1]    = 89012;
58      a[2][0] = 90123;
59      a[2][1] = 12345;
60      a[2][2] = 23456;
61      y[2]    = 34567;
62      /* print_matr(a, y); */
63      /* printf("\n"); */
64  }
65
66  /* pascal program to calculate the determinant of a 3-by-3matrix */
```

```
67  float deter(float** a) {
68      return a[0][0] * (a[1][1] * a[2][2] - a [2][1] * a[1][2])
69           - a[0][1] * (a[1][0] * a[2][2] - a [2][0] * a[1][2])
70           + a[0][2] * (a[1][0] * a[2][1] - a [2][0] * a[1][1]);
71  }
72
73  void setup(float** a, float** b, float* coef, float* y, int j,
    ↪  float det) {
74      for (int i = 0; i < RMAX; i++) {
75          b[i][j] = y[i];
76          if (j > 0) {
77              b[i][j-1] = a[i][j-1];
78          }
79      }
80      coef[j] = deter(b) / det;
81  }
82
83  bool solve(float** a, float* y, float* coef) {
84      float** b = _alloc_matr(RMAX, CMAX);
85      float det = 0;
86      for (int i = 0; i < RMAX; i++) {
87          for (int j = 0; j < CMAX; j++) {
88              b[i][j] = a[i][j];
89          }
90      }
91      det = deter(b);
92      if (det == 0) {
93          printf("ERROR: matrix is singular.");
94          return true;
95      }
96      setup(a, b, coef, y, 0, det);
97      setup(a, b, coef, y, 1, det);
98      setup(a, b, coef, y, 2, det);
99      _free_matr(b, RMAX);
```

```
100     return false;
101 }
102
103 void write_data(float* coef) {
104     for (int i = 0; i < CMAX; i++) {
105         printf("%f ", coef[i]);
106     }
107     printf("\n");
108 }
109
110 int main() {
111     float** a = _alloc_matr(RMAX, CMAX);
112     float* y = (float*)malloc(CMAX * sizeof(float));
113     float* coef = (float*)malloc(CMAX * sizeof(float));
114     bool error;
115     get_data(a, y);
116     error = solve(a, y, coef);
117     if (!error) {
118         write_data(coef);
119     }
120     free(y);
121     free(coef);
122     _free_matr(a, RMAX);
123     return 0;
124 }
```