Arpit Chauhan
2019CS10332

15th Oct. 2022

COL732: Virtualization and Cloud Computing
Lab-3: Benchmarking Address Translation

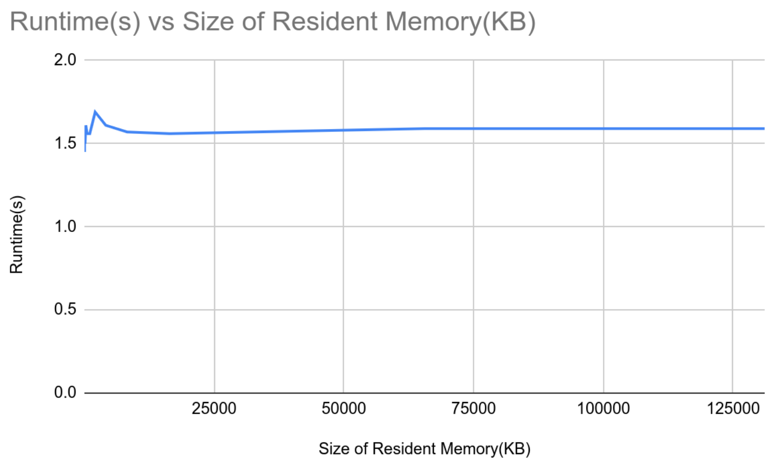
1) Page table benchmark

A. Experimental setup. Exact commands/programs that were run for the measurement.

- First of all, we boot the given Linux VM. Next we create a tap network and connect the VM with the host using the commands mentioned in the assignment. Then we compile the benchmark ```pagetable_bench.c``` using *gcc* and copy it from the host to the VM using the *nc* command given in the assignment.
- Let us say that the copied executable is ```a.out`` in the VM. Then we execute ```chmod +x a.out``` to give the file executable permissions in the VM.
- Next we run ```time ./a.out``` and note the real time displayed. We repeat this for varying sizes of the ```ptr``` array, for both, EPT=0 and EPT=1.

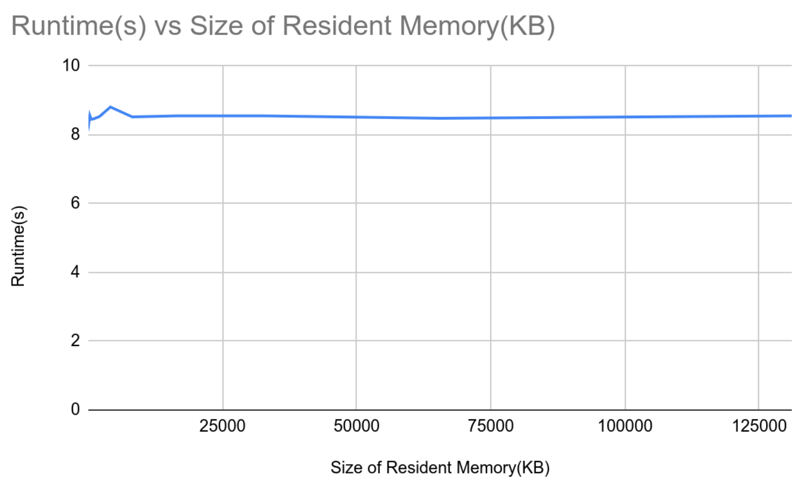
B. Relationship of runtime (with EPT=1) with the size of the resident memory (4KB-128MB)

- The runtime remained almost constant (in the range of 1.4-1.7s) with increase in the size of resident memory.



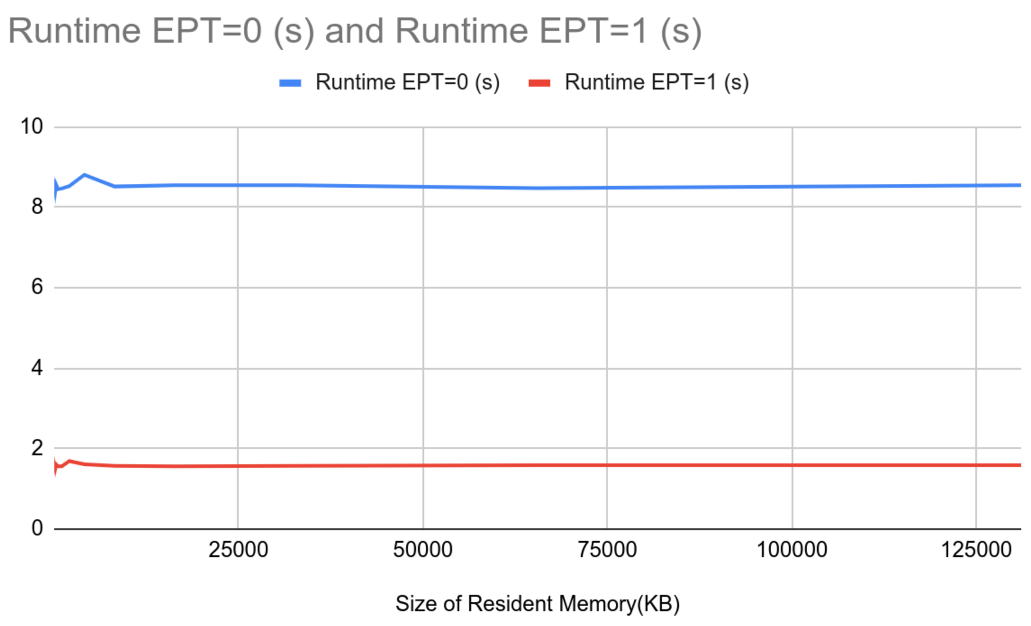
C. Relationship of runtime (with EPT=0) with the size of the resident memory (4KB-128MB)

- The runtime remained almost constant (in the range of 8-9s) with increase in the size of resident memory, although the values are higher than the case when EPT was enabled.



D. Justification of observations

- The values of runtimes in both the cases are constant with respect to size of resident memory because fork implements copy-on-write and since the benchmark does not write anything, minimal copying is involved.
- The values for runtimes for EPT=0 are significantly higher than for EPT=1 because when EPT=0 and shadow paging is being used, on every fork call, there will be a VM exit so that the Hypervisor can build the Shadow page table. These exits are expensive and thus explain the higher runtimes for EPT=0. In case of EPT=1, these exits are not required as we can just copy the parent page table and use the same EPT for address translation.



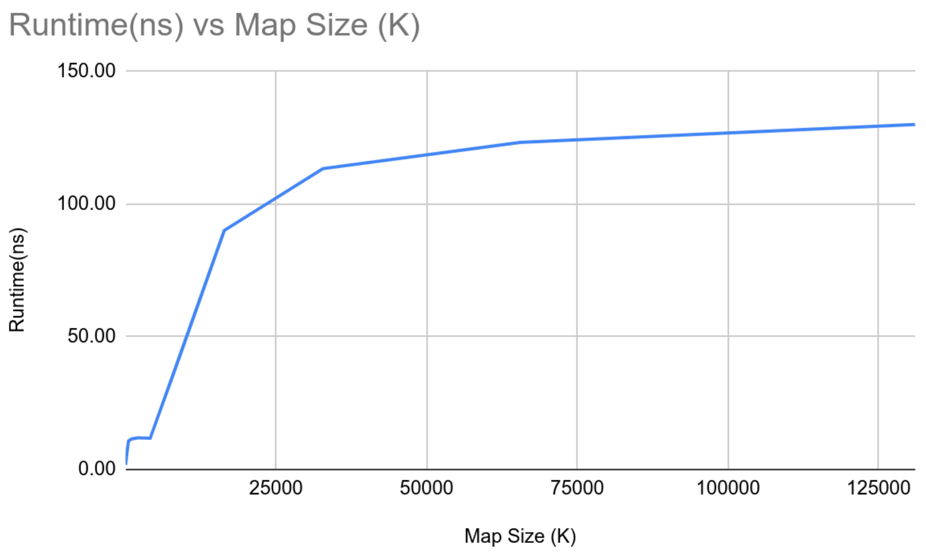
2) TLB benchmark

A. Experimental setup. Exact commands/programs that were run for the measurement

- First of all, we boot the given Linux VM. Next we create a tap network and connect the VM with the host using the commands mentioned in the assignment. Then we compile the benchmark ```tlb_bench.c``` using *gcc* and copy it from the host to the VM using the *nc* command given in the assignment.
- Let us say that the copied executable is ```a.out`` in the VM. Then we execute ```chmod +x a.out``` to give the file executable permissions in the VM.
- Next we run ```./a.out <map_size>``` and note the runtime and number of cycles displayed by the benchmark.. We repeat this for varying map sizes, for both, EPT=0 and EPT=1.

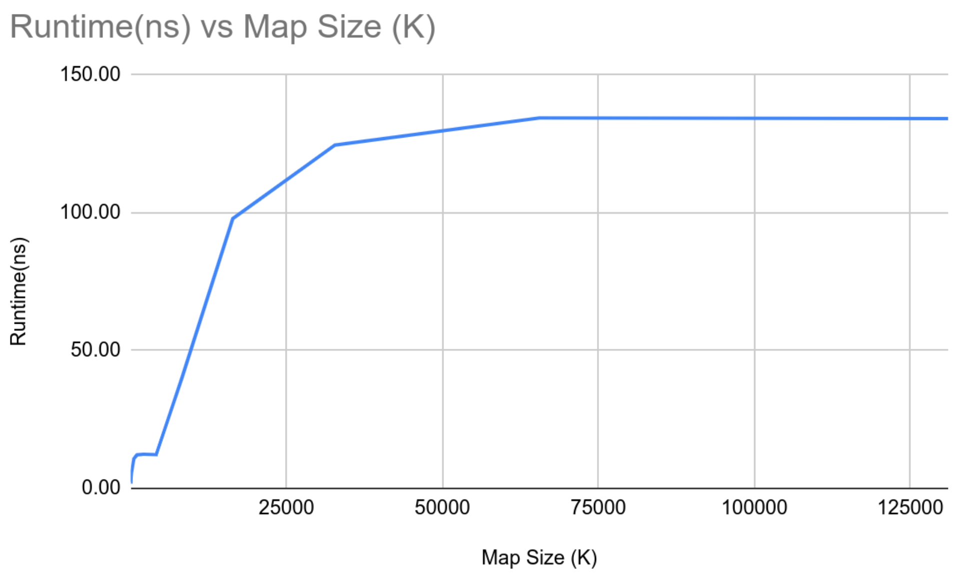
B. Relationship of runtime (with EPT=0) with the map size (4K – 256M)

- We observe that the runtime values increase with increase in map size. The runtime values are constant for a few initial values of map sizes, then start increasing drastically and later the rate of increase of runtime goes down.



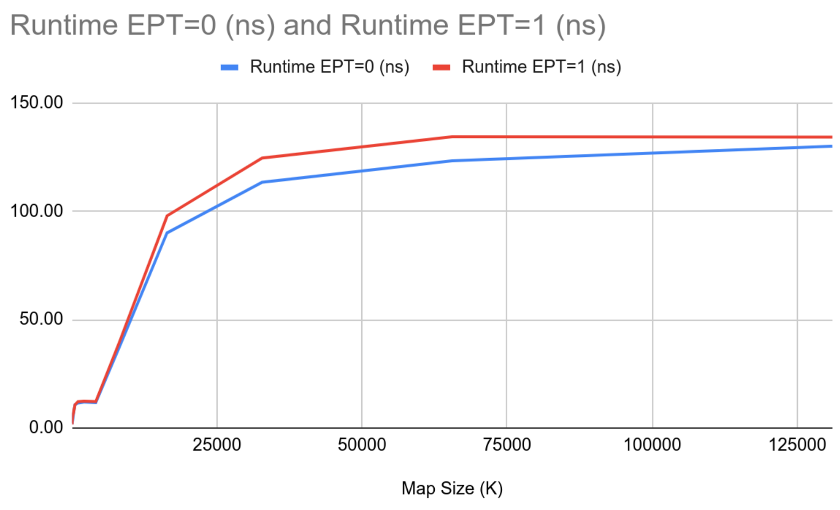
C. Relationship of runtime (with EPT=1) with the map size (4K – 256M)

- We observe that the runtime values increase with increase in map size. The runtime values are constant for a few initial values of map sizes, then start increasing drastically and later the rate of increase of runtime goes down in the same way as for EPT=0. The only difference being that the values for runtimes were slightly higher than for EPT=1.



D. Justification of observations

- We observe that for both EPT=0 and EPT=1, the runtime and cycle values are the same for small map sizes (< 32K). This is because at such small map sizes, there is no scope for TLB miss, and thus, it does not matter whether EPT is enabled or disabled, as there is no address translation required.
- For larger values of map size, we observe that the runtime and cycle counts for EPT=1 are greater than that for EPT=0. This is because if EPT=1 and there is a TLB miss, we need to perform an entire 2D address translation using the EPTP which is more expensive than the case where EPT=0 and shadow paging is implemented. Since the map size is larger, there will be more TLB misses therefore increasing the runtimes and cycles required if EPT is enabled. This explains the observation made in the plot given below.



APPENDIX - Readings

Page Table benchmark with EPT=0

Size of Resident Memory	Runtime(s)
4KB	8.26
8KB	8.49
16KB	8.35
32KB	8.51
64KB	8.48
128KB	8.4
256KB	8.56
512KB	8.45
1MB	8.47
2MB	8.53
4MB	8.81
8MB	8.52
16MB	8.55
32MB	8.55
64MB	8.48
128MB	8.55

Page Table benchmark with EPT=1

Resident Memory	Runtime(s)
4KB	1.45
8KB	1.46
16KB	1.47
32KB	1.51
64KB	1.55
128KB	1.5
256KB	1.61
512KB	1.56
1MB	1.56
2MB	1.69
4MB	1.61
8MB	1.57
16MB	1.56
32MB	1.57
64MB	1.59
128MB	1.59

TLB Benchmark with EPT=0

Map Size	Runtime(ns)	Cycles
4K	2.35	9.2
8K	2.35	9.2
16K	2.35	9.2
32K	2.35	9.2
64K	3.95	15.4
128K	3.94	15.4
256K	6.85	26.7
512K	10.73	41.9
1M	11.52	44.9
2M	11.96	46.6
4M	11.77	45.9
8M	37.57	146.5
16M	90.08	351.3
32M	113.42	442.3
64M	123.29	480.8
128M	130.02	507.1

TLB Benchmark with EPT=1

Map Size	Runtime(ns)	Cycles
4K	2.35	9.2
8K	2.35	9.2
16K	2.35	9.2
32K	2.35	9.2
64K	3.95	15.4
128K	5.72	22.3
256K	7.09	27.6
512K	10.74	41.9
1M	12.20	47.6
2M	12.37	48.2
4M	12.23	47.7
8M	39.83	155.3
16M	97.91	381.9
32M	124.58	485.9
64M	134.39	524.1
128M	134.21	523.4