TLBs

Example ⇒ L1 TLB hit: ~5ns
L2 TLB hit: ~20ns
TLB miss: ~70ns

① gettimeofday() ⇒ Has precision of microsecond

wall-clock time

Operation taking longer than 1μs would be easier to measure, but above times are all ns so we will need to loop multiple times to time properly.

monotonic clock

We could use clock_gettime, that has clock_gettime_nsec_np. to get ns precision.

Iterations needed

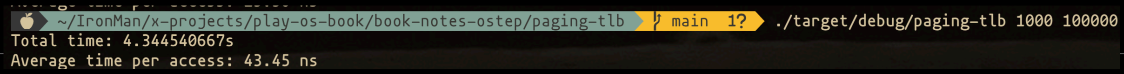
measurable duration (1ms = 1,000,000 ns)

$$\frac{1{,}000{,}000}{5ns/access} = 200{,}000 \text{ accesses}$$

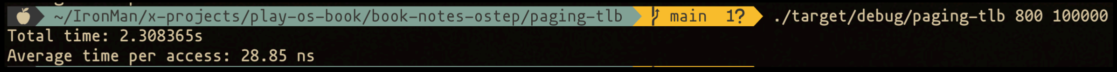
TLB miss

$$\frac{1000000}{70ns/access} \approx 14\,285 \text{ accesses}$$

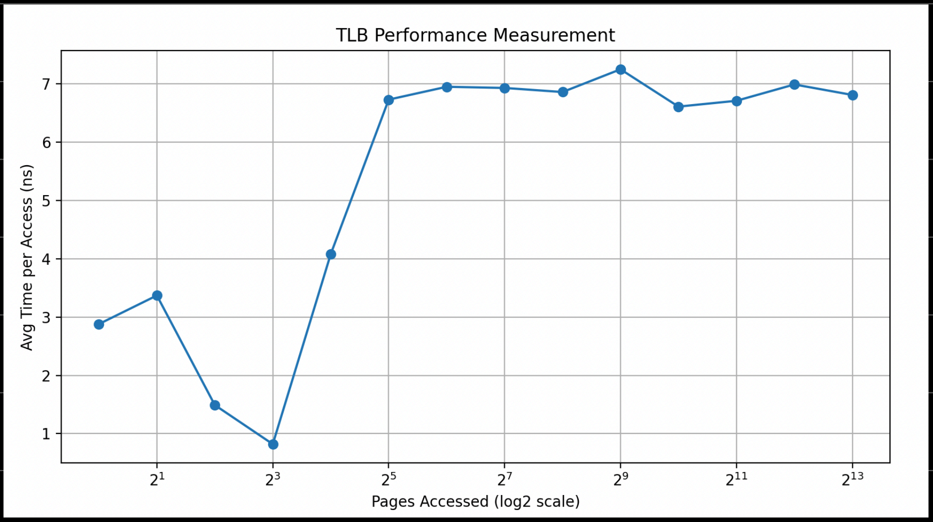
2



```
 ~/IronMan/x-projects/play-os-book/book-notes-ostep/paging-tlb  main  1?  ./target/debug/paging-tlb 1000 100000
Total time: 4.344540667s
Average time per access: 43.45 ns
```



```
 ~/IronMan/x-projects/play-os-book/book-notes-ostep/paging-tlb  main  1?  ./target/debug/paging-tlb 800 100000
Total time: 2.308365s
Average time per access: 28.85 ns
```



Compiler optimization

- Removing loops which increment values but unused post that.

Forcing compiler not to remove by using volatile variables that informs compiler not to optimise them away.

· write_volatile / read_volatile

Code can be executed on multiple CPUs, each with its own TLB. We don't want that to measure TLB access time more accurately.

To achieve that we need to pin our program thread to a single CPU

thread_policy_set

Initialization ⇒ If array for example isn't initialised before accessing it, first access is very expensive.

Due to initial access costs such as demand zeroing.

Note: TLB measurement done in this homework would still be off, due to things like caching prefetching, branch prediction, CPU prefetching