CXL and HMSDK Setup using QEMU

Started with Ubuntu Plucky : Linux 6.14.08

qemu-system-x86_64 –version : 9.2.1

Had `CONFIG_CXL` support (after downloading `linux-modules-extra` and `sudo modprobing cxl_mem`)

Most helpful doc : Fujitsu_PDF among others.

Official QEMU CXL doc : Qemu website

Also need to install `ndctl` (building from source should not be needed).

Key takeaways from doc :

- `cxl-fmw.0.size=16G` : this size should be more size of the CXL memory, to allow creation of CXL region
- KVM inside QEMU CXL will cause issues.

Command to create volatile CXL memory region:

```
sudo cxl create-region -m mem0 -d decoder0.0 -t ram
```

The region created will be the size of `mem0` (need to ensure size of `decoder0.0` more than size of `mem0` as stated above)

Linux 6.14 with QEMU 9.2 is sufficient to "detect" a CXL device, without needing source build (since we could have all modules asked mandatory on slide 19 (except one))

BUT, issue lies in creating the CXL region:

CXL `create-region` command fails : `cxl region: create_region: region0: failed to commit decode: No such device or address when trying inside qemu vm`

`dmesg` shows issue with synchronizing cache.

Looking at slide 19 : It mentions, CONFIG_CXL_REGION_INVALIDATION_TEST is essential in **emulation environment**. (Perhaps not needed with actual CXL hardware). Reason : Reference

Conclusion 1 : Current mainline kernel can support real CXL without rebuild but will most likely need a rebuild for QEMU emulation (because `CONFIG_CXL_REGION_INVALIDATION_TEST` is unlikely to be `y` by default)

So we have got one reason to build the kernel :

for getting `CONFIG_CXL_REGION_INVALIDATION_TEST=y` and `CONFIG_MEMORY_HOTPLUG_DEFAULT_ONLINE=y`

(makes life easier since just creating cxl-region brings memory online).

Also for hmsdk capacity expansion, a DAMON-enabled kernel is needed and if `CONFIG_DAMON` is not `y` (mostly true for Ubuntu versions), that gives another reason to build.

So went ahead with compiling the linux submodule in hmsdk repo : it's **Linux 6.12**. Qemu is still 9.2.

Followed this from their wiki:

```
$ cd hmsdk/linux
$ cp /boot/config-$(uname -r) .config
$ echo 'CONFIG_DAMON=y' >> .config
$ echo 'CONFIG_DAMON_VADDR=y' >> .config
$ echo 'CONFIG_DAMON_PADDR=y' >> .config
$ echo 'CONFIG_DAMON_SYSFS=y' >> .config
$ echo 'CONFIG_MEMCG=y' >> .config
$ echo 'CONFIG_MEMORY_HOTPLUG=y' >> .config
# also CONFIG_CXL_REGION_INVALIDATION_TEST=y and CONFIG_MEMORY_HOTPLUG_DEFAULT_ONLINE=y
$ make menuconfig
$ make -j$(nproc)
# had to turn off some certificate configs
$ sudo make INSTALL_MOD_STRIP=1 modules_install
$ sudo make headers_install
$ sudo make install
```

After making the GRUB menu visible (which is not in cloud-init by default), we are good to go.

Initial script (almost, since I might have changed some parameter which I may not remember) which first showed some success:

```
#!/bin/bash

VAR_MAIN_MEM_SIZE=8
VAR_CXL_MEM_SIZE=8
VAR_CPU_CNT=12
MAX_MEM_SIZE=$((${VAR_MAIN_MEM_SIZE} + ${VAR_CXL_MEM_SIZE} + 20))

qemu-system-x86_64 \
  --nographic \
  -m ${VAR_MAIN_MEM_SIZE}G,maxmem=${MAX_MEM_SIZE}G,slots=8 \
  -machine q35,accel=tcg,cxl=on \
  -smp cpus=$VAR_CPU_CNT \
  -drive file=$1,index=0,format=qcow2,if=none,id=disk0 \
  -device virtio-blk-pci,drive=disk0,id=virtio-disk0 \
  -netdev user,id=net0 \
  -device virtio-net-pci,netdev=net0,bus=pcie.0 \
  -object memory-backend-ram,size=${VAR_CXL_MEM_SIZE}G,id=m1,share=on \
  -device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 \
```

```
-device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 \
-device cxl-type3,bus=root_port13,volatile-memdev=m1,id=cxl-vmem0 \
-M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=16G
```

This script allowed me to use almost all features, and could start damo through hmsdk as well.
Illustrative Examples :

- Before creating CXL region :

```
ubuntu@cloudimg:~$ dmesg | grep -i cxl
dmesg: read kernel buffer failed: Operation not permitted
ubuntu@cloudimg:~$ sudo dmesg | grep -i cxl
[    3.280308] acpi ACPI0016:00: _OSC: OS supports [CXL11PortRegAccess CXL20PortDevRegAccess CXLProtocolErrorReporting CXLNativeHot]
[    3.287858] acpi ACPI0016:00: _OSC: OS now controls [CXLMemErrorReporting]
[   25.536580]  pci0000:0c: host supports CXL
ubuntu@cloudimg:~$ ls /sys/bus/cxl/devices/
decoder0.0  decoder1.0  decoder2.0  decoder2.1  decoder2.2  decoder2.3  endpoint2  mem0  nvdimm-bridge0  port1  root0
ubuntu@cloudimg:~$ cxl list -MDu
[
  {
    "memdevs":[
      {
        "memdev":"mem0",
        "ram_size":"8.00 GiB (8.59 GB)",
        "serial":"0",
        "host":"0000:0d:00.0"
      }
    ]
  },
  {
    "root decoders":[
      {
        "decoder":"decoder0.0",
        "size":"16.00 GiB (17.18 GB)",
        "interleave_ways":1,
        "max_available_extent":"16.00 GiB (17.18 GB)",
        "pmem_capable":true,
        "volatile_capable":true,
        "accelmem_capable":true,
        "nr_targets":1
      }
    ]
  }
]
ubuntu@cloudimg:~$ cxl list -R
  Warning: no matching devices found
```

- After creating CXL region (`sudo cxl create-region -m mem0 -d decoder0.0 -t ram`):

```
ubuntu@cloudimg:~/.../cxl-index$ sudo dmesg | grep -i cxl
[    3.296644] acpi ACPI0016:00: _OSC: OS supports [CXL11PortRegAccess CXL20PortDevRegAccess CXLProtocolErrorReporting CXLNativeHot]
[    3.303597] acpi ACPI0016:00: _OSC: OS now controls [CXLMemErrorReporting]
[   25.896328]  pci0000:0c: host supports CXL
[   65.298593] cxl region0: Bypassing cpu_cache_invalidate_memregion() for testing!
ubuntu@cloudimg:~/.../cxl-index$ ls /sys/bus/cxl/devices/
dax_region0  decoder0.0  decoder1.0  decoder2.0  decoder2.1  decoder2.2  decoder2.3  endpoint2  mem0  nvdimm-bridge0  port1  region0  root0
ubuntu@cloudimg:~/.../cxl-index$
ubuntu@cloudimg:~/.../cxl-index$ cxl list -MDu
[
  {
    "memdevs":[
      {
        "memdev":"mem0",
        "ram_size":"8.00 GiB (8.59 GB)",
        "serial":"0",
        "host":"0000:0d:00.0"
      }
    ]
  },
  {
    "root decoders":[
      {
        "decoder":"decoder0.0",
        "size":"16.00 GiB (17.18 GB)",
        "interleave_ways":1,
        "max_available_extent":"83886.04 TiB (18446744.03 TB)",
        "pmem_capable":true,
        "volatile_capable":true,
        "accelmem_capable":true,
        "nr_targets":1
      }
    ]
  },
  {
    "port decoders":[
      {
        "decoder":"decoder1.0",
        "size":"8.00 GiB (8.59 GB)",
        "interleave_ways":1,
        "region":"region0",
        "nr_targets":1
      }
    ]
  },
```

- Automatically onlined as CPU-less NUMA node due to `MEMORY_HOTPLUG_DEFAULT_ONLINE` :

```
ubuntu@cloudimg:~/.../cxl-index$ numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11
node 0 size: 7943 MB
node 0 free: 7233 MB
node 1 cpus:
node 1 size: 8192 MB
node 1 free: 7164 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
ubuntu@cloudimg:~/.../cxl-index$ free -h
               total        used        free      shared  buff/cache   available
Mem:            15Gi       1.7Gi        14Gi       1.0Mi       344Mi        14Gi
Swap:           15Gi          0B        15Gi
ubuntu@cloudimg:~/.../cxl-index$ lsmem
RANGE                                  SIZE  STATE REMOVABLE   BLOCK
0x0000000000000000-0x000000007fffffff   2G online       yes    0-15
0x0000000100000000-0x000000027fffffff   6G online       yes   32-79
0x0000000b90000000-0x0000000d8fffffff   8G online       yes 370-433

Memory block size:       128M
Total online memory:      16G
Total offline memory:      0B
```

- Being identified as different tiers automatically (yes!) :

```
ubuntu@cloudimg:~/hmsdk$ ls /sys/devices/virtual/memory_tiering/
memory_tier22  memory_tier4  power  uevent
ubuntu@cloudimg:~/hmsdk$ cat /sys/devices/virtual/memory_tiering/memory_tier4/nodelist
0
ubuntu@cloudimg:~/hmsdk$ cat /sys/devices/virtual/memory_tiering/memory_tier22/nodelist
1
ubuntu@cloudimg:~/hmsdk$
```

But one catch : Any memory access forced on CXL region led to message : `qemu-system-x86_64: virtio: bogus descriptor or out of resources` in running VM terminal beyond which VM would become unresponsive/hung, would eventually lead to kernel messages system hung on so and so task for this much time. Would have to kill the qemu process as the only option.

Which brings to : **why no kvm** :

`accel=kvm, -cpu host,pmu=on` supposedly behave in very weird ways when QEMU runs them for CXL : When using the above script with these 2 lines and forcing memory access on the CXL region, `Bad swap entry` messages would fill the screen, even though benchmark should not exceed capacity of CXL at all (like running a 3G or even 1G benchmark on 8G CXL node). When directly executing the executable instead of the python script (which would usually give benchmark not found), would lead to **Illegal instruction (core dumped)** : again needing to kill the vm. Eg : Membind 0 works but membind 1 fails :

```
ubuntu@cloudimg:~/.../experiments$ numactl --membind 0 --cpunodebind 0  /home/ubuntu/epfl-research-intern/cxl-index/bench/apps/bin/gupstoy 1G 0
Not using huge pages
Size: 1073741824 bytes, 1 GB
Hugepage: no
Running with 12 threads
Elapsed time: 36.316666 seconds
Update cnt: 4294967296
Total GUPS: 0.118264 GUPS
GUPS per thread: 0.009855 GUPS
== mbench-start ==
secs: 36.316666
work: 0.118264
work_type: GUPS
alloc_secs: 0.062301
== mbench-end ==                numactl --membind 1 --cpunodebind 0  /home/ubuntu/epfl-research-intern/cxl-index/bench/apps/bin/gupstoy 1G 0
Illegal instruction (core dumped)
ubuntu@cloudimg:~/.../experiments$
```

Sometimes would get other strange kernel errors, again having to kill the vm finally.

Conclusion 2: KVM with QEMU for CXL emulation is probably not a good idea at least with this (Linux 6.12 + Qemu 9.2) combination. (Also mentioned in the Fujitsu doc linked above). And looking at this QEMU Issue, it doesn't seem to be fixed in latest version as well, please correct me on this if you have more info.

What does this imply? : Since we are using TCG, things are pretty slow. And, **no pmu as well** (I think).

About the `qemu-system-x86_64: virtio: bogus descriptor or out of resources` issue, LLM suggested me perhaps it is an I/O issue, and looked like that since `./memeater --membind 1` had worked (although slowly), so suggested me this script :

```
#!/bin/bash

cp /usr/share/OVMF/OVMF_VARS_4M.fd my_ovmf_vars.fd

VAR_MAIN_MEM_SIZE=8
VAR_CXL_MEM_SIZE=8
VAR_CPU_CNT=12
MAX_MEM_SIZE=$((${VAR_MAIN_MEM_SIZE} + ${VAR_CXL_MEM_SIZE} + 20))

qemu-system-x86_64 \
  --nographic \
  -m ${VAR_MAIN_MEM_SIZE}G,maxmem=${MAX_MEM_SIZE}G,slots=8 \
  -machine q35,accel=tcg,cxl=on \
  -smp cpus=$VAR_CPU_CNT \
  -drive if=pflash,format=raw,readonly=on,file=/usr/share/OVMF/OVMF_CODE_4M.fd \
  -drive if=pflash,format=raw,file=my_ovmf_vars.fd \
  -drive file=$1,format=qcow2,if=none,id=disk0 \
  -device ich9-ahci,id=ahci \
  -device ide-hd,drive=disk0,bus=ahci.0 \
  -netdev user,id=net0 \
  -device e1000e,netdev=net0 \
  -object memory-backend-ram,size=${VAR_CXL_MEM_SIZE}G,id=m1,share=on \
  -device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 \
  -device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 \
  -device cxl-type3,bus=root_port13,volatile-memdev=m1,id=cxl-vmem0 \
  -M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=16G
```

Not sure why the firmware lines are needed and if they are really helpful. This allowed me to run `gups 1G --membind node 1` (finally!) but it was terribly slow. Also a more ambitious benchmark (like `gups 3G with 6G hogged`)

Tried today this script directly inspired from the Fujitsu PDF :

```
#!/bin/bash

VAR_MAIN_MEM_SIZE=8
VAR_CXL_MEM_SIZE=8
VAR_CPU_CNT=12
MAX_MEM_SIZE=$((${VAR_MAIN_MEM_SIZE} + ${VAR_CXL_MEM_SIZE} + 20))

qemu-system-x86_64 \
  -drive file=$1,format=qcow2,index=0,media=disk,id=hd \
  -m ${VAR_MAIN_MEM_SIZE}G,maxmem=${MAX_MEM_SIZE}G,slots=8 \
  -machine type=q35,accel=tcg,cxl=on \
  -smp $VAR_CPU_CNT \
  -nographic \
  -device e1000,netdev=net0 \
  -netdev user,id=net0 \
  -object memory-backend-ram,size=${VAR_CXL_MEM_SIZE}G,id=m1,share=on \
  -device pxb-cxl,bus_nr=12,bus=pcie.0,id=cxl.1 \
  -device cxl-rp,port=0,bus=cxl.1,id=root_port13,chassis=0,slot=2 \
  -device cxl-type3,bus=root_port13,volatile-memdev=m1,id=cxl-vmem0 \
  -M cxl-fmw.0.targets.0=cxl.1,cxl-fmw.0.size=16G
```

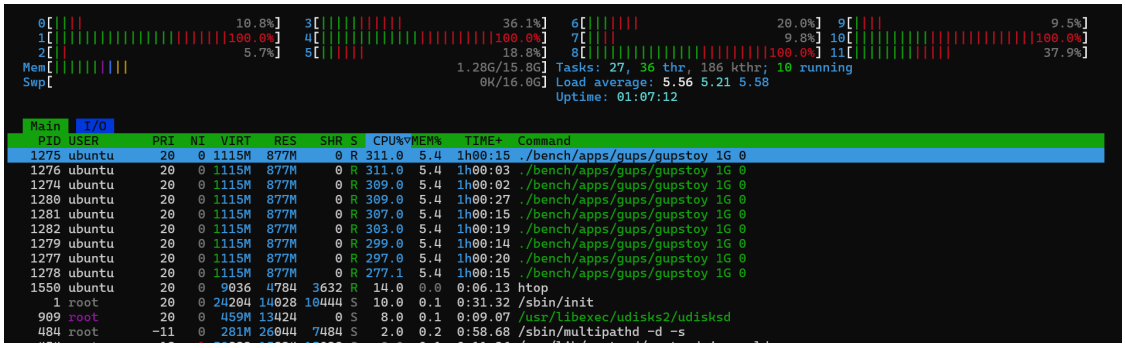
Quite similar to the last once, just without the firmware lines. This one again allows to access CXL memory but terribly slow.

Case in point :

- `numactl --membind 0 ./bench/apps/gups/gupstoy 1G 0` : Completed under 4 mins
- `numactl --membind 1 ./bench/apps/gups/gupstoy 1G 0` : Not completed even after 4 hours (took more than 1 hr just to occupy 1G on CXL)

Illustrative Examples :

- Still loaded just 877M/1G in 1h :



- Able to use 1G on CXL :

```
ubuntu@cloudimg:~/.../cxl-index$ numastat -m

Per-node system memory usage (in MBs):
Token Unaccepted not in hash table.
Token Unaccepted not in hash table.
                          Node 0          Node 1           Total
                 --------------- --------------- ---------------
MemTotal               7943.91         8192.00        16135.91
MemFree                7289.16         7164.46        14453.62
MemUsed                 654.75         1027.54         1682.29
SwapCached                0.00            0.00            0.00
Active                  245.87         1024.54         1270.40
Inactive                 65.43            0.03           65.46
Active(anon)             55.95         1024.20         1080.14
Inactive(anon)            0.00            0.00            0.00
Active(file)            189.92            0.34          190.26
Inactive(file)           65.43            0.03           65.46
Unevictable              25.68            0.00           25.68
Mlocked                  25.68            0.00           25.68
Dirty                     0.00            0.00            0.00
Writeback                 0.00            0.00            0.00
FilePages               263.77            0.37          264.13
```

- *An interesting observation*: `numa_hits` on node 1 increase rapidly when amount of memory loaded in it is increasing (as expected), but more or less **stagnates** once 1G allocation is complete but the benchmark is running, while node0 numa hits increase much more rapidly. Examples :

```
KReclaimable                22.30            0.00
ubuntu@cloudimg:~/.../cxl-index$ numastat
                            node0           node1
numa_hit                   375018          262898
numa_miss                       0               0
numa_foreign                    0               0
interleave_hit               1699               0
local_node                 375018               0
other_node                      0          262898
ubuntu@cloudimg:~/.../cxl-index$ numastat
                            node0           node1
numa_hit                   375211          262898
numa_miss                       0               0
numa_foreign                    0               0
interleave_hit               1699               0
local_node                 375211               0
other_node                      0          262898
ubuntu@cloudimg:~/.../cxl-index$ numastat
                            node0           node1
numa_hit                   375402          262898
numa_miss                       0               0
numa_foreign                    0               0
interleave_hit               1699               0
local_node                 375402               0
other_node                      0          262898
ubuntu@cloudimg:~/.../cxl-index$
```

```
ubuntu@cloudimg:~/.../cxl-index$ numastat
                           node0           node1
numa_hit                  375402          262898
numa_miss                      0               0
numa_foreign                   0               0
interleave_hit              1699               0
local_node                375402               0
other_node                     0          262898
ubuntu@cloudimg:~/.../cxl-index$ htop
ubuntu@cloudimg:~/.../cxl-index$ numastat
                           node0           node1
numa_hit                  428068          262901
numa_miss                      0               0
numa_foreign                   0               0
interleave_hit              1699               0
local_node                428068               0
other_node                     0          262901
ubuntu@cloudimg:~/.../cxl-index$
```

```
ubuntu@cloudimg:~/.../cxl-index$ numastat
                           node0           node1
numa_hit                  478141          262910
numa_miss                      0               0
numa_foreign                   0               0
interleave_hit              1699               0
local_node                478141               0
other_node                     0          262910
ubuntu@cloudimg:~/.../cxl-index$
```

A more ambitious benchmark (in current settings atleast), like hogging large portion on node 0, will make system "very slow" to the point of unresponsive, the benchmark may not complete (or sometimes not even start).

Question : *Can we improve this, speed this up, make it usable?*: Will try to look into it, but no answer currently.

GUPS Benchmark with CXL nodes

DDR RAM : 12 cores and 8G at node 0
CXL : no cores and 8G at node 1

Designed such that max CXL used memory reaches 200M in one of them (and 0 for 1G and 5G without memory hogging):

Committed at `experiments/cxl-attached-gups`:

No HMSDK, only NUMA_balancing (combined with memory hogging for one case) :

NUMA config common for all

```
"numa_balancing": {
    "numa_balancing": "2",
    "demotion_enabled": "true",
    "zone_reclaim_mode": "1",
    "lru_gen_enabled": "0x0000",
    "hot_threshold_ms": "1000",
    "scan_delay_ms": "1000",
    "scan_period_max_ms": "1100",
    "scan_period_min_ms": "1000",
    "scan_size_mb": "1024"
}
```

Table 1: GUPS Benchmark Results

Metric	GUPS 1G	GUPS 5G	GUPS 1G with 6000M memhog	GUPS 7G
numa_hit	285190	1340912	309789	1945183
numa_miss	0	0	5848	37208
numa_foreign	0	0	5848	37208
numa_interleave	0	0	0	0
numa_local	285190	1340912	285244	1887016
numa_other	0	0	30393	95375
pgpromote_success	29	0	5470	45657
pgpromote_candidate	0	0	96815	217138
pgdemote_kswapd	0	0	23700	56494
pgdemote_direct	0	0	0	0
pgdemote_khugepaged	0	0	0	0
pgalloc_dma	0	0	3443	3410
pgalloc_dma32	0	29934	236649	469456
pgalloc_normal	285636	1311514	75976	1510320
pgfault	294634	1350134	400053	2450348
pgmajfault	5	25	24	25
kswapd_low_wmark_hit_quickly	0	0	0	0
kswapd_high_wmark_hit_quickly	0	0	6	15
numa_pte_updates	80	0	102902	522354
numa_huge_pte_updates	0	0	0	0
numa_hint_faults	29	0	101691	513410
numa_hint_faults_local	0	0	0	0
numa_pages_migrated	29	0	5470	45691
pgmigrate_success	29	0	29489	102345
pgmigrate_fail	0	0	521	1506

Some words on HMSDK (Capacity Expansion)

Reference

From what I could grasp, for capacity expansion / memory tiering, HMSDK uses existing DAMON and DAMOS frameworks and adds promotion/demotion actions (integrated in kernel 6.11 onwards, hence no local patch needed). From initial read, it seems to just helping in creating the initial `.yaml` config for `damo`, (in which we have to specify promotion and demotion targets, ie we need not depend on auto detection).

DAMO Usage : doc

Apart from that, I believe we have to interact with damo interface only and in order to tune it (if needed), its advanced options need to be studied.

I have tried a very simple setup currently :

```
$ sudo ~/hmsdk/tools/gen_migpol.py --demote 0 1 --promote 1 0 -o ~/hmsdk.yaml
echo true | sudo tee /sys/kernel/mm/numa/demotion_enabled
sudo mount -t cgroup2 none /sys/fs/cgroup
echo '+memory' | sudo tee /sys/fs/cgroup/cgroup.subtree_control
sudo mkdir -p /sys/fs/cgroup/hmsdk
sudo ~/hmsdk/damo/damo start ~/hmsdk.yaml
```

This starts damo with provided config, config will depend on device connected (different for CXL or memory numa node).

From usage guidelines of hmsdk, it is by default enabled for a cgroup.Running a benchmark:

```
sudo sh -c "echo \$$ > /sys/fs/cgroup/hmsdk/cgroup.procs && exec ./run.py"
```

To see a snapshot of active damo's stats:

```
$ sudo ~/hmsdk/damo/damo show
```

Stop damo with:

```
$ sudo ~/hmsdk/damo/damo stop
```

`damo show` when running a benchmark :

```
ubuntu@cloudimg:~/.../experiments$ sudo ~/hmsdk/damo/damo show
kdamond 0 / context 0 / scheme 1 / target id None / recorded for 2 s from 487935 h 44 m 37.931 s
0   addr [4.000 KiB   , 640.000 KiB) (636.000 KiB) access 0 %   age 12 s
1   addr [1024.000 KiB, 5.117 MiB  ) (4.117 MiB  ) access 50 %  age 2 s
2   addr [5.117 MiB   , 6.148 MiB  ) (1.031 MiB  ) access 40 %  age 2 s
3   addr [6.148 MiB   , 11.422 MiB ) (5.273 MiB  ) access 70 %  age 0 ns
4   addr [11.422 MiB  , 32.531 MiB ) (21.109 MiB ) access 55 %  age 0 ns
5   addr [32.531 MiB  , 33.109 MiB ) (592.000 KiB) access 25 %  age 2 s
6   addr [33.109 MiB  , 38.324 MiB ) (5.215 MiB  ) access 35 %  age 2 s
7   addr [38.324 MiB  , 97.633 MiB ) (59.309 MiB ) access 80 %  age 0 ns
8   addr [97.633 MiB  , 99.840 MiB ) (2.207 MiB  ) access 55 %  age 0 ns
9   addr [99.840 MiB  , 102.047 MiB) (2.207 MiB  ) access 70 %  age 0 ns
10  addr [102.047 MiB , 160.012 MiB) (57.965 MiB ) access 75 %  age 0 ns
11  addr [160.012 MiB , 161.055 MiB) (1.043 MiB  ) access 95 %  age 0 ns
12  addr [161.055 MiB , 161.348 MiB) (300.000 KiB) access 65 %  age 0 ns
13  addr [161.348 MiB , 162.523 MiB) (1.176 MiB  ) access 60 %  age 0 ns
14  addr [162.523 MiB , 228.215 MiB) (65.691 MiB ) access 75 %  age 0 ns
15  addr [228.215 MiB , 257.570 MiB) (29.355 MiB ) access 70 %  age 0 ns
16  addr [257.570 MiB , 270.156 MiB) (12.586 MiB ) access 65 %  age 0 ns
17  addr [270.156 MiB , 275.867 MiB) (5.711 MiB  ) access 85 %  age 0 ns
18  addr [275.867 MiB , 277.918 MiB) (2.051 MiB  ) access 10 %  age 0 ns
19  addr [277.918 MiB , 278.148 MiB) (236.000 KiB) access 0 %   age 2 s
20  addr [278.148 MiB , 283.082 MiB) (4.934 MiB  ) access 60 %  age 4 s
21  addr [283.082 MiB , 283.633 MiB) (564.000 KiB) access 55 %  age 4 s
22  addr [283.633 MiB , 287.289 MiB) (3.656 MiB  ) access 90 %  age 0 ns
23  addr [287.289 MiB , 355.738 MiB) (68.449 MiB ) access 70 %  age 0 ns
24  addr [355.738 MiB , 383.434 MiB) (27.695 MiB ) access 95 %  age 0 ns
25  addr [383.434 MiB , 419.711 MiB) (36.277 MiB ) access 75 %  age 0 ns
26  addr [419.711 MiB , 423.656 MiB) (3.945 MiB  ) access 95 %  age 0 ns
```

Since we are only concerned with how HMSDK affects promotions/demotions only (I believe), hence currently limiting myself to these commands. Apart from them key commands are `damo record` and `damo report <subcommand>` but not using them since (I think) we don't need statistics from DAMON, we only require its `migrate_cold` and `migrate_hot` actions. Also, `damo record` needs perf.

Could use `damo` commands when working with cxl node as well :

```
ubuntu@cloudimg:~/hmsdk$ sudo ./damo/damo show
kdamond 0 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.938 s
0   addr [4.000 KiB   , 640.000 KiB) (636.000 KiB) access 0 %   age 22 s
1   addr [1024.000 KiB, 8.000 MiB  ) (7.000 MiB  ) access 0 %   age 22 s
2   addr [8.031 MiB   , 8.043 MiB  ) (12.000 KiB ) access 0 %   age 22 s
3   addr [8.047 MiB   , 8.066 MiB  ) (20.000 KiB ) access 0 %   age 22 s
4   addr [9.000 MiB   , 1.966 GiB  ) (1.957 GiB  ) access 0 %   age 22 s
5   addr [1.966 GiB   , 1.976 GiB  ) (10.469 MiB ) access 0 %   age 22 s
6   addr [1.977 GiB   , 1.987 GiB  ) (10.926 MiB ) access 0 %   age 22 s
7   addr [1.990 GiB   , 1.998 GiB  ) (8.066 MiB  ) access 0 %   age 22 s
8   addr [1.998 GiB   , 1.999 GiB  ) (536.000 KiB) access 0 %   age 22 s
9   addr [4.000 GiB   , 4.284 GiB  ) (290.344 MiB) access 0 %   age 22 s
10  addr [4.284 GiB   , 4.345 GiB  ) (62.430 MiB ) access 0 %   age 0 ns
11  addr [4.345 GiB   , 4.360 GiB  ) (15.609 MiB ) access 0 %   age 22 s
12  addr [4.360 GiB   , 4.375 GiB  ) (15.391 MiB ) access 0 %   age 0 ns
13  addr [4.375 GiB   , 5.806 GiB  ) (1.431 GiB  ) access 0 %   age 22 s
14  addr [5.806 GiB   , 5.883 GiB  ) (79.613 MiB ) access 0 %   age 24 s
15  addr [5.883 GiB   , 5.946 GiB  ) (64.000 MiB ) access 0 %   age 26 s
16  addr [5.946 GiB   , 8.326 GiB  ) (2.380 GiB  ) access 0 %   age 10 s
17  addr [8.326 GiB   , 8.361 GiB  ) (35.438 MiB ) access 5 %   age 10 s
18  addr [8.361 GiB   , 10.000 GiB ) (1.639 GiB  ) access 0 %   age 10 s
total size: 7.994 GiB
kdamond 1 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.942 s
0   addr [46.250 GiB  , 54.250 GiB ) (8.000 GiB  ) access 0 %   age 58 s
total size: 8.000 GiB
kdamond 0 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.938 s
0   addr [4.000 KiB   , 640.000 KiB) (636.000 KiB) access 0 %   age 22 s
1   addr [1024.000 KiB, 8.000 MiB  ) (7.000 MiB  ) access 0 %   age 22 s
2   addr [8.031 MiB   , 8.043 MiB  ) (12.000 KiB ) access 0 %   age 22 s
3   addr [8.047 MiB   , 8.066 MiB  ) (20.000 KiB ) access 0 %   age 22 s
4   addr [9.000 MiB   , 1.966 GiB  ) (1.957 GiB  ) access 0 %   age 22 s
5   addr [1.966 GiB   , 1.976 GiB  ) (10.469 MiB ) access 0 %   age 22 s
6   addr [1.977 GiB   , 1.987 GiB  ) (10.926 MiB ) access 0 %   age 22 s
7   addr [1.990 GiB   , 1.998 GiB  ) (8.066 MiB  ) access 0 %   age 22 s
```

```
total size: 7.994 GiB
kdamond 1 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.942 s
0   addr [46.250 GiB  , 54.250 GiB ) (8.000 GiB  ) access 0 %   age 58 s
total size: 8.000 GiB
kdamond 0 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.938 s
0   addr [4.000 KiB   , 640.000 KiB) (636.000 KiB) access 0 %   age 22 s
1   addr [1024.000 KiB, 8.000 MiB  ) (7.000 MiB  ) access 0 %   age 22 s
2   addr [8.031 MiB   , 8.043 MiB  ) (12.000 KiB ) access 0 %   age 22 s
3   addr [8.047 MiB   , 8.066 MiB  ) (20.000 KiB ) access 0 %   age 22 s
4   addr [9.000 MiB   , 1.966 GiB  ) (1.957 GiB  ) access 0 %   age 22 s
5   addr [1.966 GiB   , 1.976 GiB  ) (10.469 MiB ) access 0 %   age 22 s
6   addr [1.977 GiB   , 1.987 GiB  ) (10.926 MiB ) access 0 %   age 22 s
7   addr [1.990 GiB   , 1.998 GiB  ) (8.066 MiB  ) access 0 %   age 22 s
8   addr [1.998 GiB   , 1.999 GiB  ) (536.000 KiB) access 0 %   age 22 s
9   addr [4.000 GiB   , 4.284 GiB  ) (290.344 MiB) access 0 %   age 22 s
10  addr [4.284 GiB   , 4.345 GiB  ) (62.430 MiB ) access 0 %   age 0 ns
11  addr [4.345 GiB   , 4.360 GiB  ) (15.609 MiB ) access 0 %   age 22 s
12  addr [4.360 GiB   , 4.375 GiB  ) (15.391 MiB ) access 0 %   age 0 ns
13  addr [4.375 GiB   , 5.806 GiB  ) (1.431 GiB  ) access 0 %   age 22 s
14  addr [5.806 GiB   , 5.883 GiB  ) (79.613 MiB ) access 0 %   age 24 s
15  addr [5.883 GiB   , 5.946 GiB  ) (64.000 MiB ) access 0 %   age 26 s
16  addr [5.946 GiB   , 8.326 GiB  ) (2.380 GiB  ) access 0 %   age 10 s
17  addr [8.326 GiB   , 8.361 GiB  ) (35.438 MiB ) access 5 %   age 10 s
18  addr [8.361 GiB   , 10.000 GiB ) (1.639 GiB  ) access 0 %   age 10 s
total size: 7.994 GiB
kdamond 1 / context 0 / scheme 1 / target id None / recorded for 2 s from 487912 h 39 m 13.942 s
0   addr [46.250 GiB  , 54.250 GiB ) (8.000 GiB  ) access 0 %   age 58 s
total size: 8.000 GiB
ubuntu@cloudimg:~/hmsdk$
```

HMSDK benchmarks (with two NUMA nodes and NOT CXL)

Since we need tiering detected for our NUMA nodes, we should specify some HMAT attributes and then kernel will do rest of the job (no local patch should be needed) : Kernel Patch

Referring this excerpt from QEMU manpage

source and destination are NUMA node IDs. distance is the NUMA distance from source to destination. The distance from a node to itself is always 10. If any pair of nodes is given a distance, then all pairs must be given distances. Although, when distances are only given in one direction for each pair of nodes, then the distances in the opposite directions are assumed to be the same. If, however, an asymmetrical pair of distances is given for even one node pair, then all node pairs must be provided distance values for both directions, even when they are symmetrical. When a node is unreachable from another node, set the pair's distance to 255.

Note that the `-numa` option doesn't allocate any of the specified resources, it just assigns existing resources to NUMA nodes. This means that one still has to use the `-m`, `-smp` options to allocate RAM and VCPUs respectively.

Use 'hmat-lb' to set System Locality Latency and Bandwidth Information between initiator and target NUMA nodes in ACPI Heterogeneous Attribute Memory Table (HMAT). Initiator NUMA node can create memory requests, usually it has one or more processors. Target NUMA node contains addressable memory.

In 'hmat-lb' option, node are NUMA node IDs. hierarchy is the memory hierarchy of the target NUMA node: if hierarchy is 'memory', the structure represents the memory performance; if hierarchy is 'first-level|second-level|third-level', this structure represents aggregated performance of memory side caches for each domain. type of 'data-type' is type of data represented by this structure instance: if 'hierarchy' is 'memory', 'data-type' is 'access|read|write' latency or 'access|read|write' bandwidth of the target memory; if 'hierarchy' is 'first-level|second-level|third-level', 'data-type' is 'access|read|write' hit latency or 'access|read|write' hit bandwidth of the target memory side cache.

lat is latency value in nanoseconds. bw is bandwidth value, the possible value and units are NUM[M|G|T], mean that the bandwidth value are NUM byte per second (or MB/s, GB/s or TB/s depending on used suffix). Note that if latency or bandwidth value is 0, means the corresponding latency or bandwidth information is not provided.

In 'hmat-cache' option, node-id is the NUMA-id of the memory belongs. size is the size of memory side cache in bytes. level is the cache level described in this structure, note that the cache level 0 should not be used with 'hmat-cache' option. associativity is the cache associativity, the possible value is 'none/direct(direct-mapped)/complex(complex cache indexing)'. policy is the write policy. line is the cache Line size in bytes.

For example, the following options describe 2 NUMA nodes. Node 0 has 2 cpus and a ram, node 1 has only a ram. The processors in node 0 access memory in node 0 with access-latency 5 nanoseconds, access-bandwidth is 200 MB/s; The processors in NUMA node 0 access memory in NUMA node 1 with access-latency 10 nanoseconds, access-bandwidth is 100 MB/s. And for memory side cache information, NUMA node 0 and 1 both have 1 level memory cache, size is 10KB, policy is write-back, the cache Line size is 8 bytes:

```
-machine hmat=on \
-m 2G \
-object memory-backend-ram,size=1G,id=m0 \
-object memory-backend-ram,size=1G,id=m1 \
-smp 2,sockets=2,maxcpus=2 \
-numa node,nodeid=0,memdev=m0 \
-numa node,nodeid=1,memdev=m1,initiator=0 \
-numa cpu,node-id=0,socket-id=0 \
-numa cpu,node-id=0,socket-id=1 \
-numa hmat-lb,initiator=0,target=0,hierarchy=memory,data-type=access-latency,latency=5 \
-numa hmat-lb,initiator=0,target=0,hierarchy=memory,data-type=access-bandwidth,bandwidth=200M \
-numa hmat-lb,initiator=0,target=1,hierarchy=memory,data-type=access-latency,latency=10 \
-numa hmat-lb,initiator=0,target=1,hierarchy=memory,data-type=access-bandwidth,bandwidth=100M \
-numa hmat-cache,node-id=0,size=10K,level=1,associativity=direct,policy=write-back,line=8 \
-numa hmat-cache,node-id=1,size=10K,level=1,associativity=direct,policy=write-back,line=8
```

Ran with this qemu config :

```
#!/bin/bash

VAR_FAST_MEM_SIZE=8
VAR_SLOW_MEM_SIZE=8
VAR_TOTAL_MEM=$((${VAR_FAST_MEM_SIZE} + ${VAR_SLOW_MEM_SIZE}))
VAR_CPU_CNT=12

qemu-system-x86_64 \
    --enable-kvm \
    --nographic \
    -machine q35,hmat=on \
    -m ${VAR_TOTAL_MEM}G \
    -cpu host,pmu=on \
    -smp ${VAR_CPU_CNT} \
    -drive file=$1,index=0,format=qcow2,if=virtio \
    -object memory-backend-ram,size=${VAR_FAST_MEM_SIZE}G,id=m0 \
    -object memory-backend-ram,size=${VAR_SLOW_MEM_SIZE}G,id=m1 \
    -numa node,nodeid=0,cpus=0-$((VAR_CPU_CNT-1)),memdev=m0 \
    -numa node,nodeid=1,memdev=m1,initiator=0 \
    -numa hmat-lb,initiator=0,target=0,hierarchy=memory,data-type=access-latency,latency=112 \
    -numa hmat-lb,initiator=0,target=0,hierarchy=memory,data-type=access-bandwidth,bandwidth=271G \
```

```
    -numa hmat-lb,initiator=0,target=1,hierarchy=memory,data-type=access-latency,latency=237 \
    -numa hmat-lb,initiator=0,target=1,hierarchy=memory,data-type=access-bandwidth,bandwidth=46G \
    -netdev user,id=net0,hostfwd=tcp::2222-:22 \
    -device virtio-net-pci,netdev=net0 \
    -monitor none \
    -serial stdio
```

Resulting in this:

```
ubuntu@cloudimg:~/.../experiments$ ls /sys/devices/virtual/memory_tiering/
memory_tier4  memory_tier56  power  uevent
ubuntu@cloudimg:~/.../experiments$
```

Latency and bandwidth values taken from [1].

Afaik, access-latency, access-bw, specify both read/write. Though they can be specified explicitly if needed also. Also since already specified lat, bw hence not specifying dist between nodes.

Committed at `experiments/hmsdk_damon_numa`:

Ran GUPS for 9G for 4 cases :

- hmsdk/damo enabled but numa_balancing disabled (demotion_enabled=true though as per hmsdk usage guidelines)
- numa_balancing enabled but damo stopped
- both hmsdk/damo and numa_balancing enabled
- neither hmsdk/damo nor numa_balancing enabled
- both hmsdk/damo and numa_balancing enabled with 2G hogged on node 0

NOTE: Not sure how to interpret results with hmsdk enabled, the values are strange. Like, with numa-balancing enabled, there are pgpromote_candidate but no pgpromote, this is weird.

NOTE2: Not sure how to get number of migrate_hot and migrate_cold actions performed by DAMOS (looked it up but didn't find some concrete answer). Like, damo status shows cumulative values afaik. (One option can be to stop and start after each benchmark). Don't know if damo report is helpful in this regard or not.

To give an idea of latency wrt CXL, GUPS 7G in previous case took more than 20mins, where CXL occupied was around 200M. Here GUPS 9G took around 2 mins, even though 2.2G of node 1 were occupied.

Bottomline:Perhaps, considering the behaviour of scripts, we may say CXL emulation scripts are good for understanding functional behaviour of hardware but for the purpose of benchmarking, tiered DRAM NUMA nodes look like the way to go.

PS: You can boot the VM yourself. You can find it at my account at : /home/roopam/epfl-research-intern/qemu-vms/epfl-vm.img. The three scripts used are `cxl-vm.sh`, `new-hope.sh` and `numa-tiered.sh`. Inside the VM, please select Linux 6.12+ (it's also the default option in the grub menu).

Login : ubuntu Password : roopam1234

PS2 : Powering off VM writes some garbage on terminal sometimes. So better idea to sudo poweroff and start again instead of sudo reboot.

References

[1] Musa Unal et al. "Tolerate It if You Cannot Reduce It: Handling Latency in Tiered Memory". In: *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*. 2025, pp. 50–57.