Київський національний університет

імені Тараса Шевченка

Лабораторна робота №6

з предмету “Системне програмування”

Виконав

Студент 3 курсу

Групи ТТП-32

Факультету комп’ютерних наук

та кібернетики

Бариш Артем Богданович

Київ

18.11.2023

**Умова завдання**

****

**Виконання**

Для виконання лабораторної роботи треба було прийнято рішення поставити віртуальну машину лінукс на віндовс, після чого було встановлено git та clion на дану віртуальну машину.

Було створено 2 різні варіанти програми: програми виконують однакові функції, але різниця між оптимізацією між ними.

**#include <bits/stdc++.h>**

**using namespace std;**

**int sumOfDivisors(int n){**

**int sum = 0;**

**int sum\_2 = 0;**

**for(int i = 1; i<=n; i++){**

**sum\_2=sum;**

**if(n%i==0){**

**sum\_2 = sum\_2 + i;**

**}**

**sum = sum\_2;**

**}**

**return sum;**

**}**

**int main() {**

**for(int i = 20; i<100; i++){**

**for(int j = 500; j<1000 ;j++){**

**cout<<"Sum of division "<<sumOfDivisors(i)+ sumOfDivisors(j)<<endl;**

**}**

**}**

**return 0;**

**}**

Та оптимізований код буде

#include <bits/stdc++.h>

using namespace std;

int sumOfDivisors(int n){

int sum = 0;

for(int i = 1; i<=n; i++){

if(n%i==0){

sum = sum + i;

}

}

return sum;

}

int main() {

for(int i = 20; i<100; i++){

for(int j = 500; j<1000 ;j++){

cout<<"Sum of division "<<sumOfDivisors(i)+ sumOfDivisors(j)<<endl;

}

}

return 0;

}

Тепер потрібно створити асемблер код для коєний функцій для цього будемо використовувати сайт godbolt.org зараз будуть наведені відповідно код для кожної з функції

sumOfDivisors(int):

pushq %rbp

movq %rsp, %rbp

subq $16, %rsp

movl %ecx, 16(%rbp)

movl $0, -4(%rbp)

movl $0, -8(%rbp)

movl $1, -12(%rbp)

jmp .L2

.L4:

movl -4(%rbp), %eax

movl %eax, -8(%rbp)

movl 16(%rbp), %eax

cltd

idivl -12(%rbp)

movl %edx, %eax

testl %eax, %eax

jne .L3

movl -12(%rbp), %eax

addl %eax, -8(%rbp)

.L3:

movl -8(%rbp), %eax

movl %eax, -4(%rbp)

addl $1, -12(%rbp)

.L2:

movl -12(%rbp), %eax

cmpl 16(%rbp), %eax

jle .L4

movl -4(%rbp), %eax

addq $16, %rsp

popq %rbp

ret

та оптимізований код

sumOfDivisors(int):

pushq %rbp

movq %rsp, %rbp

subq $16, %rsp

movl %ecx, 16(%rbp)

movl $0, -4(%rbp)

movl $1, -8(%rbp)

jmp .L2

.L4:

movl 16(%rbp), %eax

cltd

idivl -8(%rbp)

movl %edx, %eax

testl %eax, %eax

jne .L3

movl -8(%rbp), %eax

addl %eax, -4(%rbp)

.L3:

addl $1, -8(%rbp)

.L2:

movl -8(%rbp), %eax

cmpl 16(%rbp), %eax

jle .L4

movl -4(%rbp), %eax

addq $16, %rsp

popq %rbp

ret

1. Для початку зробимо флейм графи

## Спочатку треба склонувати проєкт який допоможе будувати флейм граф будемо використосувати гіт <https://github.com/brendangregg/FlameGraph>

## Для запуску коду потрібно виконати команди **perf record -F 50 --call-graph dwarf ./main** тепер треба виконати команду **perf script | /home/artem/CLionProjects/flame/FlameGraph/stackcollapse-perf.pl |** **/home/artem/CLionProjects/flame/FlameGraph/stackcollapse-recursive.pl |** **/home/artem/CLionProjects/flame/FlameGraph/flamegraph.pl > outMain.svg**

Ця команда створить граф для мейна, те саме треба зробити для optimized.ccp

## **perf record -F 50 --call-graph dwarf ./optimized** **perf script | /home/artem/CLionProjects/flame/FlameGraph/stackcollapse-perf.pl |** **/home/artem/CLionProjects/flame/FlameGraph/stackcollapse-recursive.pl |** **/home/artem/CLionProjects/flame/FlameGraph/flamegraph.pl > outOptimized.svg**

Наведемо команди для статистики

## /usr/bin/time --verbose ./main

 Command being timed: "./main"  
        User time (seconds): 0.55  
        System time (seconds): 1.00  
        Percent of CPU this job got: 13%  
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:11.95  
        Average shared text size (kbytes): 0  
        Average unshared data size (kbytes): 0  
        Average stack size (kbytes): 0  
        Average total size (kbytes): 0  
        Maximum resident set size (kbytes): 3328  
        Average resident set size (kbytes): 0  
        Major (requiring I/O) page faults: 0  
        Minor (reclaiming a frame) page faults: 137  
        Voluntary context switches: 175  
        Involuntary context switches: 2849  
        Swaps: 0  
        File system inputs: 0  
        File system outputs: 0  
        Socket messages sent: 0  
        Socket messages received: 0  
        Signals delivered: 0  
        Page size (bytes): 4096  
        Exit status: 0

## /usr/bin/time --verbose ./optimized

 Command being timed: "./optimized"  
        User time (seconds): 0.53  
        System time (seconds): 0.96  
        Percent of CPU this job got: 12%  
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:12.38  
        Average shared text size (kbytes): 0  
        Average unshared data size (kbytes): 0  
        Average stack size (kbytes): 0  
        Average total size (kbytes): 0  
        Maximum resident set size (kbytes): 3456  
        Average resident set size (kbytes): 0  
        Major (requiring I/O) page faults: 0  
        Minor (reclaiming a frame) page faults: 136  
        Voluntary context switches: 187  
        Involuntary context switches: 2290  
        Swaps: 0  
        File system inputs: 0  
        File system outputs: 0  
        Socket messages sent: 0  
        Socket messages received: 0  
        Signals delivered: 0  
        Page size (bytes): 4096  
        Exit status: 0

## perf stat -d ./main

Performance counter stats for './main':  
  
          1 346,62 msec task-clock                       #    0,101 CPUs utilized              
             2 841      context-switches                 #    2,110 K/sec                      
                15      cpu-migrations                   #   11,139 /sec                        
               125      page-faults                      #   92,825 /sec                        
   <not supported>      cycles                                                                  
   <not supported>      instructions                                                            
   <not supported>      branches                                                                
   <not supported>      branch-misses                                                          
   <not supported>      L1-dcache-loads                                                        
   <not supported>      L1-dcache-load-misses                                                  
   <not supported>      LLC-loads                                                              
   <not supported>      LLC-load-misses                                                        
  
      13,335672292 seconds time elapsed  
  
       0,493898000 seconds user  
       0,858709000 seconds sys

## perf stat -d ./ optimized

 Performance counter stats for './optimized':  
  
          1 465,68 msec task-clock                       #    0,111 CPUs utilized              
             1 449      context-switches                 #  988,622 /sec                        
                 6      cpu-migrations                   #    4,094 /sec                        
               124      page-faults                      #   84,603 /sec                        
   <not supported>      cycles                                                                  
   <not supported>      instructions                                                            
   <not supported>      branches                                                                
   <not supported>      branch-misses                                                          
   <not supported>      L1-dcache-loads                                                        
   <not supported>      L1-dcache-load-misses                                                  
   <not supported>      LLC-loads                                                              
   <not supported>      LLC-load-misses                                                        
  
      13,202772268 seconds time elapsed  
  
       0,451480000 seconds user  
       1,029780000 seconds sys

## perf record ./main

[ perf record: Woken up 1 times to write data ]  
[ perf record: Captured and wrote 0,124 MB perf.data (3206 samples) ]

## perf report

32,04%  main     [unknown]             [k] 0xffffffffa17125c8                                                                   ◆  
  27,51%  main     main                  [.] sumOfDivisors                                                                        ▒  
   6,12%  main     [unknown]             [k] 0xffffffffa1734b0a                                                                   ▒  
   5,04%  main     [unknown]             [k] 0xffffffffa26af26e                                                                   ▒  
   4,71%  main     [unknown]             [k] 0xffffffffc0467ee9                                                                   ▒  
   3,23%  main     [unknown]             [k] 0xffffffffa26adc71                                                                   ▒  
   1,00%  main     libc.so.6             [.] \_\_GI\_\_\_libc\_write                                                                    ▒  
   0,89%  main     libstdc++.so.6.0.30   [.] std::ostream::put                                                                    ▒  
   0,78%  main     [unknown]             [k] 0xffffffffa26a6f75                                                                   ▒  
   0,74%  main     libc.so.6             [.] \_IO\_fwrite                                                                           ▒  
   0,67%  main     libc.so.6             [.] \_IO\_file\_write@@GLIBC\_2.2.5                                                          ▒  
   0,48%  main     libc.so.6             [.] \_\_strlen\_sse2                                                                        ▒  
   0,44%  main     libc.so.6             [.] \_IO\_do\_write@@GLIBC\_2.2.5                                                            ▒  
   0,44%  main     libc.so.6             [.] \_IO\_fflush                                                                           ▒  
   0,44%  main     libc.so.6             [.] \_IO\_file\_xsputn@@GLIBC\_2.2.5                                                         ▒  
   0,44%  main     libstdc++.so.6.0.30   [.] std::ostream::sentry::sentry                                                         ▒  
   0,41%  main     libc.so.6             [.] \_\_memcpy\_ssse3                                                                       ▒  
   0,41%  main     [unknown]             [k] 0xffffffffa17c5881                                                                   ▒  
   0,37%  main     libstdc++.so.6.0.30   [.] std::endl<char, std::char\_traits<char> >                                             ▒  
   0,33%  main     libstdc++.so.6.0.30   [.] std::num\_put<char, std::ostreambuf\_iterator<char, std::char\_traits<char> > >::do\_put ▒  
   0,33%  main     [unknown]             [k] 0xffffffffa2698b70                                                                   ▒  
   0,33%  main     [unknown]             [k] 0xffffffffa26ae585                                                                   ▒  
   0,30%  main     libstdc++.so.6.0.30   [.] \_\_gnu\_cxx::stdio\_sync\_filebuf<char, std::char\_traits<char> >::sync                   ▒  
   0,26%  main     [unknown]             [k] 0xffffffffa2021a06                                                                   ▒  
   0,26%  main     [unknown]             [k] 0xffffffffa2025ba0                                                                   ▒  
   0,26%  main     [unknown]             [k] 0xffffffffa26a8594                                                                   ▒  
   0,22%  main     libc.so.6             [.] \_\_mempcpy\_ssse3                                                                      ▒

## perf record ./optimized

 perf record: Woken up 1 times to write data ]  
[ perf record: Captured and wrote 0,188 MB perf.data (4883 samples) ]

## perf report

41,68%  optimized  [unknown]             [k] 0xffffffffa17125c8                                                                 ◆  
  20,41%  optimized  optimized             [.] sumOfDivisors                                                                      ▒  
   8,58%  optimized  [unknown]             [k] 0xffffffffa26af26e                                                                 ▒  
   5,97%  optimized  [unknown]             [k] 0xffffffffa1734b0a                                                                 ▒  
   3,60%  optimized  [unknown]             [k] 0xffffffffa26adc71                                                                 ▒  
   2,84%  optimized  [unknown]             [k] 0xffffffffc0467ee9                                                                 ▒  
   1,07%  optimized  libc.so.6             [.] \_\_GI\_\_\_libc\_write                                                                  ▒  
   0,69%  optimized  libstdc++.so.6.0.30   [.] std::ostream::put                                                                  ▒  
   0,52%  optimized  [unknown]             [k] 0xffffffffa26a6f75                                                                 ▒  
   0,47%  optimized  libc.so.6             [.] \_IO\_fwrite                                                                         ▒  
   0,43%  optimized  libc.so.6             [.] \_IO\_file\_write@@GLIBC\_2.2.5                                                        ▒  
   0,43%  optimized  libc.so.6             [.] \_\_memcpy\_ssse3                                                                     ▒  
   0,38%  optimized  libc.so.6             [.] \_IO\_file\_xsputn@@GLIBC\_2.2.5                                                       ▒  
   0,36%  optimized  libc.so.6             [.] \_\_strlen\_sse2                                                                      ▒  
   0,33%  optimized  libc.so.6             [.] \_IO\_fflush                                                                         ▒  
   0,31%  optimized  [unknown]             [k] 0xffffffffa17c5881                                                                 ▒  
   0,28%  optimized  [unknown]             [k] 0xffffffffa2698b70                                                                 ▒  
   0,26%  optimized  libc.so.6             [.] \_\_mempcpy\_ssse3                                                                    ▒  
   0,24%  optimized  libstdc++.so.6.0.30   [.] std::ostream::\_M\_insert<long>                                                      ▒  
   0,24%  optimized  libstdc++.so.6.0.30   [.] std::ostream::flush                                                                ▒  
   0,24%  optimized  [unknown]             [k] 0xffffffffa26931a7                                                                 ▒  
   0,24%  optimized  [unknown]             [k] 0xffffffffa26ae585

## 2)Команди для temperature/power analysis

Для початку треба скачати проэкт якиф допоможе побудувати цю статистиску

[**https://github.com/RRZE-HPC/likwid.git**](https://github.com/RRZE-HPC/likwid.git)

**потім треба виконати команду sudo apt install make,**

**Після чого виконуємо команди sudo apt likwid-powermeter -i**

CPU name:       Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz  
CPU type:       Intel Kabylake processor  
CPU clock:      1.98 GHz  
--------------------------------------------------------------------------------  
Base clock:     2000.00 MHz  
Minimal clock:  400.00 MHz  
Turbo Boost Steps:  
C0 4200.00 MHz  
C1 4200.00 MHz  
C2 4200.00 MHz  
C3 4200.00 MHz  
--------------------------------------------------------------------------------  
Info for RAPL domain PKG:  
Thermal Spec Power: 0 Watt  
Minimum Power: 0 Watt  
Maximum Power: 0 Watt  
Maximum Time Window: 0 micro sec  
  
Info for RAPL domain PLATFORM:  
Thermal Spec Power: 5120 Watt  
Minimum Power: 0 Watt  
Maximum Power: 5120 Watt  
Maximum Time Window: 0 micro sec  
  
Info about Uncore:  
Minimal Uncore frequency: 800 MHz  
Maximal Uncore frequency: 4100 MHz

Тепер температура за проміжок часу sudo likwid-powermeter –t 100s

CPU name: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz

CPU type: Intel Kabylake processor

CPU clock: 1.99 GHz

--------------------------------------------------------------------------------

Runtime: 100.107 s

Measure for socket 0 on CPU 0

Domain PKG:

Energy consumed: 8000 Joules

Power consumed: 80 Watts

Domain PP0:

Energy consumed: 6000 Joules

Power consumed: 60 Watts

Domain PP1:

Energy consumed: 2000 Joules

Power consumed: 20 Watts

Domain DRAM:

Energy consumed: 1000 Joules

Power consumed: 10 Watts

Domain PLATFORM:

Energy consumed: 500 Joules

Power consumed: 5 Watts

## likwid-powermeter ./main

sudo likwid-powermeter -i ./main  
--------------------------------------------------------------------------------  
CPU name: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz  
CPU type: Intel Kabylake processor  
CPU clock: 1.99 GHz  
--------------------------------------------------------------------------------  
Base clock: 2000.00 MHz  
Minimal clock: 400.00 MHz  
Turbo Boost Steps:  
C0 4200.00 MHz  
C1 4200.00 MHz  
C2 4200.00 MHz  
C3 4200.00 MHz  
--------------------------------------------------------------------------------  
Info for RAPL domain PKG:  
Thermal Spec Power: 0 Watt  
Minimum Power: 0 Watt  
Maximum Power: 0 Watt  
Maximum Time Window: 0 micro sec  
  
Info for RAPL domain PLATFORM:  
Thermal Spec Power: 5120 Watt  
Minimum Power: 0 Watt  
Maximum Power: 5120 Watt  
Maximum Time Window: 0 micro sec  
  
Info about Uncore:  
Minimal Uncore frequency: 800 MHz  
Maximal Uncore frequency: 4100 MHz  
  
Performance energy bias: 6 (0=highest performance, 15 = lowest energy)

## likwid-powermeter ./optimized

--------------------------------------------------------------------------------  
CPU name: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz  
CPU type: Intel Kabylake processor  
CPU clock: 1.99 GHz  
--------------------------------------------------------------------------------  
Base clock: 2000.00 MHz  
Minimal clock: 400.00 MHz  
Turbo Boost Steps:  
C0 4200.00 MHz  
C1 4200.00 MHz  
C2 4200.00 MHz  
C3 4200.00 MHz  
--------------------------------------------------------------------------------  
Info for RAPL domain PKG:  
Thermal Spec Power: 0 Watt  
Minimum Power: 0 Watt  
Maximum Power: 0 Watt  
Maximum Time Window: 0 micro sec  
  
Info for RAPL domain PLATFORM:  
Thermal Spec Power: 5120 Watt  
Minimum Power: 0 Watt  
Maximum Power: 5120 Watt  
Maximum Time Window: 0 micro sec  
  
Info about Uncore:  
Minimal Uncore frequency: 800 MHz  
Maximal Uncore frequency: 4100 MHz  
  
Performance energy bias: 6 (0=highest performance, 15 = lowest energy)

Асемблер код

Порівняємо неоптимізовний код та оптимізований

Порівнюючи два фрагменти асемблерного коду для функції sumOfDivisors(int), можна побачити, що вони реалізують однакову базову функціональність, але з деякими відмінностями у підході та ефективності. Обидві функції призначені для обчислення суми дільників даного цілого числа. Ось аналіз відмінностей:

Ініціалізація локальних змінних:

Перший код: Ініціалізує три локальні змінні за адресами -4(%rbp), -8(%rbp) і -12(%rbp) зі значеннями 0, 0 і 1 відповідно.

Другий код: Ініціалізує дві локальні змінні за адресами -4(%rbp) і -8(%rbp) зі значеннями 0 і 1 відповідно.

Спостереження: Другий код ефективніший з точки зору використання пам'яті, оскільки використовує на одну локальну змінну менше.

Цикл перевірки дільників:

Обидва коди: Реалізують цикл для перевірки кожного числа від 1 до введеного числа, ділячи введене число на цей ітератор і перевіряючи, чи залишок дорівнює 0 (що означає, що це дільник).

Відмінності: Перший код використовує додаткову локальну змінну (-8(%rbp)) для тимчасового зберігання суми дільників, яку потім передає в -4(%rbp). Другий код безпосередньо оновлює суму в -4(%rbp).

Ефективність:

Перший код включає більше операцій з перенесенням даних між регістрами та пам'яттю, що зазвичай є менш ефективним.

Другий код більш оптимізований з меншою кількістю інструкцій та меншим використанням пам'яті.

Чіткість та здатність до підтримки:

Перший код: Трохи складніший через додаткову локальну змінну та переноси між змінними.

Другий код: Простіший і легший для розуміння.

Функціональність:

Обидва коди досягають одного й того ж результату - сумують дільники введеного числа. Основний цикл та основна логіка (ділення введеного числа та перевірка на відсутність залишку) залишаються по суті ті самі.

Підсумовуючи, хоча обидва коди служать одній меті, другий код є більш лаконічним та ефективним, як з точки зору використання пам'яті, так і кількості виконуваних операцій. Це може потенційно призвести до трохи кращої продуктивності, особливо для великих вхідних чисел.

Тепер перейдемо до графу

Ширина блоків (Час виконання):

Більш широкі блоки на flame graph вказують на те, що функція займала більше часу. У неоптимізованому графіку ширші блоки можуть свідчити про надмірне використання певних функцій, зациклення або неефективні алгоритми. Натомість, у оптимізованому графіку вужчі блоки свідчать про швидше виконання функцій, показуючи ефективність використання часу ЦПУ.

Глибина стеків (Виклики функцій):

Глибші стеки в неоптимізованому графіку можуть вказувати на більшу кількість вкладених викликів функцій, що може свідчити про зайву складність алгоритму чи надмірне використання рекурсії. Навпаки, менша глибина у оптимізованому графіку може свідчити про ефективнішу організацію коду, меншу кількість зайвих чи вкладених викликів функцій.

Розподіл функцій:

Якщо певні функції домінують у неоптимізованому графіку, це може вказувати на потенційні "вузькі місця" у продуктивності, де оптимізація може принести найбільшу користь. В оптимізованому графіку бажано бачити більш рівномірний розподіл функцій, що свідчить про ефективне балансування навантаження та оптимізацію критичних ділянок коду.

Висновок аналізу Flame Graphs:

Аналізуючи обидва flame graph, можна зробити висновки про продуктивність і ефективність обох версій програми. Неоптимізована версія, імовірно, має неефективну структуру викликів, надмірне використання ресурсів або менш ефективні алгоритми, що призводить до довшого часу виконання та підвищення навантаження на систему. Навпаки, оптимізована версія показує ознаки кращої ефективності з меншою кількістю і глибиною викликів функцій, а також балансуванням навантаження, що свідчить про більш оптимальне використання ресурсів і кращу загальну продуктивність.

**Висновок**

Цей проект перетворився на всебічне дослідження системного аналізу та оптимізації. Я занурився в детальне вивчення різних аспектів системної архітектури, приділяючи особливу увагу аналізу продуктивності та ефективності нашого коду. Моя робота охоплювала не тільки поглиблене дослідження коду, але й використання різноманітних інструментів профілювання, що дозволили нам не тільки зрозуміти поточний стан системи, але й виявити ключові області для оптимізації.

Основна мета полягала у забезпеченні глибокого розуміння того, як можна покращити систему, щоб вона працювала найбільш ефективно. Це включало аналіз використання ресурсів, часу відповіді системи та оцінку можливостей масштабування. Я також зосередився на виявленні та усуненні "вузьких місць" продуктивності, які могли уповільнювати систему, та реалізації ефективних рішень, які могли б забезпечити оптимальне виконання програми.

Завдяки цьому дослідженню я не тільки значно покращив продуктивність системи, але й отримав цінні знання та досвід у галузі оптимізації та системного аналізу, що несумнівно знадобляться у майбутньому.