

Project 2 - DB

Teammates

Minghan Zhu (mz2716)

Zeyu Liu (zl3087)

Tushar Arora (ta2673)

1. The machine characteristics for the for the google cloud are -

```
root@kali:~/# lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 46 bits physical, 48 bits virtual
CPU(s): 1
On-line CPU(s) list: 0
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 85
Model name: Intel(R) Xeon(R) CPU @ 2.00GHz
Stepping: 3
CPU MHz: 2000.186
bogomips: 4000.37
Hyperthreading: KVM
Virtualization type: full
L1d cache: 32 KiB
L1i cache: 32 KiB
L2 cache: 1 MiB
L3 cache: 38.5 MiB
NUMA node0 CPU(s): 0
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf: Mitigation; PTE Inversion
Vulnerability Mds: Mitigation; Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation; Retpolines, IBPB conditional, IBRS_FW, STIBP disabled, RSB filling
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Mitigation; Clear CPU buffers; SMT Host state unknown
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpebgl rdtscp lm constant tsc rep_good nopl xtopol
og noplstop tsc cpuid tsc_known_freq pni pclmulqdq sse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch
h_invdpc single_pti e8b1_1bpb stibp fsgsbase tsc_adjust bmi1 hle avx2 smap bmi2 erms invpcid rtm mpx avx512f avx512dq rdseed adx smap clflushopt clwb avx512cd
```

Figure 1: `lscpu` command result

- a. GCP VM type: n1-standard-n
- b. CPU(s): 1 Physical CPU Core
- c. Memory (GB): 3.75
- d. L1d cache: 32 KiB
- e. L1i cache: 32 KiB
- f. L2 cache: 1 MiB

Chart of the five functions

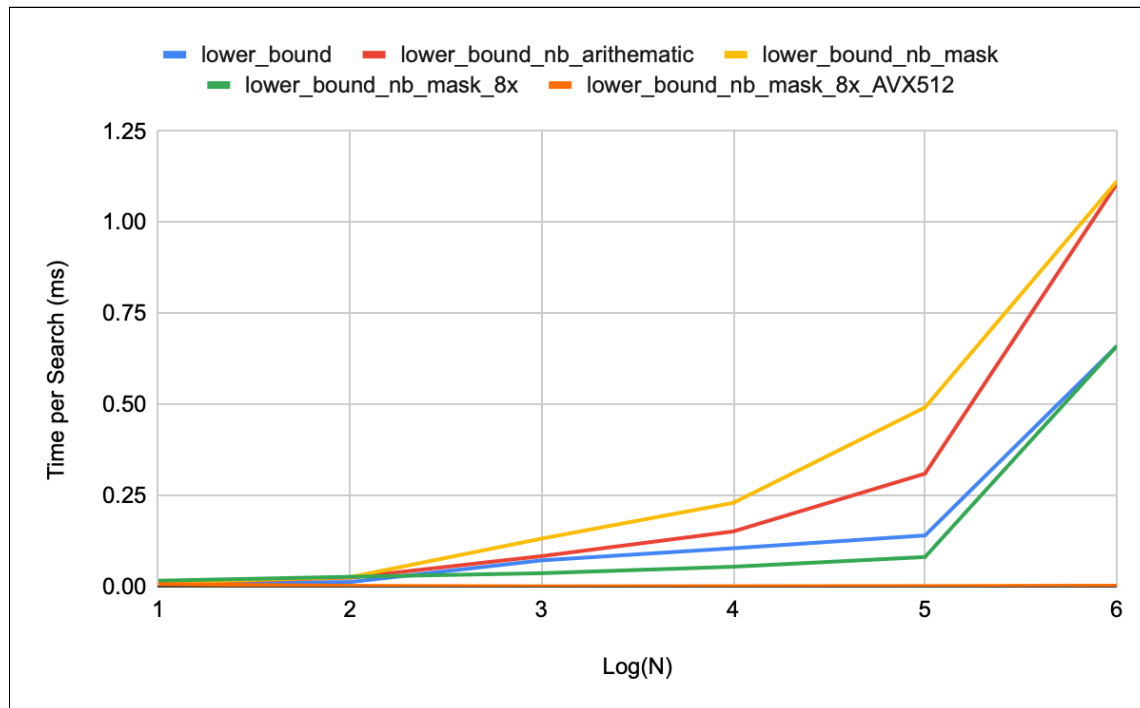


Figure 2: Measured performance of the five routines as N varied from 10 to 10^6

Executing command - ./db4112 N 1 1 1 1000

R = 1000

where the value of “N” was varied from 10 to 1000000 (10^6)

From the graph, it is very evident that the method, lower_bound_nb_mask_8x_AVX512 performs exceptionally well when the value of N starts increasing. The performance metrics of all the methods is almost the same for small numbers, between 10 and 100 but after N=100 the performance metrics of the lower_bound_nb_mask decreases exponentially. This method takes the most time to complete the scanning and this was also very evident while we were running this method for analyzing the performance. In some of the cases, it took over 30 mins to finish the scan.

But a general analysis of the graph signifies that the 8x methods have a comparatively better performance than all the other methods. The lower_bound_nb_mask_8x method outperforms all

the other methods (except AVX512) for all values above $N=100$. One of the highlighted behaviors in the above graph is that after $N=100000$, all the methods (except AVX512) get an exponential growth in the time taken to run the scan which is visible from the slope of the individual line graphs. This could possibly be because now the number of values is the dominant number in the algorithm and this becomes a huge factor when going to data dependency. Clearly at this point the advantage of moving from control dependency to data dependency does not provide the optimized results anymore.