**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\_v2

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Алтухов А.Д. |
| Преподаватель |  | Ефремов М.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\_v2 тестовые программы test\_cyc.c и test\_sub.c. Результаты представлены на рисунках 1 и 2.

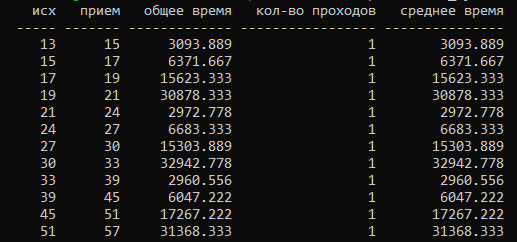


Рисунок 1 – Результат работы SAMPLER\_v2 для программы test\_cyc.c

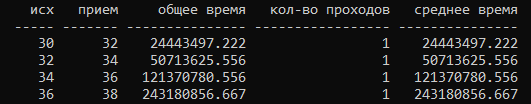


Рисунок 2 – Результат работы SAMPLER\_v2 для программы test\_sub.c

Под управлением монитора SAMPLER\_v2 была выполнена программа из лабораторной работы №1. Результат измерений полного времени выполнения функции simps представлен на рисунке 3. Исходный код программы представлен в приложении А.



Рисунок 3 – Результат работы SAMPLER\_v2 для выполнения функции simps

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\_v2 результаты представлены на рисунке 4.

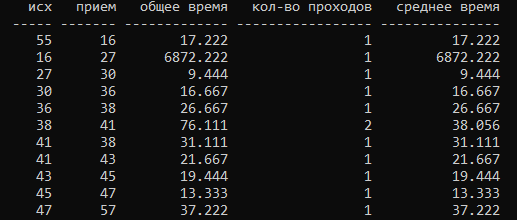


Рисунок 4 – Результат работы SAMPLER\_v2 для выполнения функции simps

Как видно из результатов измерения времени выполнения функциональных участков, наиболее затратным фрагментом являются строчки 16-27 с большим количеством вычислений. Для оптимизации работы программы были убраны вызовы дополнительных расчетных функций.

Была выполнена проверка измененной программы. Результат представлен на рисунке 5. Исходный код модифицированной программы представлен в приложении В.

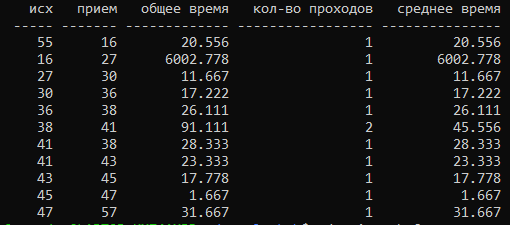


Рисунок 5 – Результат работы SAMPLER\_v2 для выполнения модифицированной функции simps

Время выполнения проблемного участка составило 6002,778 мкс, по сравнению с прошлыми 6872,222 мкс. Удалось добиться сокращения времени выполнения на 869,444 мкс (12%).

**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.**

#include <stdio.h>

#include <math.h>

#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){

return exp(-x/2);

}

double dfx(double x){

return -(exp(-x/2))/2;

}

double simps(double lower, double upper, double tol, double\* sum){

int pieces=2;

double delta\_x=(upper-lower)/pieces;

double odd\_sum = fx(lower+delta\_x);

double even\_sum =0.0;

double end\_sum =fx(lower)+fx(upper);

double end\_cor =dfx(lower)-dfx(upper);

\*sum=(end\_sum+4.0\*odd\_sum)\*delta\_x/3.0;

double sum1;

double x;

do

{

pieces=pieces\*2;

sum1=\*sum;

delta\_x=(upper-lower)/pieces;

even\_sum=even\_sum+odd\_sum;

odd\_sum=0.0;

for (int i=1; i<=pieces/2; i++) {

x=lower+delta\_x\*(2.0\*i-1.0);

odd\_sum=odd\_sum+fx(x);

}

\*sum=(7.0\*end\_sum+14.0\*even\_sum+16.00\*odd\_sum+end\_cor\*delta\_x)\*delta\_x/15.0;

} while ( (\*sum!=sum1) && (fabs(\*sum-sum1)<=fabs(tol\*(\*sum))) );

}

int main(int argc, char\*\* argv)

{

sampler\_init(&argc, argv);

double lower=1.0;

double upper=9.0;

double sum = 0.0;

SAMPLE;

simps(lower,upper,tol,&sum);

SAMPLE;

//printf("\narea= %f\n", sum);

return 0;

}

**Приложение Б.**

#include <stdio.h>

#include <math.h>

#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){

return exp(-x/2);

}

double dfx(double x){

return -(exp(-x/2))/2;

}

double simps(double lower, double upper, double tol, double\* sum){

SAMPLE;

int pieces=2;

double delta\_x=(upper-lower)/pieces;

double odd\_sum = fx(lower+delta\_x);

double even\_sum =0.0;

double end\_sum =fx(lower)+fx(upper);

double end\_cor =dfx(lower)-dfx(upper);

\*sum=(end\_sum+4.0\*odd\_sum)\*delta\_x/3.0;

double sum1;

double x;

SAMPLE;

do

{

SAMPLE;

pieces=pieces\*2;

sum1=\*sum;

delta\_x=(upper-lower)/pieces;

even\_sum=even\_sum+odd\_sum;

odd\_sum=0.0;

SAMPLE;

for (int i=1; i<=pieces/2; i++) {

SAMPLE;

x=lower+delta\_x\*(2.0\*i-1.0);

odd\_sum=odd\_sum+fx(x);

SAMPLE;

}

SAMPLE;

\*sum=(7.0\*end\_sum+14.0\*even\_sum+16.00\*odd\_sum+end\_cor\*delta\_x)\*delta\_x/15.0;

SAMPLE;

} while ( (\*sum!=sum1) && (fabs(\*sum-sum1)<=fabs(tol\*(\*sum))) );

SAMPLE;

}

int main(int argc, char\*\* argv)

{

sampler\_init(&argc, argv);

double lower=1.0;

double upper=9.0;

double sum = 0.0;

SAMPLE;

simps(lower,upper,tol,&sum);

SAMPLE;

//printf("\narea= %f\n", sum);

return 0;

}

**Приложение В.**

#include <stdio.h>

#include <math.h>

#include "sampler.h"

const double tol = 1.0E-6;

double fx(double x){

return exp(-x/2);

}

double dfx(double x){

return -(exp(-x/2))/2;

}

double simps(double lower, double upper, double tol, double\* sum){

SAMPLE;

int pieces=2;

double delta\_x=(upper-lower)/pieces;

double odd\_sum = exp(-(lower+delta\_x)/2);

double even\_sum =0.0;

double end\_sum =exp(-lower/2)+exp(-upper/2);

double end\_cor =-(exp(-lower/2))/2-(-(exp(-upper/2))/2);

\*sum=(end\_sum+4.0\*odd\_sum)\*delta\_x/3.0;

double sum1;

double x;

SAMPLE;

do

{

SAMPLE;

pieces=pieces\*2;

sum1=\*sum;

delta\_x=(upper-lower)/pieces;

even\_sum=even\_sum+odd\_sum;

odd\_sum=0.0;

SAMPLE;

for (int i=1; i<=pieces/2; i++) {

SAMPLE;

x=lower+delta\_x\*(2.0\*i-1.0);

odd\_sum=odd\_sum+fx(x);

SAMPLE;

}

SAMPLE;

\*sum=(7.0\*end\_sum+14.0\*even\_sum+16.00\*odd\_sum+end\_cor\*delta\_x)\*delta\_x/15.0;

SAMPLE;

} while ( (\*sum!=sum1) && (fabs(\*sum-sum1)<=fabs(tol\*(\*sum))) );

SAMPLE;

}

int main(int argc, char\*\* argv)

{

sampler\_init(&argc, argv);

double lower=1.0;

double upper=9.0;

double sum = 0.0;

SAMPLE;

simps(lower,upper,tol,&sum);

SAMPLE;

// printf("\narea= %f\n", sum);

return 0;

}