Benchmarking Lua's Garbage Collection (GC)

1. Analyzing Callgrind Outputs

Full GC Analysis:

- Function Calls and Their Percentages:
  - lua_gc: 11 calls (0.00%)
  - luaC_fullgc: 1 call (0.00%)
  - singlestep: 882,571 calls (1.89%)
  - propagatemark: 869,969 calls (6.58%)
  - reallymarkobject: 873,427 calls (2.01%)
  - luaH_realasize: 3,086,860 calls (1.32%)
- Total Percentage of Instructions Consumed: **11.80%** (approx.)

Generational GC Analysis:

- Function Calls and Their Percentages:
  - luaC_step: 25 calls (0.00%)
  - singlestep: 10,831 calls (0.02%)
  - propagatemark: 788,864 calls (6.01%)
  - reallymarkobject: 792,328 calls (1.84%)
  - luaH_realasize: 3,005,766 calls (1.29%)
  - freeobj: 601,276 calls (0.78%)
- Total Percentage of Instructions Consumed: **9.94%** (approx.)

Incremental GC Analysis:

- Function Calls and Their Percentages:
  - lua_gc: 20 calls (0.00%)
  - luaC_fullgc: 2 calls (0.00%)
  - singlestep: 1,401,500 calls (1.47%)
  - propagatemark: 1,379,142 calls (5.13%)
  - reallymarkobject: 1,384,030 calls (1.57%)
  - luaH_realasize: 5,812,969 calls (1.22%)
  - freeobj: 1,202,278 calls (0.76%)
- Total Percentage of Instructions Consumed: **10.15%** (approx.)

Conclusion:

- Full GC performs a comprehensive sweep of memory less frequently, leading to a higher percentage of instructions consumed per cycle. This deep clean can be more resource-intensive.
- Incremental GC breaks down the collection process into smaller steps that occur more frequently, distributing the GC workload throughout the program’s execution and leading to a moderate total instruction consumption.
- Generational GC focuses on younger objects that are more likely to become unreachable soon, optimizing the GC process by reducing overhead and minimizing the number of instructions consumed.

---

2. Analyzing the Effect of Changing Parameters

Here, we analyze the impact of different matrix sizes (m) and the number of operations (n) on GC performance.

Matrix Size (m)	Operations (n)	Full GC (%)	Generational GC (%)	Incremental GC (%)
100	100	14.02	10.03	12.85
500	100	14.22	10.05	13.78
5000	100	10.93	9.47	10.39

Explanation:

- **Small Matrix Size (m = 100, n = 100):**
  - The garbage generated is low, so the overhead of GC operations is more apparent. Full GC, with its comprehensive memory sweep, has the highest overhead, followed by Incremental GC and then Generational GC.
- **Medium Matrix Size (m = 500, n = 100):**
  - The performance of all three GCs is relatively comparable, indicating balanced GC activities and overhead across the board.
- **Large Matrix Size (m = 5000, n = 100):**

    - More garbage is generated, and Full GC consumes more resources for its deep clean. Incremental GC shows high instruction consumption due to more frequent collection cycles. Generational GC remains the most efficient, focusing on younger generations and minimizing overhead.

---

3. PERF Analysis

Analyzed several performance metrics using the PERF tool, which provides detailed insights into hardware performance.

Metric	No GC	Full GC	Incremental GC	Generational GC
Branch Misses	**294,072**	**293,497**	**519,832**	**285,181**
Page Faults	**11,132**	**11,131**	**11,638**	**10,089**
Cache Misses	**6,474,070**	**7,949,986**	**12,968,838**	**8,384,380**
IPC (Instructions per Cycle)	**3.36**	**3.11**	**3.48**	**2.96**

Note that the above results have been obtained by taking weighted average of cpu core and cpu atom processors as shown in the pictures in the end.

Explanation:

- **Branch Misses:**
    - Branch misses are relatively consistent across all GC types because the underlying marking and sweeping algorithms are similar. Incremental GC, however, shows higher branch misses due to more frequent execution, while Generational GC has the least, thanks to its focused collection strategy.
- **Page Faults:**
    - The number of page faults remains similar across all GCs, indicating that GC type does not significantly impact memory paging. Generational GC shows slightly fewer page faults, potentially due to better management of memory in young generations.

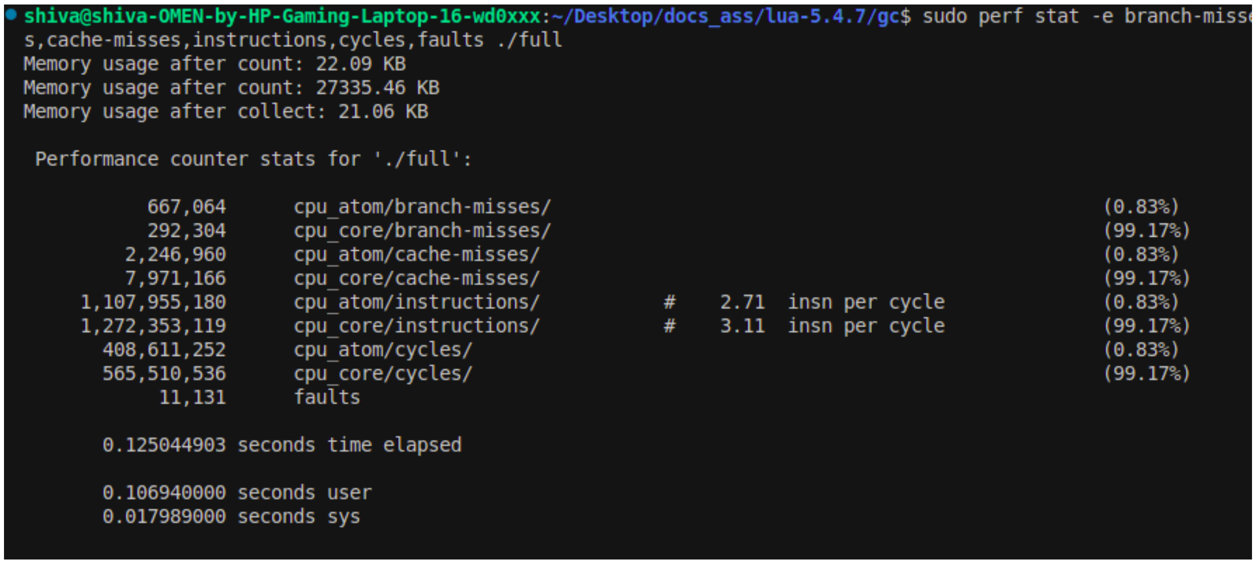
- **Cache Misses:**
    - No GC results in fewer cache misses, as there’s no interference from GC operations. Incremental GC has the highest cache misses, likely due to frequent pauses and resumed executions disrupting the cache. Generational GC, with targeted collection, maintains moderate cache misses.
- **IPC (Instructions per Cycle):**
    - Incremental GC shows the highest IPC because of the higher number of instructions executed, coupled with routine work. Generational GC shows a lower IPC due to its efficient memory collection process, which focuses on minimizing work done per cycle.

Final Observations

- Full GC is thorough but incurs higher overhead due to less frequent but comprehensive sweeps.
- Incremental GC balances GC workload across the program’s execution timeline, leading to frequent GC operations but a steady state of memory consumption.
- Generational GC optimizes GC by focusing on the young generation, reducing overhead and improving cache performance, making it the most efficient for programs with high object turnover.

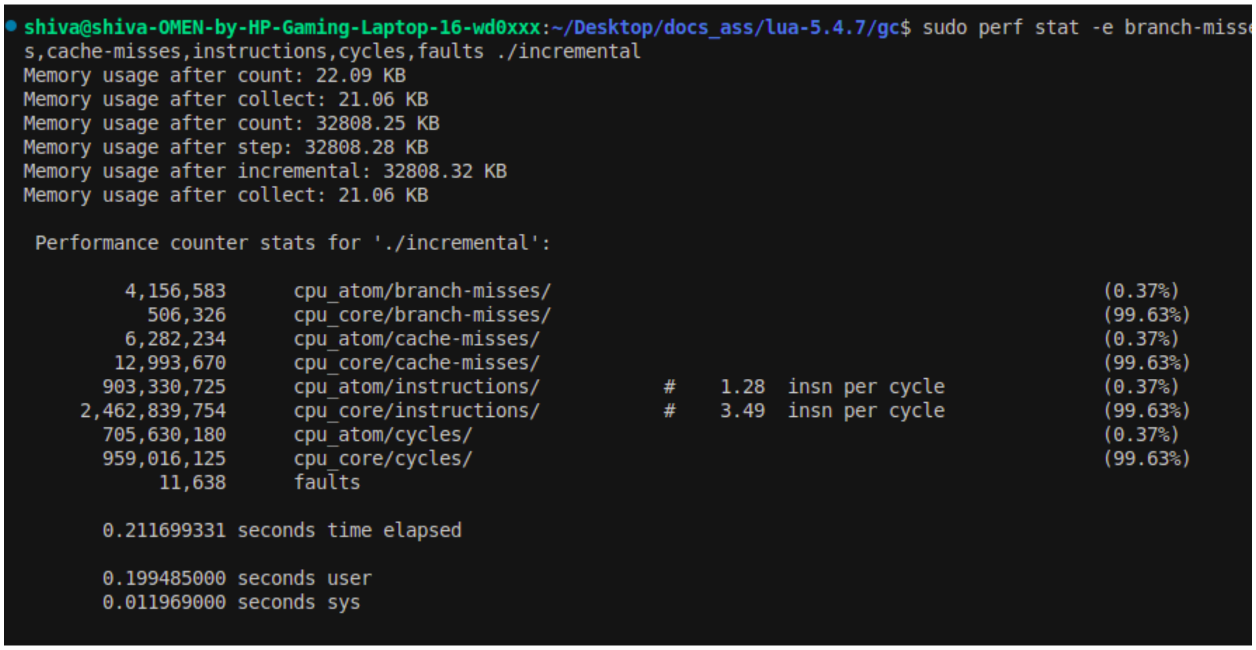
PERF Analysis

1. **Stats for full gc**



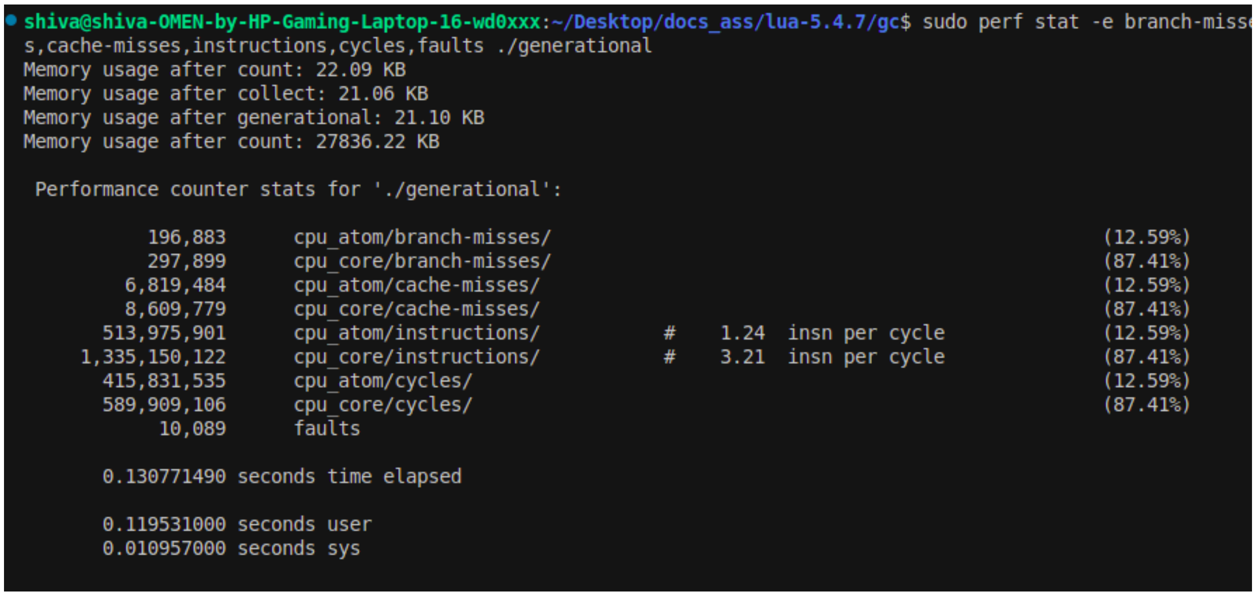
```
shiva@shiva-OMEN-by-HP-Gaming-Laptop-16-wd0xxx:~/Desktop/docs_ass/lua-5.4.7/gc$ sudo perf stat -e branch-misse
s,cache-misses,instructions,cycles,faults ./full
Memory usage after count: 22.09 KB
Memory usage after count: 27335.46 KB
Memory usage after collect: 21.06 KB

 Performance counter stats for './full':

           667,064      cpu_atom/branch-misses/                                                 (0.83%)
           292,304      cpu_core/branch-misses/                                                 (99.17%)
         2,246,960      cpu_atom/cache-misses/                                                  (0.83%)
         7,971,166      cpu_core/cache-misses/                                                  (99.17%)
     1,107,955,180      cpu_atom/instructions/           #    2.71  insn per cycle              (0.83%)
     1,272,353,119      cpu_core/instructions/           #    3.11  insn per cycle              (99.17%)
       408,611,252      cpu_atom/cycles/                                                        (0.83%)
       565,510,536      cpu_core/cycles/                                                        (99.17%)
            11,131      faults

       0.125044903 seconds time elapsed

       0.106940000 seconds user
       0.017989000 seconds sys
```

2. **Stats for incremental gc**



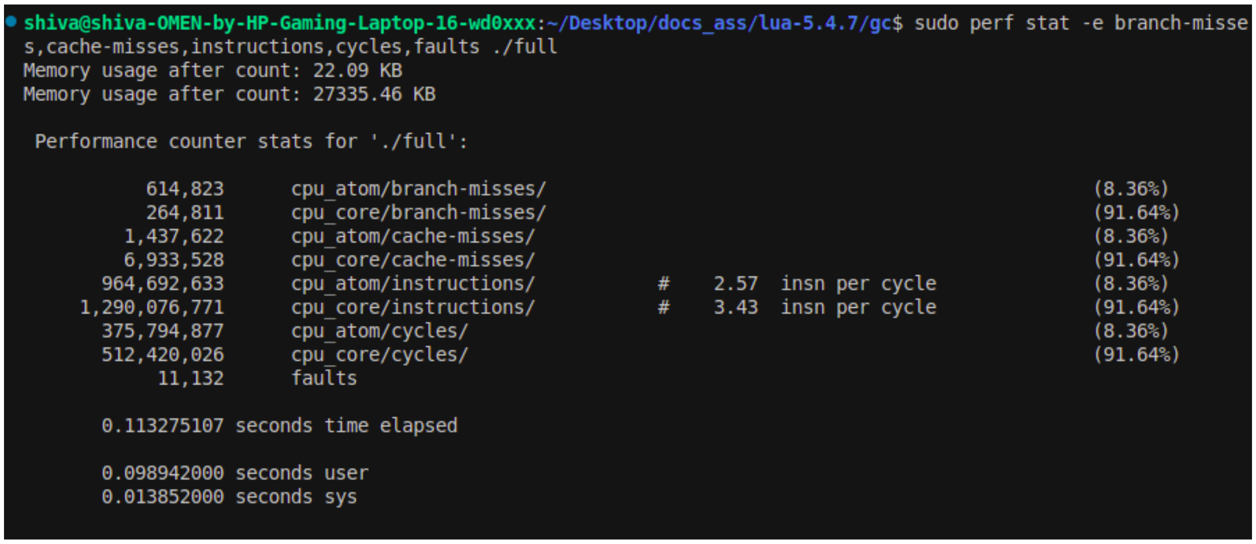
```
shiva@shiva-OMEN-by-HP-Gaming-Laptop-16-wd0xxx:~/Desktop/docs_ass/lua-5.4.7/gc$ sudo perf stat -e branch-misse
s,cache-misses,instructions,cycles,faults ./incremental
Memory usage after count: 22.09 KB
Memory usage after collect: 21.06 KB
Memory usage after count: 32808.25 KB
Memory usage after step: 32808.28 KB
Memory usage after incremental: 32808.32 KB
Memory usage after collect: 21.06 KB

 Performance counter stats for './incremental':

         4,156,583      cpu_atom/branch-misses/                                                 (0.37%)
           506,326      cpu_core/branch-misses/                                                 (99.63%)
         6,282,234      cpu_atom/cache-misses/                                                  (0.37%)
        12,993,670      cpu_core/cache-misses/                                                  (99.63%)
       903,330,725      cpu_atom/instructions/           #    1.28  insn per cycle              (0.37%)
     2,462,839,754      cpu_core/instructions/           #    3.49  insn per cycle              (99.63%)
       705,630,180      cpu_atom/cycles/                                                        (0.37%)
       959,016,125      cpu_core/cycles/                                                        (99.63%)
            11,638      faults

       0.211699331 seconds time elapsed

       0.199485000 seconds user
       0.011969000 seconds sys
```

3. **Stats for generational gc**



```
shiva@shiva-OMEN-by-HP-Gaming-Laptop-16-wd0xxx:~/Desktop/docs_ass/lua-5.4.7/gc$ sudo perf stat -e branch-misse
s,cache-misses,instructions,cycles,faults ./generational
Memory usage after count: 22.09 KB
Memory usage after collect: 21.06 KB
Memory usage after generational: 21.10 KB
Memory usage after count: 27836.22 KB

 Performance counter stats for './generational':

           196,883      cpu_atom/branch-misses/                                                 (12.59%)
           297,899      cpu_core/branch-misses/                                                 (87.41%)
         6,819,484      cpu_atom/cache-misses/                                                  (12.59%)
         8,609,779      cpu_core/cache-misses/                                                  (87.41%)
       513,975,901      cpu_atom/instructions/           #    1.24  insn per cycle              (12.59%)
     1,335,150,122      cpu_core/instructions/           #    3.21  insn per cycle              (87.41%)
       415,831,535      cpu_atom/cycles/                                                        (12.59%)
       589,909,106      cpu_core/cycles/                                                        (87.41%)
            10,089      faults

       0.130771490 seconds time elapsed

       0.119531000 seconds user
       0.010957000 seconds sys
```

4. **Stats for No gc configuration**



```
shiva@shiva-OMEN-by-HP-Gaming-Laptop-16-wd0xxx:~/Desktop/docs_ass/lua-5.4.7/gc$ sudo perf stat -e branch-misse
s,cache-misses,instructions,cycles,faults ./full
Memory usage after count: 22.09 KB
Memory usage after count: 27335.46 KB

 Performance counter stats for './full':

           614,823      cpu_atom/branch-misses/                                                 (8.36%)
           264,811      cpu_core/branch-misses/                                                 (91.64%)
         1,437,622      cpu_atom/cache-misses/                                                  (8.36%)
         6,933,528      cpu_core/cache-misses/                                                  (91.64%)
       964,692,633      cpu_atom/instructions/           #    2.57  insn per cycle              (8.36%)
     1,290,076,771      cpu_core/instructions/           #    3.43  insn per cycle              (91.64%)
       375,794,877      cpu_atom/cycles/                                                        (8.36%)
       512,420,026      cpu_core/cycles/                                                        (91.64%)
            11,132      faults

       0.113275107 seconds time elapsed

       0.098942000 seconds user
       0.013852000 seconds sys
```