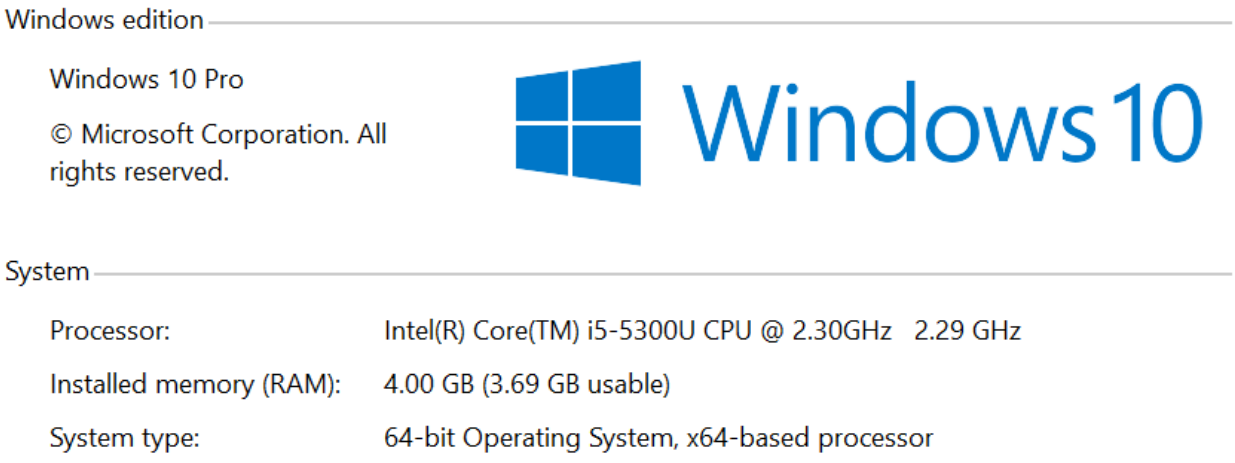
Jasmit Kakkar

COEN 241 HW#1
System vs OS Virtualization

The configuration of the host system

OS Name Microsoft Windows 10 Pro Version 10.0.19041 Build 19041
System Manufacturer LENOVO
System Model 20BS003CUS
System Type x64-based PC
System SKU LENOVO_MT_20BS_BU_Think_FM_ThinkPad X1 Carbon 3rd
Processor Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz, 2295 Mhz, 2 Core(s), 4 Logical
BIOS Version/Date LENOVO N14ET32W (1.10), 8/13/2015
SMBIOS Version 2.7
Hardware Abstraction Layer Version = "10.0.19041.1151"
Installed Physical Memory (RAM) 4.00 GB
Total Physical Memory 3.69 GB
Available Physical Memory 227 MB
Total Virtual Memory 11.4 GB
Available Virtual Memory 1.48 GB
Page File Space 7.71 GB



Steps taken to enable QEMU on my system

1. Install the latest version of QEMU from official website for 64 bit computers, currently the latest version of QEMU is 6.1
2. Run through the installation wizard
3. Navigate to the environment variables properties and add a PATH variable for qemu pointing to your installation folder
4. From here qemu is installed, next is installing a VM on qemu
5. Download an .iso image of whatever OS you wish to install, in our case this is ubuntu server 20.04
6. Run the following commands
    a. Qemu-img.exe create ubuntu.img 10G
    b. Qemu-system-x86_64.exe -hda ubuntu.img -boot d -cdrom ./ubuntu-20.04.3-live-server-amd64.iso -m 2048
7. The above commands will create a blank image with 10G of space and then boot the image from the file path you provided on qemu with the image you created.
8. QEMU should start up and boot the OS you provided, from here you can start the OS installation, in our case, for ubuntu server, the installation is straightforward.
9. Once the installation is complete, qemu will reboot and you can run qemu with the image with the following command
    a. Qemu-system-x86_64.exe -hda ubuntu.img -m 2048
10. Qemu should be running fine and the OS you installed should be running fine

Here are the configurations for QEMU I used in my various experiments…

Qemu-system-x86_64.exe -hda ubuntu.img -m 512
Qemu-system-x86_64.exe -hda ubuntu.img -m 2048
Qemu-system-x86_64.exe -hda ubuntu.img -m 2048 -accel whpx
Qemu-system-x86_64.exe -hda ubuntu.img -m 2048 -smp 16,sockets=1,cores=16
Qemu-system-x86_64.exe -hda ubuntu.img -m 4G -smp 2 -object memory-backend-ram,size=2G,id=m0 - object memory-backend-ram,size=2G,id=m1 -numa node,nodeid=0,memdev=m0 -numa node,nodeid=1,memdev=m1

The above configuration will test qemu with 512MB of RAM, 2G of RAM, aiwth a WHPX accelerator on, emulating 16 CPUs with smp option, and emulating a numa node configuration of 2 CPUs with 2G each

Steps taken to enable Docker Container on my system

1. Navigate to the Docker website and download the installation for Docker Desktop
2. Run the exe file to install Docker Desktop
3. Docker CLI should now be enabled, enter the command line and type docker to confirm
4. Download the provided image which contains ubuntu with sysbench preinstalled
5. Run the following command to boot up the image in docker
    a. Docker run -it csminpp/ubuntu-sysbench bash
6. Docker should be running fine and the OS you installed should be running fine as well

Some of the operations that I used to manage the containers and images are as follows…

Docker images
This command will show a list of all available images locally stored in the docker engine

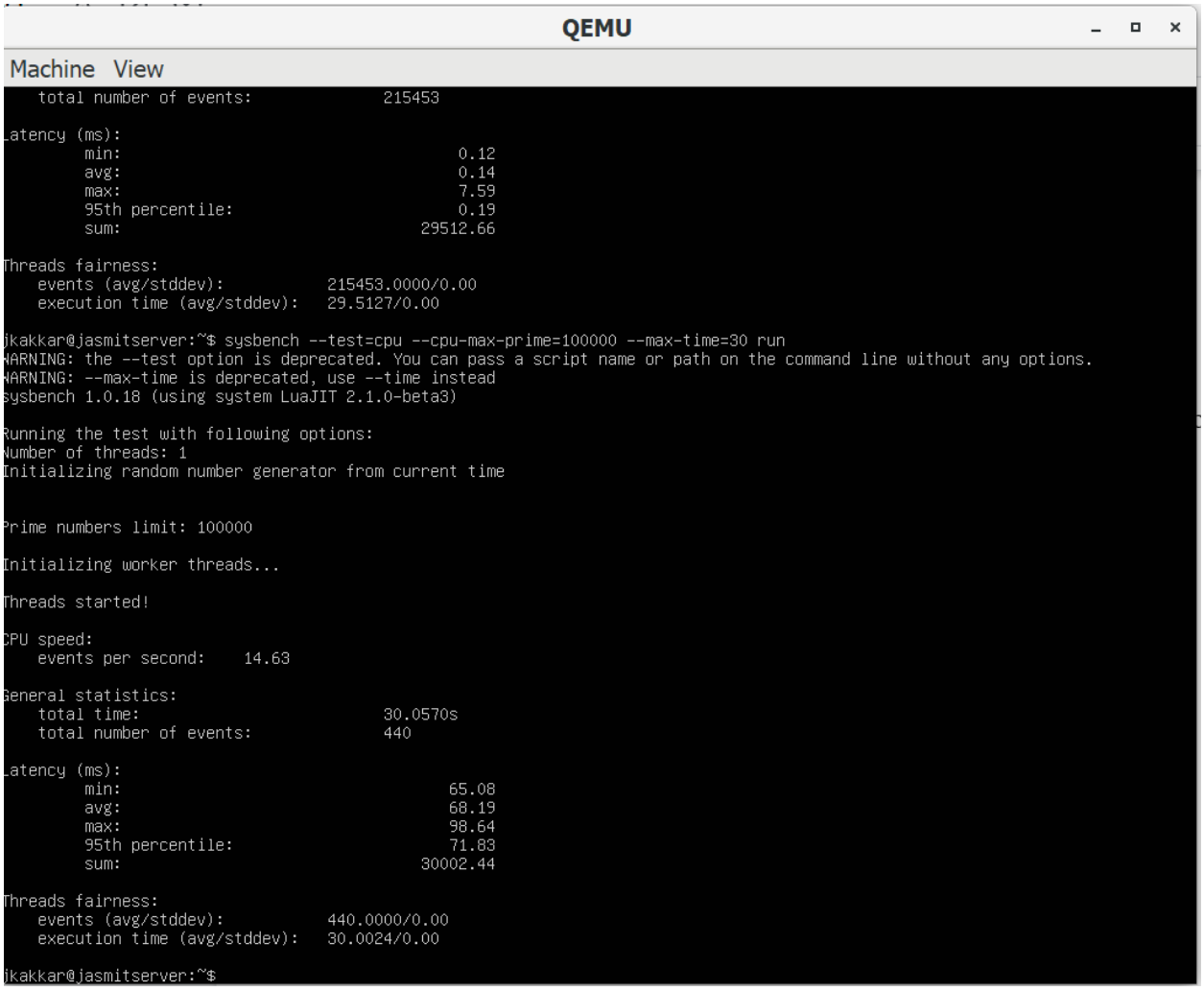
Docker pull
This will pull an image from a specified repository

Docker ps
This will show a list of all the containers currently running on the Docker engine

Docker stop
This will stop a specified running container

Docker run
This command will run a specified container, we can use additional options as well such as -it to allocate tty for processes like shell

Below is a proof of experiment for QEMU, this image shows QEMU running ubuntu server 20.04 and finishing a sysbench cpu test



Below is a proof of experiment for Docker, the image shows a docker container running ubuntu live server 20.04 finishing a sysbench cpu test

```
root@da105763ffed: /
root@da105763ffed:/# ./sysbench_script.sh > output.txt
root@da105763ffed:/# sysbench --test=cpu --cpu-max-prime=1000 --max-time=30 run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 1000


Test execution summary:
    total time:                          0.4980s
    total number of events:              10000
    total time taken by event execution: 0.4950
    per-request statistics:
         min:                                  0.04ms
         avg:                                  0.05ms
         max:                                  0.46ms
         approx.  95 percentile:               0.06ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   0.4950/0.00

root@da105763ffed:/#
```

How the experiments are performed

The experiments are placed into separate script files to run, as an example the cpu-max-prime test script is shown below…

```
#!/bin/bash

sysbench --test=cpu --cpu-max-prime=100 --max-time=60 --max-requests=10000 run
sysbench --test=cpu --cpu-max-prime=1000 --max-time=60 --max-requests=10000 run
sysbench --test=cpu --cpu-max-prime=10000 --max-time=60 --max-requests=10000 run

```

Following it are the fileIO commands I use

```
#!/bin/bash

# sequential write test
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqwr prepare
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqwr run
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqwr cleanup

# sequential read test
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqrd prepare
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqrd run
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=seqrd cleanup

# combined random read/write test
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=rndrw prepare
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=rndrw run
sysbench --num-threads=16 --test=fileio --file-total-size=2G --file-test-mode=rndrw cleanup
```

These files are placed onto the VM and given the right permissions to be executable (chmod 755 [script_file_name.sh]) from here we can run the scripts using the following command…
./script_file_name.sh > output_file_name.txt & ; top
This command will run the script and place the output into the destination specified. It will run this script command in the background because of the & tag, and it also runs an instance of top command to show the CPU utilization at user level vs kernel level while the benchmark tests are being performed.

From here we can open/process the output file to extract the relevant data and record it.

For each scenario/script we have, we can do the process above over again. This will be done 4 times to accommodate for the following test scripts I have provided, and then the whole process is done 5 times total to get better estimates.

Sysbench_cpu_prime.sh
Sysbench_fileio_seqwr.sh
Sysbench_fileio_seqrd.sh
Sysbench_fileio_rndrw.sh

In between each run of the fileio scripts we must also clear the cache, since the files created in the previous test will now be stored in the cache we will use the windows sync tool. This tool will direct the OS to flush the file system data in cache to disk. If we do not run this command there will be inconsistencies in the tests and you will see that after the first test, subsequent tests will finish much quicker.

Below I include the raw test data that I collected from all the tests

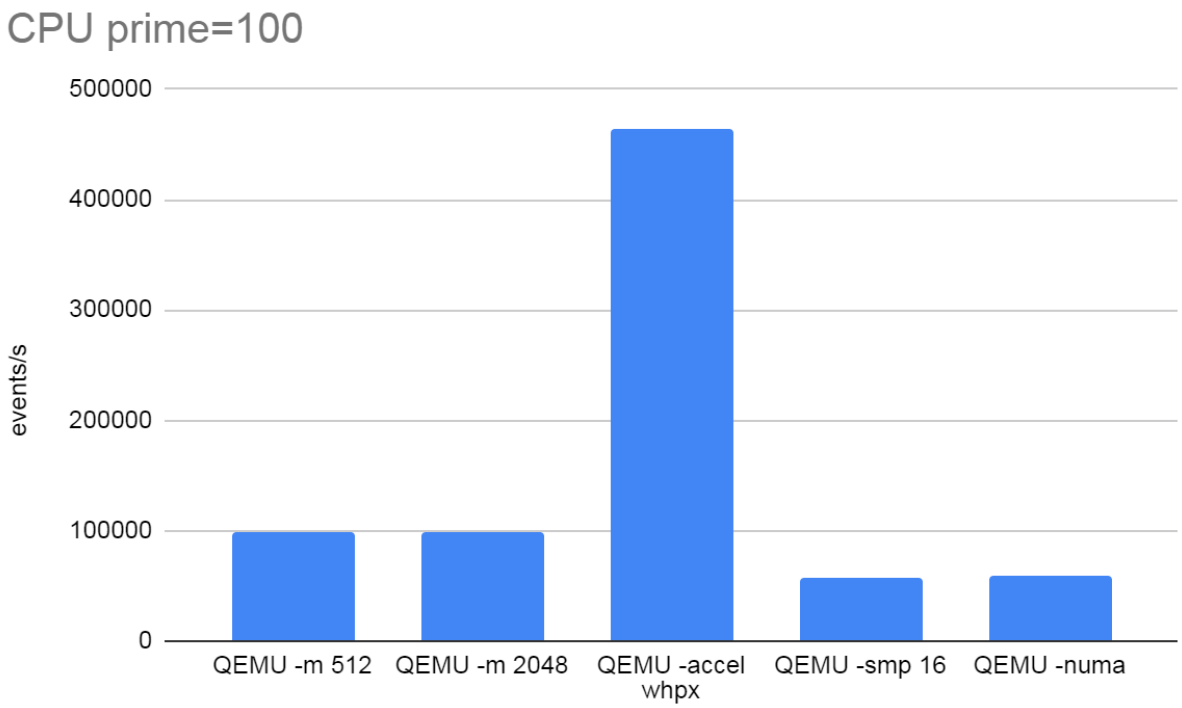
The values from all 5 experiments for all 6 VM configurations is provided in github as well, as it si too much data to include in this report.
NOTE: you will have to zoom in to see all the data on the PDF

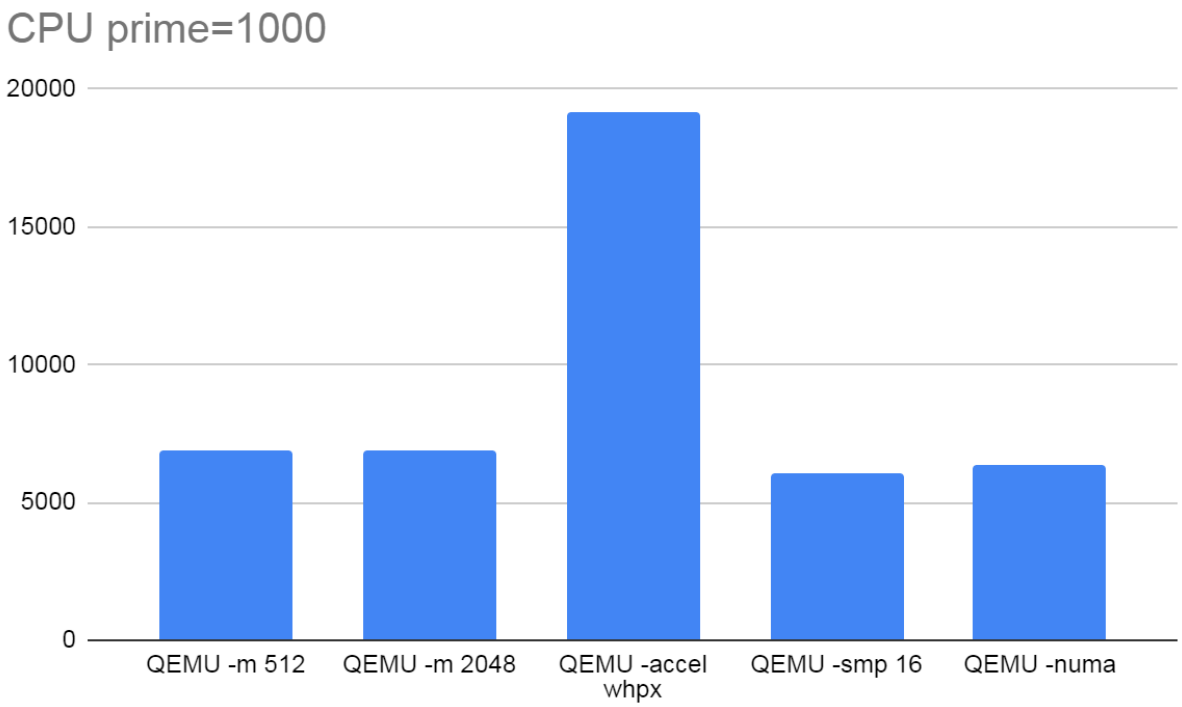
Here is a link to the github repository where everything is stored…
It is open to public

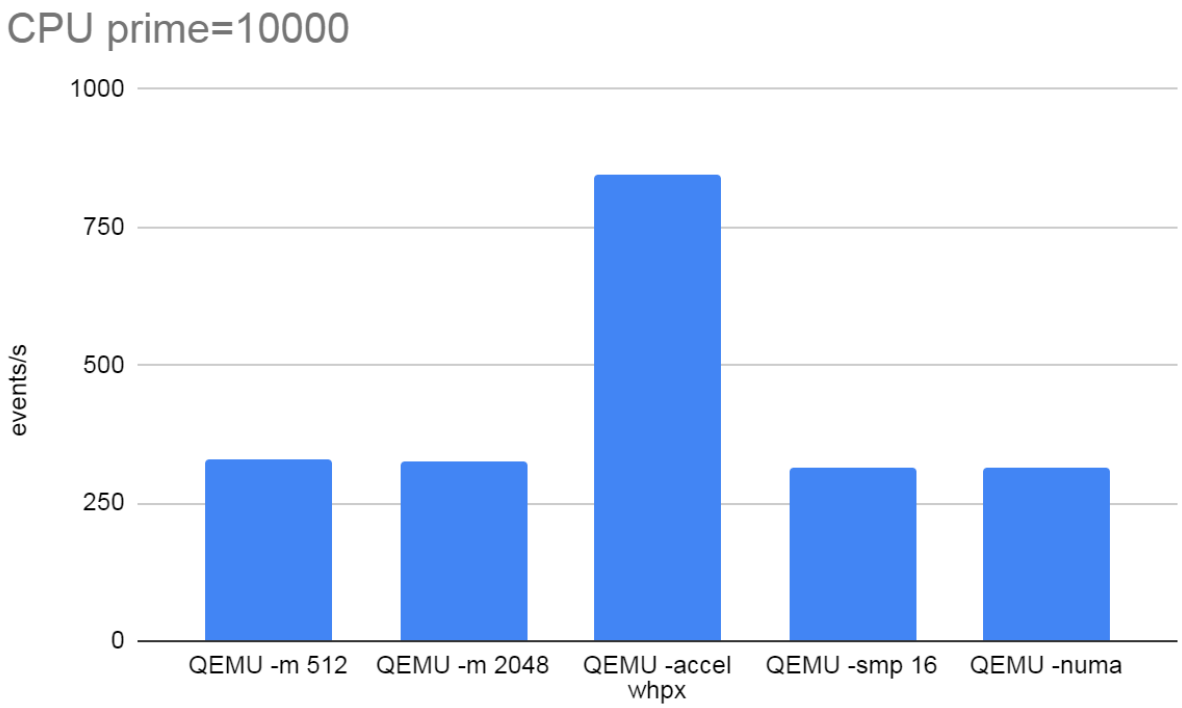
https://github.com/Portabello/COEN241-HW1

		min	max	avg	std
QEMU -m 512	events/s	90498	105782	98283	6331.18
	events/s	6161	7145	6909	375.75
	events/s	318	336	330	6.48
	writes/s	488	542	512	21.13
	reads/s	3418	3679	3595	91.89
	fysnc/s	343	441	401	32.73
QEMU -m 2048	events/s	92506	102726	98775	3799.52
	events/s	6284	7107	6857	303.74
	events/s	306	338	325	11.82
	writes/s	491	570	537	27.86
	reads/s	3523	4549	3984	377.79
	fysnc/s	433	483	471	21.34
QEMU -m 2048 -accel whpx	events/s	402570	486045	463723	31003.21
	events/s	18394	20488	19194	693.26
	events/s	836	851	847	5.59
	writes/s	975	1003	987	11.6
	reads/s	4832	6461	5823	599.89
	fysnc/s	472	541	509	25.75
QEMU -m 2048 -smp 16,sockets=1,cores=16	events/s	50485	61616	56774	4396.81
	events/s	5704	6413	6093	323.06
	events/s	309	322	314	4.59
	writes/s	168	282	228	39.38
	reads/s	641	788	699	50.24
	fysnc/s	368	403	383	15.09
QEMU -numa	events/s	57477	63110	60100	2104.44
	events/s	5840	6712	6343	304.46
	events/s	308	328	316	7.28
	writes/s	428	468	446	14.82
	reads/s	32544	39697	37199	2434.96
	fysnc/s	503	524	517	7.84
DOCKER	events/s	312500	344827	327164	11019.05
	events/s	19193	23752	21468	1472.65
	events/s	950	975	969	9.66
	writes/s	21232	27590	24938	2310.12
	reads/s	18301	24876	21478	2237.52
	fysnc/s	520	540	531	7.95

Here are graphs showing the CPU max prime test for the values 100, 1000, 10000 across all systems



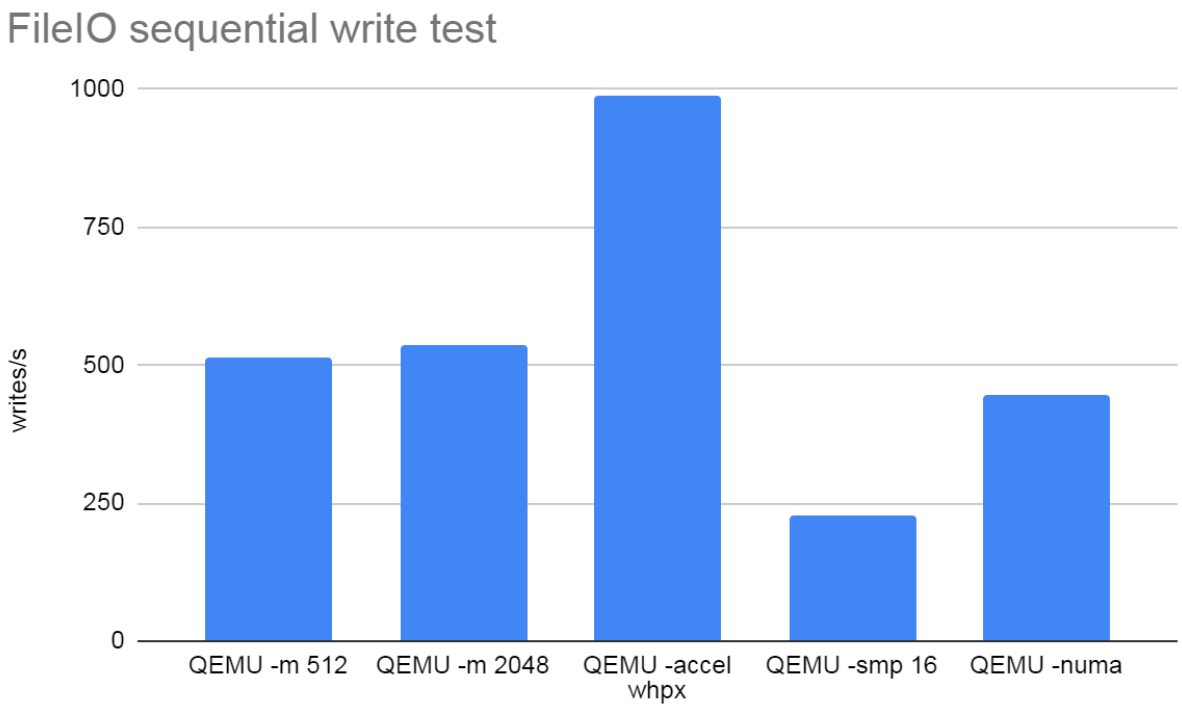


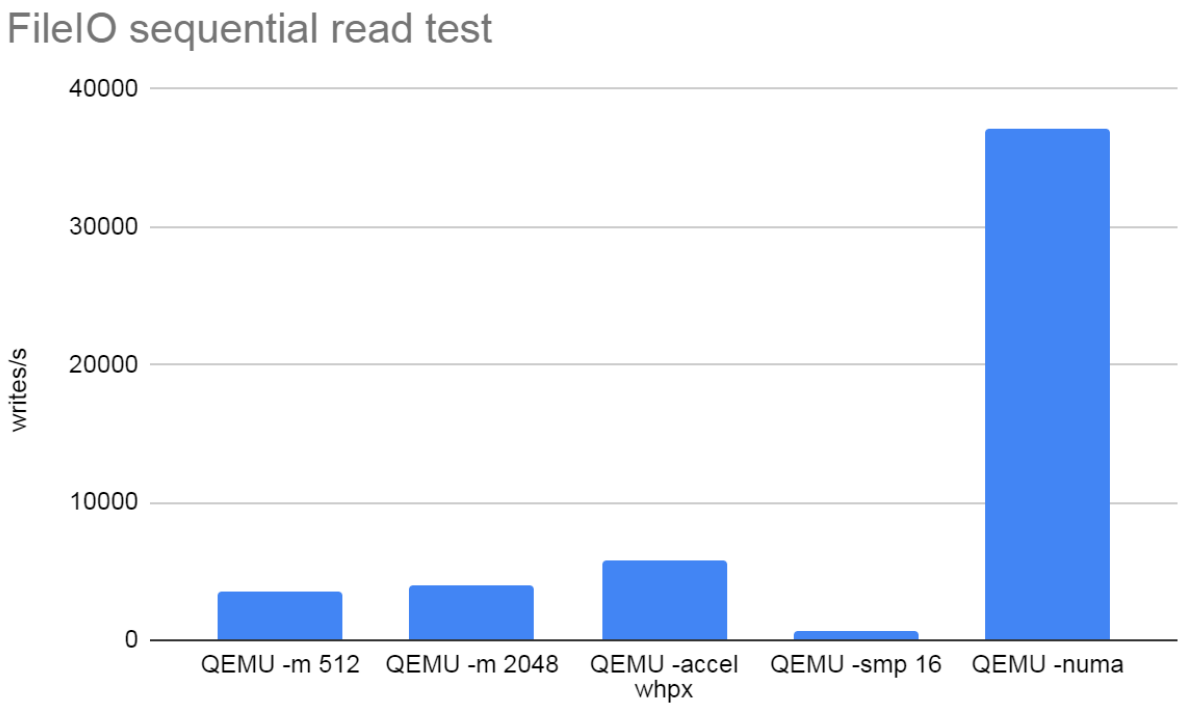


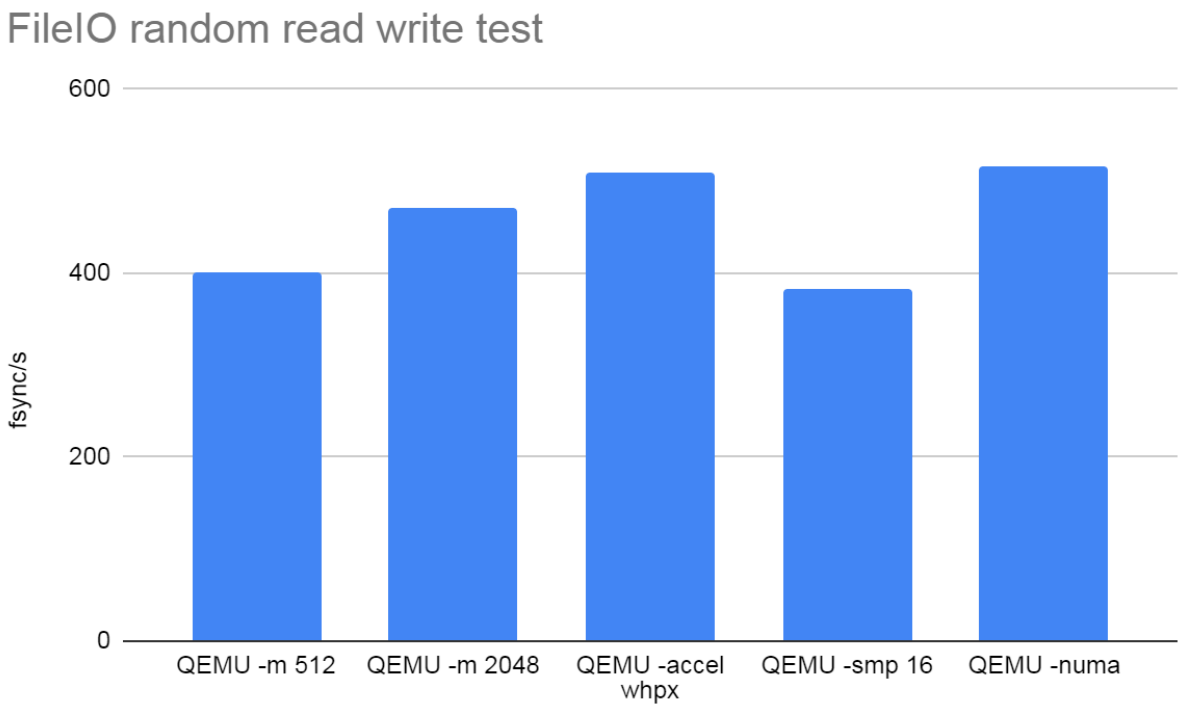
As you can see, the QEMU variant will a whpx accelerator consistently performed higher in the CPU-max-prime test across the board, no other configurations really showed any sort of variance

from the default QEMU setup, which lead to me believe that the options set in those tests do not have much of an impact when performing CPU tests

Below are some graphs showing the FileIO tests for sequential write, sequential read, and random read/write



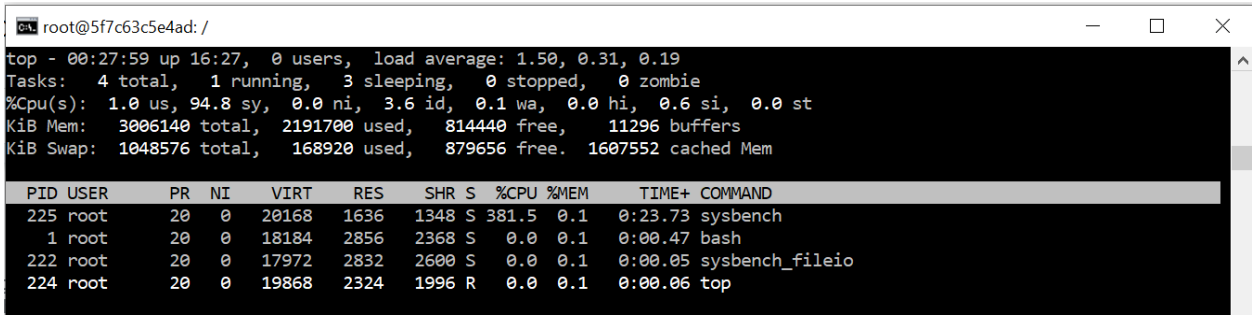




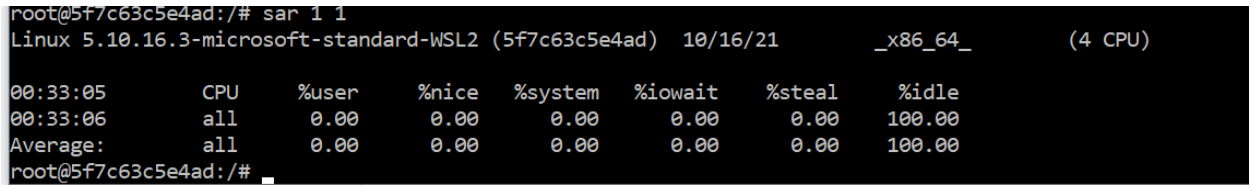
As you can see the accelerator whpx performed well in the write test in terms of writes/s. The NUMA configuration for CPU surprisingly overperformed from what I was expecting in the sequential read test, and consistently scored extraordinarily high.

Some additional performance tools I used to measure and collect data were top. This is a linux command that will show CPU utilization, as you can see below it is showing CPU utilization while the sysbench process is running.

The tags 94.8sy and 3.6id tell us that the CPU is at 94.8% utilization for user space processes and 3.6% utilized in system idle. After this sysbench process completes you will notice the user space utilization go to zero (as long as no other processes are running besided sysbench and mandatory ones) and the system idle will near 100%



To measure the I/O throughput and latency I used to sar command in linux. This will show disk latency, if needed, disk utilization, and I/O latency



Finally below I provide the raw experiment data, also included in github...

		prime	fileio			
		[events/s , total time , avg ms]	[reads/s, writes/s, fsync/s]			
		cpu #1	cpu #2	cpu #3	cpu #4	cpu #5
QEMU -m 512	prime100	90498, 0.1021, 0.01	102909, 0.0894, 0.01	101252, 0.0918, 0.01	90978, 0.1022, 0.01	105782, 0.0878, 0.01
	prime1000	6161, 1.6154, 0.16	7057, 1.4097, 0.14	7120, 1.3966, 0.14	7145, 1.392, 0.14	7065, 1.41, 0.14
	prime10000	329, 30.3012, 3.02	332, 30.11, 3.00	335, 29.7746, 2.97	318, 31.34, 3.12	336, 29.72, 2.96
	fileio seqwr	0, 542, 872	0, 522, 845	0, 488, 796	0, 488, 804	0, 521, 846
	fileio seqrd	3649, 0, 0	3679, 0, 0	3606, 0, 0	3418, 0, 0	3623, 0, 0
	fileio rndrw	120, 79, 441	112, 74, 418	89, 59, 343	108, 71, 396	110, 73, 410
QEMU -m 2048	prime100	92506, 0.098, 0.01	102570, 0.09, 0.01	97094, 0.095, 0.01	98983, 0.093, 0.01	102726, 0.089, 0.01
	prime1000	6812, 1.46, 0.14	7028, 1.41, 0.14	7107, 1.399, 0.14	6284, 1.58, 0.15	7055, 1.4, 0.14
	prime10000	306, 32.65, 3.25	325, 30.67, 3.06	338, 29.53, 2.94	320, 31.23, 3.11	337, 29.66, 2.96
	fileio seqwr	0, 570, 906	0, 562, 897	0, 530, 862	0, 491, 806	0, 535, 861
	fileio seqrd	3523, 0, 0	4109, 0, 0	3606, 0, 0	4137, 0, 0	4549, 0, 0
	fileio rndrw	142, 95, 475	141, 94, 471	149, 99, 497	121, 80, 433	151, 100, 483
QEMU -m 2048 -accel whpx	prime100	474707, 0.019, 0	483105, 0.018, 0	402570, 0.022, 0	486045, 0.018, 0	472192, 0.019, 0
	prime1000	20488, 0.4859, 0.05	18964, 0.52, 0.05	19095, 0.52, 0.05	19033, 0.52, 0.05	18394, 0.54, 0.05
	prime10000	848, 11.78, 1.18	850, 11.75, 1.17	851, 11.73, 1.17	850, 11.74, 1.17	836, 11.95, 1.19
	fileio seqwr	0, 976, 1435	0, 999, 1465	0, 975, 1433	0, 984, 1442	0, 1003, 1470
	fileio seqrd	4832, 0, 0	5721, 0, 0	5668, 0, 0	6461, 0, 0	6436, 0, 0
	fileio rndrw	152, 101, 506	171, 114, 541	144, 96, 492	142, 94, 472	164, 109, 534
QEMU -m 2048 -smp 16,sockets=1,cores=16	prime100	61619, 0.15, 0.01	58936, 0.16, 0.01	50484, 0.19, 0.01	60203, 0.15, 0.01	52628, 0.18, 0.01
	prime1000	6513, 1.52, 0.14	6440, 1.54, 0.15	5941, 1.67, 0.16	5704, 1.74, 0.17	5868, 1.69, 0.16
	prime10000	311, 32.08, 3.19	309, 32.25, 3.21	314, 31.78, 3.16	322, 31.02, 3.09	317, 31.52, 3.14
	fileio seqwr	0, 168, 388	0, 229, 446	0, 257, 493	0, 282, 528	0, 208, 434
	fileio seqrd	672, 0, 0	788, 0, 0	681, 0, 0	716, 0, 0	641, 0, 0
	fileio rndrw	118, 78, 401	105, 71, 371	115, 76, 403	105, 70, 368	107, 71, 376
QEMU -numa	prime100	61939, 0.15, 0.01	59570, 0.16, 0.01	57477, 0.16, 0.01	59407, 0.16, 0.01	63110, 0.15, 0.01
-m 4G	prime1000	6470, 1.54, 0.14	6177, 1.61, 0.15	5840, 1.71, 0.16	6712, 1.48, 0.14	6518, 1.52, 0.14
-object mem-backend-ram 1G...	prime10000	308, 32.37, 3.21	328, 30.39, 3.02	321, 31.12, 3.09	312, 31.97, 3.18	312, 31.98, 3.18
-object mem-backend-ram 1G...	fileio seqwr	0, 428, 719	0, 435, 729	0, 441, 740	0, 468, 770	0, 458, 758
-numa node 1...	fileio seqrd	37984, 0, 0	37591, 0, 0	32544, 0, 0	38182, 0, 0	39697, 0, 0
-numa node 2...	fileio rndrw	158, 105, 502	164, 109, 520	162, 108, 520	167, 111, 521	165, 110, 524
DOCKER	prime100	322580, 0.0031, 0	322580, 0.0031, 0	344827, 0.0029, 0	333333, 0.003, 0	312500, 0.0032, 0
	prime1000	19193, 0.0521, 0.05	21786, 0.0459, 0.05	20920, 0.0478, 0	23752, 0.0421, 0.04	21691, 0.0461, 0.05
	prime10000	974, 1.02, 1.03	975, 1.0253, 1.02	972, 1.0283, 1.03	975, 1.0251, 1.02	950, 1.052, 1.05
	fileio seqwr	21232 writes/s	27590 writes/s	23430 writes/s	25803 writes/s	26639 writes/s
	fileio seqrd	18301 reads/s	24876 reads/s	19945 reads/s	21834 reads/s	22435 reads/s
	fileio rndrw	520 fsync/s	532 fsync/s	540 fsync/s	524 fsync/s	539 fsync/s