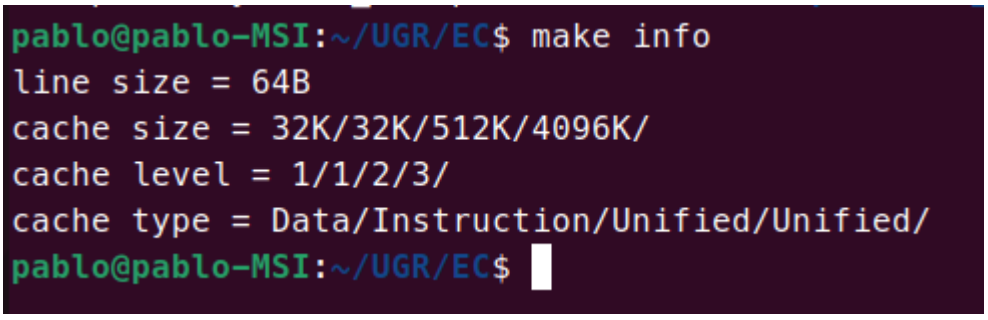
PRACTICA 6 PABLO LINARI PÉREZ

Si ejecutamos el comando lscpu obtenemos :

```
pablo@pablo-MSI:~/UGR/EC$ lscpu
Arquitectura:                         x86_64
  modo(s) de operación de las CPUs:   32-bit, 64-bit
  Address sizes:                      48 bits physical, 48 bits virtual
  Orden de los bytes:                 Little Endian
CPU(s):                               16
  Lista de la(s) CPU(s) en línea:     0-15
ID de fabricante:                     AuthenticAMD
  Nombre del modelo:                  AMD Ryzen 7 5700U with Radeon Graphics
    Familia de CPU:                   23
    Modelo:                           104
    Hilo(s) de procesamiento por núcleo: 2
    Núcleo(s) por «socket»:           8
    «Socket(s)»                       1
    Revisión:                         1
    Frequency boost:                  enabled
    CPU(s) scaling MHz:               35%
    CPU MHz máx.:                     4369,9209
    CPU MHz mín.:                     1400,0000
    BogoMIPS:                         3593,47
    Indicadores:                      fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp
                                       lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf rapl pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt a
                                      es xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr
                                      _core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate ssbd mba ibrs ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a
                                      rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr rdpr
                                      u wbnoinvd cppc arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic v_vmsave_vmload v
                                      gif v_spec_ctrl umip rdpid overflow_recov succor smca
Virtualization features:
  Virtualización:                     AMD-V
Caches (sum of all):
  L1d:                                256 KiB (8 instances)
  L1i:                                256 KiB (8 instances)
  L2:                                 4 MiB (8 instances)
  L3:                                 8 MiB (2 instances)
NUMA:
  Modo(s) NUMA:                       1
  CPU(s) del nodo NUMA 0:             0-15
Vulnerabilities:
  Gather data sampling:               Not affected
  Itlb multihit:                      Not affected
  L1tf:                               Not affected
  Mds:                                Not affected
  Meltdown:                           Not affected
  Mmio stale data:                    Not affected
  Retbleed:                           Mitigation; untrained return thunk; SMT enabled with STIBP protection
  Spec rstack overflow:               Mitigation; safe RET
  Spec store bypass:                  Mitigation; Speculative Store Bypass disabled via prctl
  Spectre v1:                         Mitigation; usercopy/swapgs barriers and __user pointer sanitization
  Spectre v2:                         Mitigation; Retpolines, IBPB conditional, STIBP always-on, RSB filling, PBRSB-eIBRS Not affected
  Srbds:                              Not affected
  Tsx async abort:                    Not affected
pablo@pablo-MSI:~/UGR/EC$
```

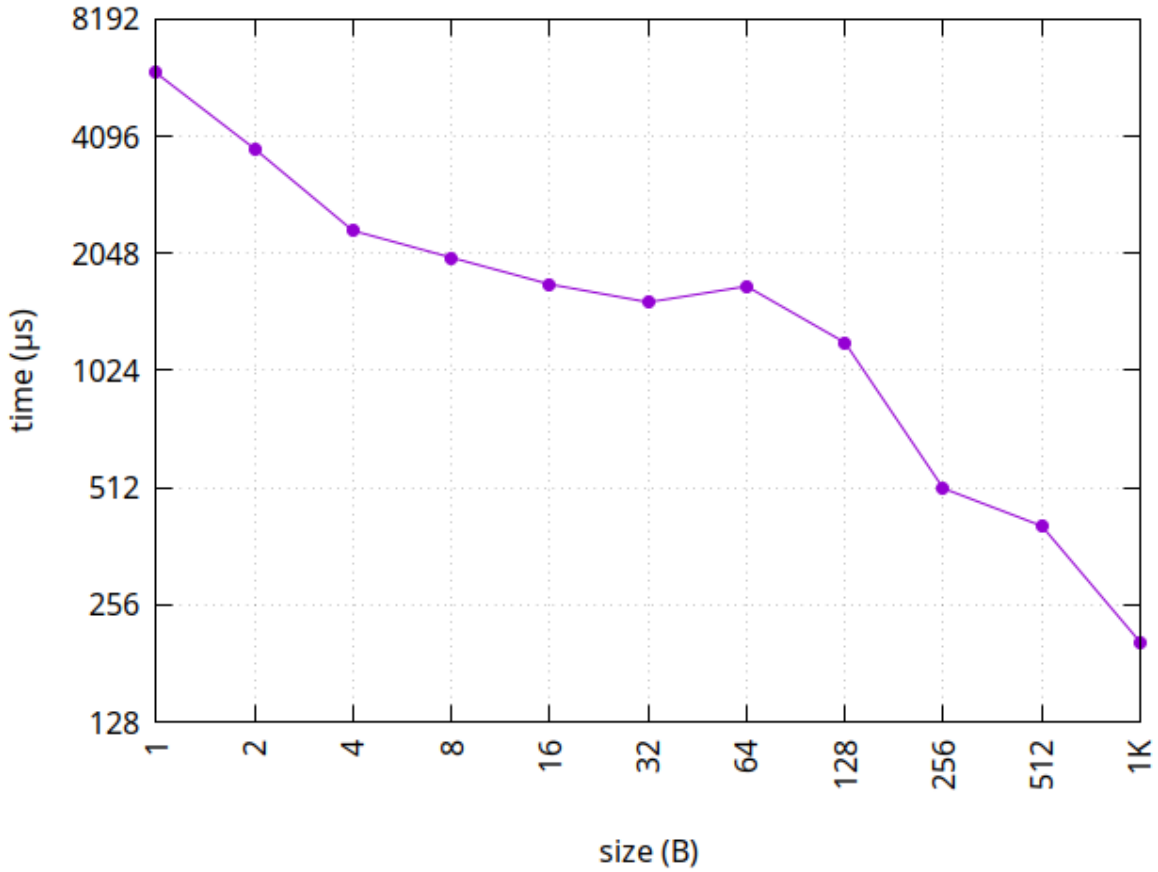
Y cuando ejecutamos make info obtenemos :



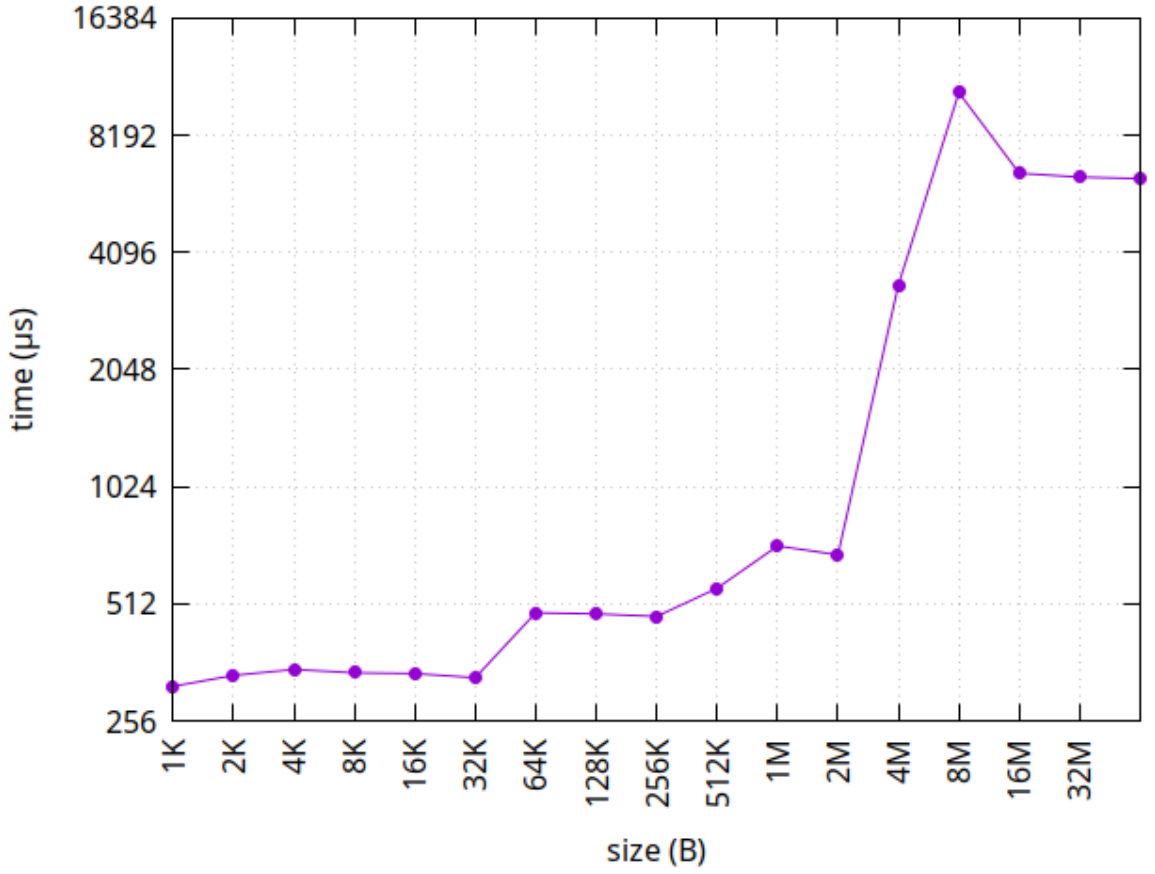
Podemos ver que el tamaño de las líneas es de 64B y que tenemos tres niveles de memoria caché el nivel 1 tiene 512 Kib el nivel 2 4 Mib el nivel 3 8 Mib.
El ordenador tiene 8 núcleos por lo que si hacemos el calculo de 64*8 obtenemos 512 que efectivamente es el tamaño que indica el comando lscpu .

En el fichero line.cc he añadido la línea de código “`bytes[i]^=1`” ya que solo nos interesa el tiempo de acceso por lo que uso el recomendado por el guión que realiza una operación rápida haciendo un XOR a cada byte.

En el fichero size.cc he añadido la linea de codigo ” `bytes[(i*64) & (size-1)]++;` ” ya que queremos estudiar los distintos niveles de la caché y como sabemos que el tamaño de línea son los 64 con este código accedemos a posiciones del vector separadas por el tamaño de línea así la caché se debe cargar por cada entrada al vector que se produzca .



En la gráfica se produce un decremento notable a partir de los 64 bytes, que es nuestro tamaño de línea, lo cual quiere decir que el tiempo de acceso se reduce, habiendo menos fallos si el tamaño es mayor que 64. Esto se debe a que si es menor que 64, estamos cargando datos innecesarios. Cuando avanzamos con un valor mayor que 64, se dejan sin cargar datos del vector que no necesitamos, dando lugar a un decrecimiento del tiempo en la gráfica ya que no cargamos datos innecesarios que ralentizan el programa.



En esta gráfica podemos observar tres trozos
constantes. El primero llega hasta los 32K, de forma que al acceder a una zona de memoria que no alcance ese límite, el tiempo de acceso es de más de 256 microsegundos. Al intentar acceder a más memoria, el tiempo de acceso pasa a ser de unos 800 microsegundos hasta llegar a los 2 MB pero el punto remarcable es cuando llega a 4 MB, deduciendo que el tamaño del nivel L2 de caché es de 4 MB. Finalmente, desde esos $ MB, el tiempo de acceso vuelve a
incrementar hasta establecerse a partir de los 8 MB donde deducimos que la Caché de nivel L3 es 8MB.