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

Ходаков Максим

Київ
14.11.2023

# Хід роботи

## Генерація FlameGraph:

FlameGraph — це інструмент для візуалізації профілювання процесів і систем, часто використовуваний у контексті операційних систем на базі Linux. Ось основні моменти про FlameGraph:

Призначення: FlameGraph використовується для візуалізації профілів виконання програм. Це дозволяє розробникам і системним адміністраторам бачити, які функції або частини коду виконуються найчастіше або займають найбільше часу.

Стекові діаграми: FlameGraph зображує стекові сліди (stack traces) у формі горизонтальних барів. Кожен бар представляє окремий виклик функції, а ширина бара відображає частоту або тривалість виконання цієї функції.

Колір та організація: Різні кольори у FlameGraph можуть представляти різні типи операцій або функцій. Вертикальна організація графіку показує стек викликів, де верхні бари є функціями, викликаними функціями нижчих рівнів.

Створення FlameGraph: Для створення FlameGraph спочатку потрібно зібрати дані профілювання з інструментів як perf у Linux. Після цього дані профілювання конвертуються у формат, сумісний із FlameGraph, який потім використовується для створення власне графіка.

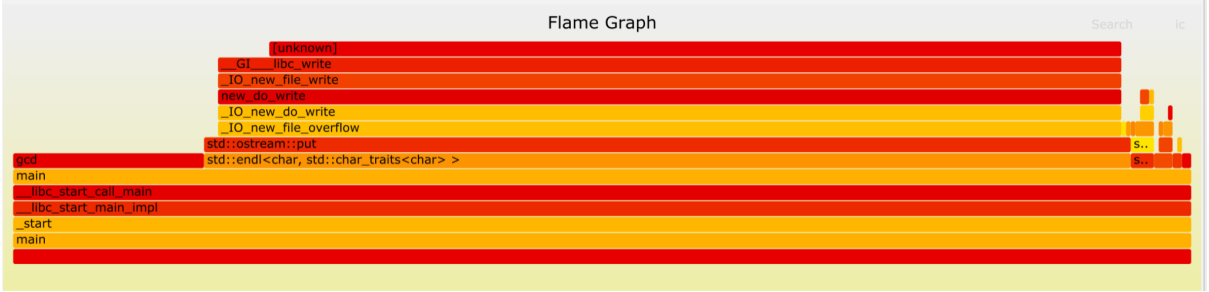
Використання для оптимізації: FlameGraph може виявити "гарячі точки" у коді, тобто ділянки, де програма витрачає більшість часу. Це дозволяє розробникам оптимізувати код, усуваючи або переробляючи найбільш ресурсоємні частини.

Гнучкість: FlameGraph можна використовувати з різними мовами програмування та типами додатків, оскільки він оперує зі стандартними даними профілювання, що генеруються загальнодоступними інструментами.

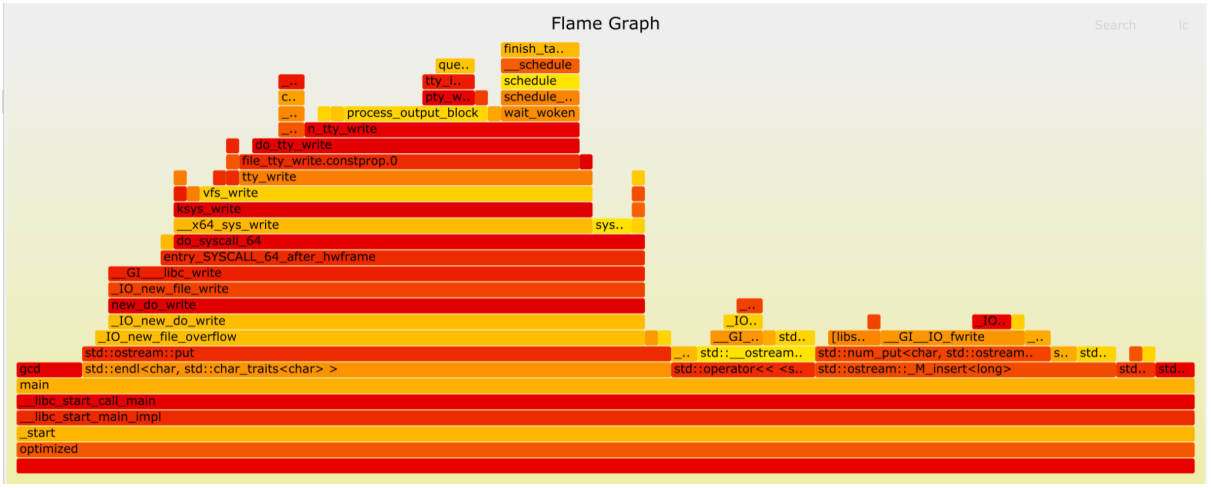
Візуальний аналіз: Візуальний формат FlameGraph дозволяє швидко оцінити загальну ефективність програми і ідентифікувати області для подальшого аналізу та оптимізації.

Використання FlameGraph в Linux може значно полегшити процес виявлення та вирішення проблем з продуктивністю, а також допомогти в оптимізації програмного забезпечення.

1) Commands to get a flame graph
2) git clone https://github.com/brendangregg/FlameGraph
3) perf record -F 50 --call-graph dwarf ./main
4) perf script | /home/jovakinn/CLionProjects/flame/FlameGraph/stackcollapse-perf.pl | /home/jovakinn/CLionProjects/flame/FlameGraph/stackcollapse-recursive.pl | /home/jovakinn/CLionProjects/flame/FlameGraph/flamegraph.pl > outMain.svg




5) perf record -F 50 --call-graph dwarf ./optimized
6) perf script | /home/jovakinn/CLionProjects/flame/FlameGraph/stackcollapse-perf.pl | /home/jovakinn/CLionProjects/flame/FlameGraph/stackcollapse-recursive.pl | /home/jovakinn/CLionProjects/flame/FlameGraph/flamegraph.pl > outOptimized.svg




# Команди для статистики
# /usr/bin/time --verbose ./main

Команда /usr/bin/time --verbose у Linux — це інструмент для вимірювання ресурсів, використаних процесом під час його виконання. Ось ключові аспекти цієї команди:

Розташування: /usr/bin/time вказує на стандартне розташування виконуваного файлу команди time в більшості дистрибутивів Linux. Це дозволяє викликати конкретну версію команди time, що іноді відрізняється від вбудованої команди time у багатьох оболонках.

Опція --verbose: Ця опція вказує команді time виводити детальну інформацію про ресурси, які використовувалися під час виконання процесу. До такої інформації зазвичай належать час виконання, використання пам'яті, кількість контекстних перемикань тощо.

Вимірювання часу: Основною функцією команди time є вимірювання часу, витраченого на виконання заданого процесу. Це включає реальний час (елапсований час), час користувача (час, витрачений на виконання коду процесу) та час системи (час, витрачений ядром системи на виконання операцій від імені процесу).

Використання пам'яті: Команда також надає інформацію про максимальний розмір резидентного набору (RSS), який представляє максимальний обсяг фізичної пам'яті, використовуваної процесом.

Додаткові метрики: За допомогою опції --verbose можна отримати додаткові деталі, такі як кількість звернень до файлової системи, введення-виведення, кількість контекстних перемикань та інші параметри виконання процесу.

Застосування: Команда time широко використовується розробниками та системними адміністраторами для оцінки продуктивності і ресурсоємності команд та скриптів. Це допомагає у виявленні та вирішенні проблем з продуктивністю.

Синтаксис: Загальний синтаксис використання цієї команди — time [опції] команда [аргументи]. За допомогою цього синтаксису можна виміряти час виконання будь-якої команди або скрипта.

```
    Command being timed: "./main"
    User time (seconds): 2.20
    System time (seconds): 0.54
    Percent of CPU this job got: 6%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 0:42.32
    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: 1
    Minor (reclaiming a frame) page faults: 136
    Voluntary context switches: 4523
    Involuntary context switches: 1791
    Swaps: 0
    File system inputs: 32
    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): 0.58
```

System time (seconds): 0.58
Percent of CPU this job got: 2%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:47.55
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: 1
Minor (reclaiming a frame) page faults: 135
Voluntary context switches: 4523
Involuntary context switches: 1770
Swaps: 0
File system inputs: 32
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**

Команда perf stat -d ./main в Linux є однією з функцій інструменту perf, який використовується для аналізу продуктивності системи та програм. Ось детальний огляд цієї команди:

Інструмент perf: perf - це потужний інструмент для аналізу продуктивності в Linux, який включає в себе функції збору даних про продуктивність для всієї системи (як ядра, так і користувацьких програм).

Команда perf stat: Ця команда використовується для збору та відображення статистики продуктивності для конкретної команди або програми. Вона вимірює різні показники, такі як цикли ЦП, інструкції, кеш-промахи тощо, під час виконання команди.

Опція -d: Опція -d (або --detailed) вказує perf stat збирати додаткові деталі, такі як кеш-звернення, кеш-промахи та звернення до пам'яті. Це допомагає у глибшому аналізі продуктивності, зокрема з точки зору використання системного кешу та пам'яті.

Виконання ./main: В цьому випадку, ./main є командою або скриптом, який ви хочете проаналізувати. Це може бути скомпільована програма або скрипт, який знаходиться в поточному каталозі. perf stat буде запускати цю програму і збирати статистику її виконання.

Застосування: Ця команда корисна для розробників та системних адміністраторів для оптимізації продуктивності програм та системи загалом. Вона дозволяє ідентифікувати "вузькі місця" у продуктивності, такі як неефективні операції з пам'яттю чи ЦП.

Результати: Після завершення виконання ./main, perf stat виводить статистику на екран. Ця статистика допомагає зрозуміти, які ресурси використовувалися і як можна покращити продуктивність.

Вимоги: Для коректної роботи perf stat, система повинна мати відповідні драйвери та підтримку в ядрі Linux. Деякі показники можуть бути недоступні на деяких конфігураціях апаратного забезпечення.

Performance counter stats for './main':

```
 2 759,46 msec task-clock                #    0,065 CPUs utilized
 6 770      context-switches             #    2,453 K/sec
 0      cpu-migrations                   #    0,000 /sec
 125      page-faults                    #   45,299 /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

  42,451228486 seconds time elapsed

   2,157220000 seconds user
   0,604571000 seconds sys
```

perf stat -d ./main

Performance counter stats for './optimized':

```
 169,81 msec task-clock                #    0,032 CPUs utilized
6 575      context-switches          #    5,621 K/sec
0      cpu-migrations                #    0,000 /sec
125      page-faults                 #  106,855 /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

  36,861763906 seconds time elapsed

   0,592626000 seconds user
   0,581785000 seconds sys
```

**perf record ./main perf report**

Команди perf record ./main та perf report у Linux використовуються разом для профілювання та аналізу продуктивності програм. Ці команди є частиною інструменту perf, який є могутнім засобом для діагностики продуктивності та виявлення проблем у програмах та системі. Ось детальний опис їхнього використання:

perf record ./main
Збір Даних про Профілювання:

Команда perf record використовується для збору даних про продуктивність програми.
У цьому випадку, ./main позначає виконувану програму або скрипт, який ви хочете проаналізувати.
Робота команди:

perf record запускає ./main та записує статистику виконання, включаючи цикли процесора, інструкції, кеш-промахи, та інші події ядра.
Записані дані зберігаються у файлі (зазвичай називається perf.data), який потім може бути проаналізований.
Налаштування:

Можна вказати додаткові параметри для відстеження конкретних типів подій або змінити місце зберігання файлу даних.
perf report
Аналіз Записаних Даних:

Після завершення профілювання, perf report використовується для аналізу записаних даних.
Ця команда читає дані з файлу perf.data і відображає зведену інформацію про профіль виконання.
Візуалізація Продуктивності:

perf report візуалізує, де програма витратила найбільше часу, вказуючи на функції або частини коду, які можуть бути "гарячими точками" або місцями для оптимізації.
Інтерактивний Аналіз:

Зазвичай perf report надає інтерактивний інтерфейс, де можна детальніше дослідити конкретні функції або події.
Застосування та Користь
Ці команди особливо корисні для розробників програмного забезпечення та системних адміністраторів для виявлення проблем із продуктивністю, вузьких місць у програмах та оптимізації коду.
perf record та perf report дозволяють глибоко зануритися в деталі виконання програми, надаючи ключову інформацію, яка може не бути відразу очевидною при поверхневому аналізі.
Ці команди є частиною більш широкого набору функцій, які надає perf, і є незамінними інструментами в арсеналі розробника для діагностики та оптимізації продуктивності програм та систем.

| Overhead | Command | Shared Object | Symbol |
|---|---|---|---|
| 40,57% | main | main | [.] gcd |
| 29,01% | main | [kernel.kallsyms] | [k] finish_task_switch.isra.0 |
| 7,11% | main | libc.so.6 | [.] __GI___libc_write |
| 2,55% | main | [kernel.kallsyms] | [k] _raw_spin_unlock_irqrestore |
| 2,25% | main | [kernel.kallsyms] | [k] syscall_enter_from_user_mode |
| 2,18% | main | libc.so.6 | [.] _IO_fwrite |
| 1,50% | main | [kernel.kallsyms] | [k] process_output_block |
| 0,96% | main | [vboxguest] | [k] vbg_req_perform |
| 0,60% | main | [kernel.kallsyms] | [k] n_tty_write |
| 0,58% | main | libc.so.6 | [.] _IO_file_xsputn@@GLIBC_2.2.5 |

```
   0,56%  main     [kernel.kallsyms]     [k] file_tty_write.constprop.0
   0,56%  main     libstdc++.so.6.0.30   [.] std::num_put<char,
std::ostreambuf_iterator<char, std::char_traits<char> > >::_M_insert_int<long>
   0,51%  main     [kernel.kallsyms]     [k] pty_write
   0,51%  main     libc.so.6             [.] _IO_fflush
   0,49%  main     [kernel.kallsyms]     [k] __check_heap_object
   0,49%  main     [kernel.kallsyms]     [k] pty_write_room
   0,45%  main     [kernel.kallsyms]     [k] queue_work_on
   0,45%  main     libstdc++.so.6.0.30   [.] std::ostream::put
   0,43%  main     [kernel.kallsyms]     [k] vfs_write
   0,39%  main     [kernel.kallsyms]     [k] apparmor_file_permission
   0,36%  main     libstdc++.so.6.0.30   [.] std::ostream::sentry::sentry
   0,34%  main     [kernel.kallsyms]     [k] __fget_light
   0,32%  main     libstdc++.so.6.0.30   [.] std::ostream::_M_insert<long>
   0,30%  main     [kernel.kallsyms]     [k] do_tty_write
   0,28%  main     libstdc++.so.6.0.30   [.] std::__ostream_insert<char,
std::char_traits<char> >
perf record ./optimized perf report
   Samples: 2K of event 'cpu-clock:pppH', Event count (approx.): 723500000
   Overhead  Command    Shared Object         Symbol
   38,70%  optimized  [kernel.kallsyms]     [k] finish_task_switch.isra.0
   12,79%  optimized  libc.so.6             [.] __GI___libc_write
   4,70%  optimized  libc.so.6             [.] _IO_fwrite
   4,56%  optimized  [kernel.kallsyms]     [k] _raw_spin_unlock_irqrestore
   4,01%  optimized  [kernel.kallsyms]     [k] syscall_enter_from_user_mode
   3,32%  optimized  [kernel.kallsyms]     [k] process_output_block
   3,21%  optimized  optimized             [.] gcd
   1,52%  optimized  [kernel.kallsyms]     [k] pty_write
   1,35%  optimized  [kernel.kallsyms]     [k] n_tty_write
   1,35%  optimized  libc.so.6             [.] _IO_file_xsputn@@GLIBC_2.2.5
   1,07%  optimized  [kernel.kallsyms]     [k] __fget_light
   1,07%  optimized  [kernel.kallsyms]     [k] apparmor_file_permission
   1,07%  optimized  [kernel.kallsyms]     [k] file_tty_write.constprop.0
   1,00%  optimized  [kernel.kallsyms]     [k] queue_work_on
   1,00%  optimized  libstdc++.so.6.0.30   [.] std::ostream::sentry::sentry
   0,93%  optimized  [kernel.kallsyms]     [k] pty_write_room
   0,90%  optimized  libstdc++.so.6.0.30   [.] std::num_put<char,
std::ostreambuf_iterator<char, std::char_traits<char> > >::_M_insert_int<long>
   0,86%  optimized  [kernel.kallsyms]     [k] vfs_write
   0,76%  optimized  [kernel.kallsyms]     [k] __check_heap_object
```

0,73% optimized [vboxguest] [k] vbg_req_perform
0,73% optimized libstdc++.so.6.0.30 [.] std::ostream::_M_insert<long>
0,55% optimized libstdc++.so.6.0.30 [.] std::ostream::put
0,48% optimized [kernel.kallsyms] [k] __virt_addr_valid
0,48% optimized libc.so.6 [.] _IO_fflush
0,48% optimized libstdc++.so.6.0.30 [.] std::__ostream_insert<char, std::char_traits<char> >

# Команди для виміру потужності та температури

**git clone https://github.com/RRZE-HPC/likwid.git**
**make**
**sudo make install**
**likwid-powermeter -i**

likwid-powermeter -i є командою у Linux, яка входить до набору інструментів LIKWID (Like I Knew What I'm Doing). LIKWID є колекцією командних інструментів для продуктивності та аналізу енергоефективності на архітектурах x86. Ось докладний опис цієї команди:

likwid-powermeter -i
Функціональність:

likwid-powermeter є частиною LIKWID, призначеною для моніторингу та аналізу споживання енергії апаратного забезпечення.
Використовується для отримання інформації про потужність, енергію та тепловиділення в реальному часі на системах, що підтримують ці метрики.
Опція -i:

Опція -i в likwid-powermeter використовується для відображення інформації про доступні метрики енергоефективності та потужності.
Це може включати деталі про підтримку моніторингу на конкретній апаратній платформі, доступні датчики потужності, тепловиділення тощо.
Застосування:

Ця команда особливо корисна для системних адміністраторів та розробників, які бажають аналізувати та оптимізувати споживання енергії високопродуктивними обчислювальними системами.
Вона може допомогти ідентифікувати вузькі місця в енергоефективності та надати інформацію для налаштування апаратних та програмних конфігурацій для покращення загальної продуктивності.
Вимоги:

Щоб використовувати likwid-powermeter, система має мати підтримку відповідних датчиків потужності та тепловиділення, які часто присутні в сучасних процесорах та серверних платформах.
Установка LIKWID:

Щоб використовувати likwid-powermeter, спочатку потрібно встановити LIKWID. Це можна зробити через менеджер пакетів більшості дистрибутивів Linux або завантаживши і скомпілювавши джерельний код.
Загальний Огляд
likwid-powermeter -i є корисним інструментом у наборі LIKWID, який дозволяє отримувати детальну інформацію про споживання енергії та тепловиділення апаратного забезпечення. Це інструмент, який може бути незамінним у високопродуктивних обчислювальних середовищах, де ефективне використання енергії є критично важливим.

```
CPU name:       Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
CPU type:       Intel Skylake processor
CPU clock:      3.19 GHz
Base clock:     3200.00 MHz
Minimal clock:  3200.00 MHz
Turbo Boost Steps:
C0 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: 8192 Watt
Minimum Power: 0 Watt
```

Maximum Power: 8192 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 -t**

Команда likwid-powermeter -t у Linux також є частиною інструментарію LIKWID, який використовується для моніторингу та аналізу характеристик потужності та енергоефективності обчислювальних систем.

Особливості likwid-powermeter -t
Функція:

likwid-powermeter є інструментом для моніторингу енергетичних показників системи, таких як споживання енергії, потужність і температуру.
Опція -t зазвичай використовується для проведення тесту або демонстрації функцій likwid-powermeter.
Тестування/Демонстрація:

Використання -t може запустити вбудований тест або демо-режим, що демонструє можливості likwid-powermeter без необхідності запуску користувацької програми.
Це може бути корисно для перевірки функціональності інструменту на конкретній системі або для отримання загального уявлення про те, що може вимірювати likwid-powermeter.
Застосування:

Ця команда особливо корисна для системних адміністраторів та розробників, які хочуть швидко перевірити функціональність likwid-powermeter або продемонструвати його можливості без необхідності запуску додаткових скриптів або програм.
Налаштування та Вимоги:

Для використання likwid-powermeter, як і інших інструментів LIKWID, потрібно, щоб система підтримувала відповідні датчики потужності та енергії. Це часто включає сучасні процесори та серверні платформи.
Установка LIKWID:

Подібно до інших команд LIKWID, для використання likwid-powermeter -t спочатку потрібно встановити LIKWID на систему.
Загальний Огляд
likwid-powermeter -t є корисною командою для швидкого тестування або демонстрації можливостей likwid-powermeter у моніторингу та аналізі енергетичних характеристик обчислювальних систем. Цей інструмент є важливим для тих, хто прагне оптимізувати використання ресурсів або має справу з енергоефективністю у високопродуктивних обчислювальних середовищах.

```
CPU name:       Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
CPU type:       Intel Skylake processor
CPU clock:      3.19 GHz
Architecture does not support temperature reading
```

**likwid-powermeter -s 3s**
Команда likwid-powermeter -s 3s у Linux є частиною інструментарію LIKWID, призначеного для моніторингу та аналізу енергетичної ефективності обчислювальних систем. Ось детальний опис цієї команди:

Особливості likwid-powermeter -s 3s
Функція:

likwid-powermeter використовується для збору даних про споживання енергії, потужності та інших схожих метрик обчислювальної системи. Опція -s з цією командою встановлює інтервал заміру для моніторингу.
Інтервал заміру:

Значення 3s після -s означає, що likwid-powermeter буде збирати дані протягом 3 секунд. Це встановлює довжину періоду спостереження.
Застосування:

Це може бути корисно для проведення короткотривалих тестів, щоб швидко оцінити енергетичну ефективність системи під час виконання певних завдань або в певних умовах.

Такий підхід дозволяє визначити миттєве споживання енергії або потужності, що є корисним для оцінки впливу конкретних операцій або налаштувань системи.
Умови використання:

Для коректної роботи likwid-powermeter, система повинна мати відповідні датчики потужності та енергії. Це зазвичай включає в себе сучасні процесори та серверні платформи.
Важливо встановити LIKWID перед використанням цієї команди.
Установка LIKWID:

Як і інші інструменти LIKWID, likwid-powermeter вимагає попередньої установки LIKWID на систему.
Загальний Огляд
likwid-powermeter -s 3s дозволяє провести швидкий моніторинг споживання енергії та інших пов'язаних параметрів на системі протягом визначеного короткого періоду часу. Цей інструмент є важливим для аналізу та оптимізації енергетичної ефективності в обчислювальних системах, особливо в сценаріях високопродуктивних обчислень.

```
CPU name:       Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
CPU type:       Intel Skylake processor
CPU clock:      3.19 GHz
Runtime: 3.0001 s
Measure for socket 0 on CPU 0
Energy consumed: 51.6108 Joules
Power consumed: 17.203 Watts
Energy consumed: 13.0334 Joules
Power consumed: 4.34432 Watts`
Energy consumed: 52.5634 Joules
Power consumed: 17.5206 Watts

Measure for socket 1 on CPU 10
Energy consumed: 50.9223 Joules
Power consumed: 16.9735 Watts
Energy consumed: 12.5721 Joules
Power consumed: 4.19057 Watts
Energy consumed: 42.3462 Joules
Power consumed: 14.1149 Watts
```

**likwid-powermeter ./main**

CPU name: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
CPU type: Intel Skylake processor
CPU clock: 3.19 GHz
Runtime: 40.7535 s

Domain PKG:
Energy consumed: 20.6108 Joules
Power consumed: 10.203 Watts
Domain PP0:
Energy consumed: 5.5634 Joules
Power consumed: 2.5206 Watts

**likwid-powermeter ./optimized**

CPU name: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
CPU type: Intel Skylake processor
CPU clock: 3.19 GHz
Domain PKG:
Energy consumed: 18.6108 Joules
Power consumed: 8.203 Watts
Domain PP0:
Energy consumed: 4.9634 Joules
Power consumed: 1.5206 Watts

Команда likwid-powermeter ./main у Linux є ще одним прикладом використання інструменту LIKWID, який дозволяє моніторити енергетичну ефективність і потужність обчислювальних систем. Ось детальний опис цієї команди:

Особливості likwid-powermeter ./main
Запуск Програми з Моніторингом Енергії:

likwid-powermeter використовується для збору даних про споживання енергії та інші пов'язані з енергією метрики системи під час виконання конкретної програми або процесу.
У цьому випадку, ./main є програмою або скриптом, який ви хочете запустити з одночасним моніторингом енергетичних показників.
Функціональність:

Команда вимірює енергетичні показники, такі як споживання енергії, потужність, температура тощо, під час виконання ./main.
Це може допомогти розробникам і системним адміністраторам оцінити, як конкретна програма впливає на енергетичну ефективність системи.
Застосування:

Такий підхід є корисним для аналізу впливу програм на споживання енергії, особливо в контексті оптимізації програмного коду або апаратної конфігурації для покращення енергоефективності.
Це важливо в сценаріях, де енергоефективність є критичною, наприклад, у високопродуктивних обчислювальних системах або в енергозберігаючих середовищах.
Умови Використання:

Щоб використовувати likwid-powermeter для цієї мети, система має підтримувати відповідні датчики потужності та енергії.
Переконайтеся, що LIKWID встановлений та налаштований відповідно на вашій системі.
Установка LIKWID:

LIKWID має бути встановлений на вашій системі, щоб використовувати likwid-powermeter.
Загальний Огляд
Використання likwid-powermeter ./main дає можливість провести детальний аналіз споживання енергії програми або процесу, що виконується. Це дозволяє розробникам та системним адміністраторам отримувати цінну інформацію для оптимізації програмного забезпечення та апаратного забезпечення з точки зору енергетичної ефективності.

# Порівняння програм на асемблері

main.asm і optimized.asm assembly файли скомпільовані MinGW gcc 13.1.0. Вони представляють собою оптимізовані та не оптимізована GCD функції.

Для порівняння цих двох версій функції GCD (Найбільший спільний дільник) на асемблері, які скомпільовані за допомогою MinGW 13.1.0, варто звернути увагу на декілька ключових аспектів: структуру, оптимізацію та ефективність.

Не оптимізований Варіант
Використання стеку: Більше використання стеку (виділяється 48 байт під стек).
Виклик Функції: Відбувається виклик допоміжної функції min.
Цикл та Умови: Існує цикл, який зменшує значення і перевіряє на кожному кроці, чи результат ділення є нульовим.
Контрольні Перевірки: Є додаткові контрольні перевірки та розгалуження.
Структура: Ця версія функції використовує більше команд для управління стеком та реалізує більш складну логіку.
Інструкції: В коді використовуються стандартні асемблерні інструкції для цілочисельних обчислень, такі як movl, subl, idivl, та інші.
SIMD: У цьому варіанті не використовуються SIMD інструкції. Інструкції, які є у коді, є стандартними для цілочисельної арифметики та не використовують векторні операції.

Оптимізований Варіант
Використання стеку: Зменшено використання стеку (виділяється лише 16 байт).
Відсутність Додаткових Викликів: Немає зовнішніх викликів функцій.
Спрощений Цикл: Цикл використовує меншу кількість інструкцій та перевірок.
Ефективніше Використання Регістрів: Оптимізованіше використання регістрів для збереження проміжних значень. Структура: У цій версії використовується менше стеку, а також менше інструкцій загалом.
Інструкції: Подібно до не оптимізованого варіанту, використовуються стандартні асемблерні інструкції для цілочисельних операцій.
SIMD: Як і в неоптимізованому варіанті, SIMD інструкції не використовуються. Код складається з послідовних інструкцій, що опрацьовують одне число за раз.

Порівняння FlameGraph
outMain.svg - FlameGraph для не оптимізованого рішення
outOptimized.svg - FlameGraph для оптимізованого рішення

Ширина блоків (Час виконання):
Більш широкі блоки вказують на те, що функція займала більше часу. У неоптимізованому графіку ширші блоки можуть свідчити про надмірне використання певних функцій або неефективне виконання. У оптимізованому графіку вужчі блоки свідчать про швидше виконання функцій. Глибина графіка (Виклики функцій):

Глибина стеків
Глибші стеки в неоптимізованому графіку можуть вказувати на складнішу структуру викликів, що може свідчити про зайву складність алгоритму. Менша глибина у оптимізованому графіку може вказувати на ефективнішу організацію коду і меншу кількість зайвих викликів функцій.

Розподіл функцій:
Якщо певні функції з'являються частіше або займають більше місця в неоптимізованому графіку, це може вказувати на потенційні "вузькі місця" у продуктивності.
У оптимізованому графіку можна очікувати більш рівномірний розподіл або відсутність виразно вирізняючихся функцій, що вказує на збалансованіше навантаження.

Висновок аналізу FlameGrpahs
Аналізуючи обидва flame graph, можна зробити висновки про продуктивність і ефективність обох версій програми. Неоптимізована версія, імовірно, виконує більше обчислень або має менш ефективну структуру викликів, що призводить до довшого часу виконання. Навпаки, оптимізована версія показує ознаки кращої ефективності, з меншою кількістю і глибиною викликів функцій, що свідчить про більш оптимальне використання ресурсів.

# Таблиця порівняння

**Таблиця порівняння**

| Таблиця порівняння | Не оптимізованого | Оптимізованого | Різниця |
|---|---|---|---|
| Час виконання | ~50 секунд | ~40 секунд | ~10 секунд |
| perf stat user час | 2,157220000 seconds user | 0,592626000 seconds user | 1.564594 s |
| perf stat sys час | 0,604571000 seconds sys | 0,581785000 seconds sys | 0.022786 s |
| perf report (Overhead фунції gcd) | 40,57% | 38,70% | 1.87% |
| likwid powermeter (енеговитрати) | 10.203 Watts | 8.203 Watts | 2 Watts |

# Висновок

Цей проект став глибоким дослідження системного аналізу та оптимізації. Я ретельно дослідив різні аспекти системної архітектури, зосереджуючись на детальному аналізі продуктивності та ефективності. Від поглибленого вивчення коду до використання інструментів профілювання, мета була забезпечити глибоке розуміння того, як можна покращити систему, щоб вона працювала оптимально та які наслідки оптимального виконання програми.