Київський національний університет імені Тараса Шевченка

Факультет комп'ютерних наук та кібернетики

Лабораторна робота №6

З дисципліни “Системне програмування”

на тему

“Performance Measurement and Analysis. (+Assembler and low-level optimisation)”

Виконав:

студент 3-го курсу групи ТТП-32

ОПП «Інформатика»

Черненко Євгеній

Київ – 2023

Лабораторна робота №6

Для обраної (скомпільованої) програми:

1) побудувати FlameGraph виконання:

1.1) продемонструвати процес побудови при захисті роботи (+3б.)

1.2) пояснити (інтерпретувати) отримані на графіку результати (+3б.)

(можна для самої програми або для системи в цілому)

(див.: https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html#Instructions)

2) зібрати та пояснити статистику її виконання, зокрема:

2.1) /usr/bin/time -- verbose <<prog>> (+2б.)

2.2) perf stat -d <<prog>> (+2б.)

2.3) perf report (after perf record) (+2б.)

(perf: створення і аналіз логів, траси виконання)

(див.: записи лекцій + https://www.brendangregg.com/perf.html#Examples)

3) заміряти енерговитрати (Power consumption):

3.1) системи при виконанні програми (+3б.)

3.2) виключно досліджуваної програми (+3б.)

(див.: https://luiscruz.github.io/2021/07/20/measuring-energy.html , але проявіть творчість!)

4) порівняти параметри виконання програми до та після оптимізації (або ключами -Ox, або внесенням змін у її вихідний код):

4.1) пояснити різницю в асемблерному коді до та після виконання оптимізації (+3б.)

4.1.1) пояснити на прикладі інструкцій AVX-* розширень (якщо застосовно), наприклад SIMD інструкції (+3б.)

(можна користуючись https://godbolt.org)

4.2) порівняти час та інші показники (див. п.п. 2.1, 2.2, тощо) виконання до і після оптимізації (+3б.)

4.3) продемонструвати зміни на FlameGraph (п. 1) після оптимізації (+3б.)

4.4) порівняти енерговитрати (п. 3) після оптимізації (+3б.)

(див.: записи лекцій, матеріали з попередніх пунктів, https://godbolt.org/)

*жирним виділено виконані етапи

Сумарна кількість балів виконаних етапів (або ж кількість балів на які я претендую) = 3 + 3 + 2 + 2 + 2 + 3 + 3 + 3 + 3 + 3 = 6 + 6 + 3 + 12 = 27

За основу для даної лабораторної роботи взято програму для обчислення чисел Трібоначі для n = 5:

```
#include <iostream>

// Функція для обчислення чисел Трібоначі через рекурсію
int tribonacci(int n) {
    if (n == 0 || n == 1) {
        return 0;
    } else if (n == 2) {
        return 1;
    } else {
        return tribonacci(n - 1) + tribonacci(n - 2) + tribonacci(n - 3);
    }
}

int main() {
    // Приклад виклику функції для n = 5
    int result = tribonacci(5);

    std::cout << "Tribonacci for n=5: " << result << std::endl;

    return 0;
}
```

Оптимізований варіант:

```
#include <iostream>
#include <unordered_map>

std::unordered_map<int, int> memo;  // Кеш для зберігання обчислених значень

// Функція для обчислення чисел Трібоначі через рекурсію з кешуванням
int tribonacci(int n) {
    if (n == 0 || n == 1) {
        return 0;
    } else if (n == 2) {
        return 1;
    } else {
        // Перевірка, чи значення для n вже є в кеші
        if (memo.find(n) != memo.end()) {
            return memo[n];
        } else {
            // Рекурсивне обчислення та збереження результату в кеші
            memo[n] = tribonacci(n - 1) + tribonacci(n - 2) + tribonacci(n - 3);
            return memo[n];
        }
    }
}
```

```
int main() {
    // Приклад виклику функції для n = 5
    int result = tribonacci(5);

    std::cout << "Tribonacci for n=5: " << result << std::endl;

    return 0;
}
```

Різниця початкового та оптимізованого варіанту полягає в тому, що в початковому варіанті в результаті рекурсії перевикликається функція із одними і тими ж значеннями, що займає зайвий час та використовує ресурси. Оптимізований варіант позбавлений цієї проблеми, бо результати функції кешуються в unordered_map, що значно прискорює роботу програми.

- **Flamegraph**

Для того, аби сформувати FlameGraph, треба до скомпільованої програми написати наступні команди в терміналі:

sudo perf record -F 99 -a -g /home/malighters/Documents/LW6/standard/tri

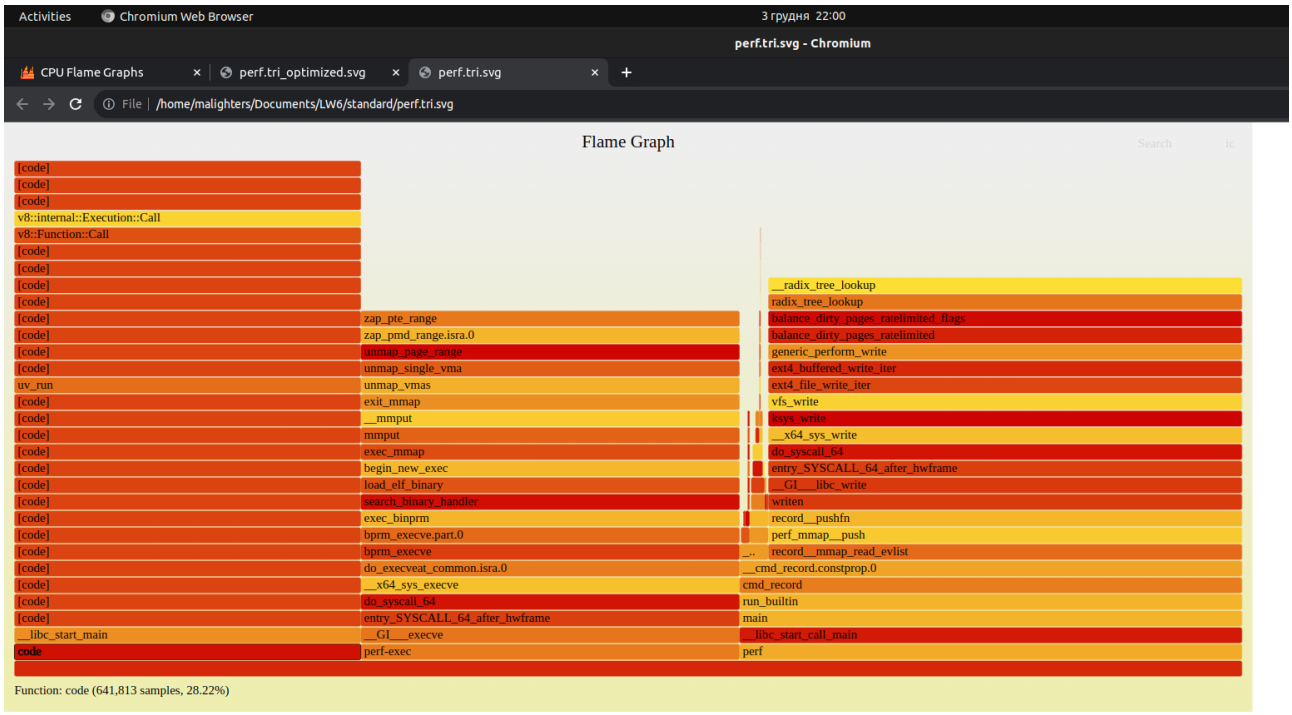
sudo perf script -f | ./stackcollapse-perf.pl > out.tri.perf-folded

./flamegraph.pl out.tri.perf-folded > perf.tri.svg

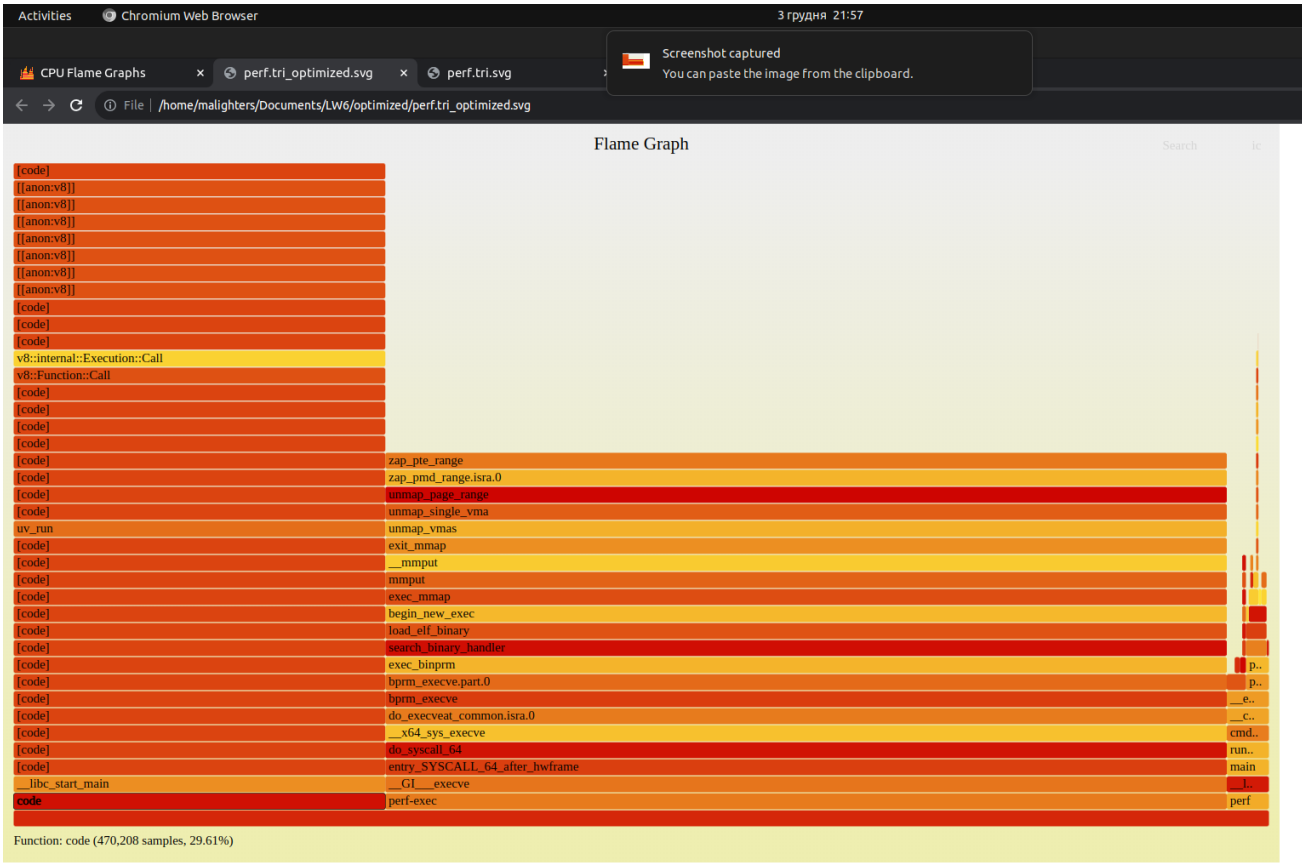
chromium perf.tri.svg # вже щоб запустити в браузері

Аналогічно треба і для tri_optimized

Flamegraph порівняння



Початкова програма (через рекурсію): 641 813 samples, або ж 28.22% програми



Оптимізована програма: 470 208 samples, або ж 29,61% програми

Бачимо, що оптимізована програма виконує менше операцій (470 208 проти 641 813).

Тому можна зробити висновок, що наша оптимізована програма має працювати швидше за початкову.

- **Статистика**

/usr/bin/time -- verbose ./tri

malighters@malighters:~/Documents/LW6/standard$ /usr/bin/time --verbose ./tri

Tribonacci for n=40: -1543615208

Command being timed: "./tri"

User time (seconds): 51.92

System time (seconds): 0.00

Percent of CPU this job got: 99%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:51.92

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: 137

Voluntary context switches: 1

Involuntary context switches: 67

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 ./tri_optimized

malighters@malighters:~/Documents/LW6/optimized$ /usr/bin/time --verbose ./tri_optimized

Tribonacci for n=40: -1543615208

Command being timed: "./tri_optimized"

User time (seconds): 0.00

System time (seconds): 0.00

Percent of CPU this job got: 100%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.00

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: 1

Involuntary context switches: 0

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

Оптимізована програма виконується настільки швидко, що /usr/bin/time навіть не встигає зафіксувати скільки часу воно виконується. Тому різниця очевидна. Приблизно 52 секунди проти <0.01 секунди.

perf stat -d ./tri

malighters@malighters:~/Documents/LW6/standard$ sudo perf stat -d ./tri

Tribonacci for n=40: -1543615208

Performance counter stats for './tri':

51 962,22 msec task-clock # 0,999 CPUs utilized

571 context-switches # 10,989 /sec

40 cpu-migrations # 0,770 /sec

125 page-faults # 2,406 /sec

175 615 688 723 cycles # 3,380 GHz (75,01%)

87 403 032 stalled-cycles-frontend # 0,05% frontend cycles idle (75,00%)

132 361 088 stalled-cycles-backend # 0,08% backend cycles idle (74,99%)

510 353 204 232 instructions # 2,91 insn per cycle

0,00 stalled cycles per insn (74,99%)

129 207 727 134 branches # 2,487 G/sec (75,00%)

223 911 200 branch-misses # 0,17% of all branches (75,01%)

128 596 817 456 L1-dcache-loads # 2,475 G/sec (75,00%)

4 411 976 L1-dcache-load-misses # 0,00% of all L1-dcache accesses (74,99%)

<not supported> LLC-loads

<not supported> LLC-load-misses

51,989267574 seconds time elapsed

51,953014000 seconds user

0,007998000 seconds sys

perf stat -d ./tri_optimized

malighters@malighters:~/Documents/LW6/optimized$ sudo perf stat -d ./tri_optimized

Tribonacci for n=40: -1543615208

Performance counter stats for './tri_optimized':

```
          1,65 msec task-clock                #    0,698 CPUs utilized
             0      context-switches          #    0,000 /sec
             0      cpu-migrations            #    0,000 /sec
           125      page-faults               #   75,963 K/sec
     5 274 250      cycles                    #    3,205 GHz
        14 729      stalled-cycles-frontend   #    0,28% frontend cycles idle
        21 906      stalled-cycles-backend    #    0,42% backend cycles idle
     4 653 632      instructions              #    0,88  insn per cycle
                                              #    0,00  stalled cycles per insn
       870 384      branches                  #  528,933 M/sec
        77 512      branch-misses             #    8,91% of all branches
   <not counted>      L1-dcache-loads                                               (0,00%)
   <not counted>      L1-dcache-load-misses                                         (0,00%)
 <not supported>      LLC-loads
 <not supported>      LLC-load-misses

     0,002358485 seconds time elapsed

     0,002453000 seconds user
     0,000000000 seconds sys
```

При perf report було відкрито наступне вікно в терміналі:

```
Samples: 66  of event 'cycles', Event count (approx.): 1587826
  Children      Self  Command         Shared Object       Symbol
+   67,03%    67,03%  perf-exec       [kernel.kallsyms]   [k] zap_pte_range
+   67,03%     0,00%  perf-exec       libc.so.6           [.] __GI___execve
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] entry_SYSCALL_64_after_hwframe
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] do_syscall_64
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] __x64_sys_execve
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] do_execveat_common.isra.0
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] bprm_execve
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] bprm_execve.part.0
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] exec_binprm
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] search_binary_handler
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] load_elf_binary
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] begin_new_exec
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] exec_mmap
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] mmput
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] __mmput
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] exit_mmap
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] unmap_vmas
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] unmap_single_vma
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] unmap_page_range
+   67,03%     0,00%  perf-exec       [kernel.kallsyms]   [k] zap_pmd_range.isra.0
+   29,61%    29,61%  code            code                [.] 0x0000000001d007cb
+   29,61%     0,00%  code            libc-2.31.so        [.] __libc_start_main
+   29,61%     0,00%  code            code                [.] 0x000055c258647c0d
+   29,61%     0,00%  code            code                [.] 0x000055c258903a65
+   29,61%     0,00%  code            code                [.] 0x000055c258903975
+   29,61%     0,00%  code            code                [.] 0x000055c258905f28
+   29,61%     0,00%  code            code                [.] 0x000055c258904d0d
+   29,61%     0,00%  code            code                [.] 0x000055c25b2d6ff4
+   29,61%     0,00%  code            code                [.] 0x000055c25b5a6131
+   29,61%     0,00%  code            code                [.] 0x000055c25b5e3c2c
+   29,61%     0,00%  code            code                [.] 0x000055c25b58213e
+   29,61%     0,00%  code            code                [.] 0x000055c25b5e38c5
+   29,61%     0,00%  code            code                [.] 0x000055c25b5e2b79
+   29,61%     0,00%  code            code                [.] 0x000055c25b5e3165
+   29,61%     0,00%  code            code                [.] 0x000055c25b5c4fbd
+   29,61%     0,00%  code            code                [.] 0x000055c2587ae091
+   29,61%     0,00%  code            code                [.] uv_run
+   29,61%     0,00%  code            code                [.] 0x000055c2586454b3
+   29,61%     0,00%  code            code                [.] 0x000055c258640457
+   29,61%     0,00%  code            code                [.] 0x000055c25f3f91f9
```

```
Samples: 66  of event 'cycles', Event count (approx.): 1587826
  Children      Self  Command         Shared Object       Symbol
+   29,61%     0,00%  code            code                [.] v8::Function::Call
+   29,61%     0,00%  code            code                [.] v8::internal::Execution::Call
+   29,61%     0,00%  code            code                [.] 0x000055c259964ad4
+   29,61%     0,00%  code            code                [.] 0x000055c2582d53c7
+   29,61%     0,00%  code            code                [.] 0x000055c2582d569c
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b833c918
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b835a049
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b85e186e
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b834bafd
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b866c417
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b866b9ff
+   29,61%     0,00%  code            [anon:v8]           [.] 0x000055c2b865c8aa
+   29,61%     0,00%  code            code                [.] 0x000055c25831f8a0
+    3,36%     0,00%  perf            libc.so.6           [.] __libc_start_call_main
+    3,36%     0,00%  perf            perf                [.] main
+    3,36%     0,00%  perf            perf                [.] run_builtin
+    3,36%     0,00%  perf            perf                [.] cmd_record
+    3,36%     0,00%  perf            perf                [.] __cmd_record.constprop.0
+    3,36%     0,00%  perf            perf                [.] __evlist__enable.constprop.0
+    1,94%     0,00%  perf            [kernel.kallsyms]   [k] entry_SYSCALL_64_after_hwframe
+    1,83%     0,00%  perf            perf                [.] perf_evsel__enable_cpu
+    1,83%     0,00%  perf            perf                [.] perf_evsel__run_ioctl
+    1,72%     0,00%  perf            [kernel.kallsyms]   [k] do_syscall_64
+    1,71%     0,72%  perf            [kernel.kallsyms]   [k] __x64_sys_ioctl
+    1,64%     0,22%  perf            libc.so.6           [.] __GI___ioctl
+    1,53%     0,00%  perf            perf                [.] evlist_cpu_iterator__next
     0,76%     0,34%  perf            [kernel.kallsyms]   [k] perf_ioctl
     0,65%     0,23%  perf            perf                [.] perf_cpu_map__idx
     0,49%     0,00%  perf            libc.so.6           [.] sched_setaffinity@@GLIBC_2.3.4
     0,46%     0,24%  perf            [kernel.kallsyms]   [k] __fget_light
     0,42%     0,00%  perf            [kernel.kallsyms]   [k] syscall_exit_to_user_mode
     0,42%     0,00%  perf            [kernel.kallsyms]   [k] exit_to_user_mode_prepare
     0,38%     0,19%  perf            [kernel.kallsyms]   [k] _perf_event_enable
     0,32%     0,13%  perf            [kernel.kallsyms]   [k] event_function
     0,30%     0,00%  perf            [kernel.kallsyms]   [k] __x64_sys_sched_setaffinity
     0,30%     0,00%  perf            [kernel.kallsyms]   [k] alloc_cpumask_var_node
     0,30%     0,00%  perf            [kernel.kallsyms]   [k] __kmalloc_node
     0,30%     0,00%  perf            [kernel.kallsyms]   [k] __kmem_cache_alloc_node
     0,29%     0,29%  perf            [kernel.kallsyms]   [k] mutex_lock
     0,23%     0,23%  perf            [kernel.kallsyms]   [k] entry_SYSRETQ_unsafe_stack
```

```
+   40,93%     0,00%  perf     libc.so.6          [.] __libc_start_call_main
+   40,93%     0,00%  perf     perf               [.] main
+   40,93%     0,00%  perf     perf               [.] run_builtin
+   40,93%     0,00%  perf     perf               [.] cmd_record
+   40,93%     0,00%  perf     perf               [.] __cmd_record.constprop.0
+   39,84%     0,00%  perf     [kernel.kallsyms]  [k] entry_SYSCALL_64_after_hwframe
+   39,56%     0,00%  perf     [kernel.kallsyms]  [k] do_syscall_64
+   38,58%    38,58%  perf     [kernel.kallsyms]  [k] __raw_spin_lock_irqsave
+   38,58%     0,00%  perf     perf               [.] record__mmap_read_evlist
+   38,58%     0,00%  perf     perf               [.] perf_mmap__push
+   38,58%     0,00%  perf     perf               [.] record__pushfn
+   38,58%     0,00%  perf     perf               [.] writen
+   38,58%     0,00%  perf     libc.so.6          [.] __GI___libc_write
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] __x64_sys_write
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] ksys_write
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] vfs_write
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] ext4_file_write_iter
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] ext4_buffered_write_iter
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] generic_perform_write
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] balance_dirty_pages_ratelimited
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] balance_dirty_pages_ratelimited_flags
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] radix_tree_lookup
+   38,58%     0,00%  perf     [kernel.kallsyms]  [k] __radix_tree_lookup
+   30,84%    30,84%  perf-ex  [kernel.kallsyms]  [k] __mod_lruvec_page_state
+   30,84%     0,00%  perf-ex  libc.so.6          [.] __GI___execve
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] entry_SYSCALL_64_after_hwframe
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] do_syscall_64
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] __x64_sys_execve
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] do_execveat_common.isra.0
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] bprm_execve
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] bprm_execve.part.0
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] exec_binprm
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] search_binary_handler
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] load_elf_binary
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] begin_new_exec
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] exec_mmap
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] mmput
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] __mmput
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] exit_mmap
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] unmap_vmas
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] unmap_single_vma
+   30,84%     0,00%  perf-ex  [kernel.kallsyms]  [k] unmap_page_range
```

```
Samples: 67  of event 'cycles', Event count (approx.): 2274017
  Children      Self  Command  Shared Object      Symbol
+   28,22%     0,00%  code     code               [.] 0x000055c25f3f870d
+   28,22%     0,00%  code     code               [.] 0x000055c25f3f37fe
+   28,22%     0,00%  code     code               [.] 0x000055c25f3f1837
+   28,22%     0,00%  code     code               [.] 0x000055c25f25c1e5
+   28,22%     0,00%  code     code               [.] 0x000055c25f2468b9
+   28,22%     0,00%  code     code               [.] v8::Function::Call
+   28,22%     0,00%  code     code               [.] v8::internal::Execution::Call
+   28,22%     0,00%  code     code               [.] 0x000055c259964ad4
+   28,22%     0,00%  code     code               [.] 0x000055c2582d53c7
+   28,22%     0,00%  code     code               [.] 0x000055c2582ccb47
+    2,35%     0,15%  perf     perf               [.] __evlist__enable.constprop.0
+    1,53%     0,00%  perf     perf               [.] perf_evsel__enable_cpu
+    1,53%     0,16%  perf     perf               [.] perf_evsel__run_ioctl
+    1,10%     0,00%  perf     libc.so.6          [.] __GI___ioctl
+    0,97%     0,15%  perf     [kernel.kallsyms]  [k] __x64_sys_ioctl
+    0,84%     0,15%  perf     perf               [.] evlist_cpu_iterator__next
     0,52%     0,24%  perf     [kernel.kallsyms]  [k] perf_ioctl
     0,36%     0,20%  perf     [kernel.kallsyms]  [k] perf_event_ctx_lock_nested.constprop.0
     0,35%     0,00%  perf     libc.so.6          [.] sched_setaffinity@@GLIBC_2.3.4
     0,31%     0,20%  perf     [kernel.kallsyms]  [k] _perf_event_enable
     0,31%     0,00%  perf     [kernel.kallsyms]  [k] __fdget
     0,31%     0,00%  perf     [kernel.kallsyms]  [k] __fget_light
     0,23%     0,12%  perf     [kernel.kallsyms]  [k] smp_call_function_single
     0,16%     0,16%  perf     [kernel.kallsyms]  [k] mutex_unlock
     0,16%     0,00%  perf     perf               [.] xyarray__max_y
     0,16%     0,16%  perf     [kernel.kallsyms]  [k] entry_SYSCALL_64
     0,16%     0,16%  perf     [kernel.kallsyms]  [k] entry_SYSRETQ_unsafe_stack
     0,16%     0,00%  perf     [kernel.kallsyms]  [k] syscall_enter_from_user_mode
     0,15%     0,00%  perf     [kernel.kallsyms]  [k] __x64_sys_sched_setaffinity
     0,15%     0,00%  perf     [kernel.kallsyms]  [k] alloc_cpumask_var_node
     0,15%     0,00%  perf     [kernel.kallsyms]  [k] __kmalloc_node
     0,15%     0,00%  perf     [kernel.kallsyms]  [k] __kmem_cache_alloc_node
     0,15%     0,12%  perf     [kernel.kallsyms]  [k] __perf_event_enable
     0,15%     0,15%  perf     [kernel.kallsyms]  [k] syscall_exit_to_user_mode
     0,15%     0,00%  perf     perf               [.] perf_evsel__ioctl
     0,15%     0,00%  perf     perf               [.] xyarray__entry
     0,15%     0,00%  perf     perf               [.] perf_cpu_map__idx
     0,13%     0,13%  perf     [kernel.kallsyms]  [k] srso_alias_return_thunk
     0,12%     0,00%  perf     [kernel.kallsyms]  [k] _perf_ioctl
     0,12%     0,00%  perf     [kernel.kallsyms]  [k] perf_event_for_each_child
     0,12%     0,00%  perf     [kernel.kallsyms]  [k] event_function_call
     0,12%     0,00%  perf     [kernel.kallsyms]  [k] generic_exec_single
```

- **Енерговитрати**

Для даного експерименту було використано n=30

malighters@malighters:~/Documents/LW6/standard$ likwid-powermeter ./tri

---

CPU name:	AMD Ryzen 7 7730U with Radeon Graphics

CPU type:	AMD K19 (Zen3) architecture

CPU clock:	2.00 GHz

Tribonacci for n=40: -1543615208

Runtime: 60.1021 s

Measure for socket 0 on CPU 0

Domain CORE:

Energy consumed: 972.892 Joules

Power consumed: 16.1873 Watt

Domain PKG:

Energy consumed: 719.19 Joules

Power consumed: 11.9661 Watt

malighters@malighters:~/Documents/LW6/optimized$ likwid-powermeter ./tri_optimized

---

CPU name: AMD Ryzen 7 7730U with Radeon Graphics

CPU type: AMD K19 (Zen3) architecture

CPU clock: 2.00 GHz

Tribonacci for n=40: -1543615208

Runtime: 1.03938 s

Measure for socket 0 on CPU 0

Domain CORE:

Energy consumed: 15.3075 Joules

Power consumed: 14.7275 Watt

Domain PKG:

Energy consumed: 12.4253 Joules

Power consumed: 11.9545 Watt

- **Різниця в асемблерному коді**

Скомпільовані коди програм були збережені в файли tri.asm та tri_optimized.asm.

- рекурсія та ітерація: `tri` використовує рекурсивні виклики, що легко ідентифікувати за `CALL tribonacci(int)`, тоді як `tri_optimized` використовує unordered_map для кешування значень, що ефективніше з точки зору використання пам'яті та швидкодії.
- управління пам'яттю: В `tri` використовується стек для зберігання стану кожного рекурсивного виклику, у той час як `tri_optimized` використовує динамічну пам’ять для збереження проміжних результатів.
- оптимізація продуктивності: tri_optimized оптимізує обчислення, зменшуючи загальну кількість операцій, порівняно з tri, де кожен рекурсивний виклик може призвести до значного зростання кількості обчислень, особливо для великих вхідних значень.

- **Зведена таблиця результатів (зібраної статистики)**

Характеристика	До оптимізації	Після оптимізації	Різниця	Різниця (%)
FlameGraph	641 813 samples (28.22%)	470 208 samples (29,61%)	**171605**	**26,7**
Час виконання	51,989267574 seconds time elapsed	0,002358485 seconds time elapsed	**51,986909089**	**99,9**
	51,953014000 seconds user	0,002453000 seconds user	**51,950561**	**99.9**
	0,007998000 seconds sys	0,000000000 seconds sys	**0,007998000**	**100**
Енерговитрати	972.892 Joules	15.3075 Joules	**957,5845**	**98**
(Domain Core)	16.1873 Watt	14.7275 Watt	**1,4598**	**9**