QAT Device Plugin

Goal

Intel® QuickAssist Technology offered with Intel processor-based platforms supports application developer to accelerate workloads such as cryptography and data compression, etc.

This Wiki describes the steps on how to deploy QAT device plugin in edge location node and how to use QAT device in ICN.

Implement details

Intel QAT device plugin is Kubernetes device plugin for discovering and advertising QAT virtual functions (VFs) in a Kubernetes host. it supports a platform that pairs the Intel®
C62x Chipset (also known as Platform Controller Hub, or PCH) with Intel® Xeon®Processor D Family System-on-a Chip (SoC) or Intel Atom® C3000 Processor Product
Family SoC.

Precondition:

(1) QAT devices installed in the system

(2) Enable grub with parameter "intel_iommu=on iommu=pt"

To automatically deploy QAT device plugin in ICN, KuD addon QAT device plugin includes:

Installation script:

(1) Install and probe QAT kernel driver (e.g. intel_qat.ko, qat_c62x.ko, qat_d15xx.ko, qat_c3xxx.ko etc.)

(2) Install adf_ctl utility

(3) Install device configuration files and firmware files

(4) Start the qat_service, which inserts the appropriate modules as required, enable virtual functions (VFs) and runs adf_ctl to bring up the devices.

(5) Set up the qat_service to run on future boots

Daemon set yaml:

(1) qat_plugin_default_configmap.yaml: default configuration to discovery and advertise QAT devices

(2) qat_plugin.yaml: deploy QAT device plugin as daemon set in edge location node. the QAT device plugin will leverage the configuration in qat_plugin_default_configmap.yaml to discovery and advertise QAT VFs to K8s.

Use QAT device plugin in ICN

(1) Create docker image crypto-perf:devel based on the instructions in https://github.com/intel/intel-device-plugins-for-kubernetes/blob/master/cmd/qat_plugin/README.md

(2) Create test_qat.yaml with below contents:

```
kind: Pod
apiVersion: v1
metadata:
   name: qat_pod
spec:
   containers:
   - name: dpdkcontainer
     image: crypto-perf:devel
     imagePullPolicy: IfNotPresent
     command: [ "/bin/bash", "-c", "--" ]
     args: [ "while true; do sleep 300000; done;" ]
     volumeMounts:
     - mountPath: /dev/hugepages
       name: hugepage
     resources:
        requests:
           cpu: "3"
           memory: "1Gi"
           qat.intel.com/generic: '2'
           hugepages-2Mi: "1Gi"
        limits:
           cpu: "3"
           memory: "1Gi"
           qat.intel.com/generic: '2'
           hugepages-2Mi: "1Gi"
    securityContext:
       capabilities:
         add:
             ["IPC_LOCK"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

(3) Create pod to consume QAT devices

```
cat test_qat.yaml  | kubectl apply -f -
```

(4) Verify QAT workload in created Pod

```
kubectl exec -it qat_pod bash

./dpdk-test-crypto-perf -l 6-7 -w $QAT1 -- --ptest throughput --devtype crypto_qat --optype cipher-only --cipher-algo aes-cbc --cipher-op encrypt --cipher-key-sz 16 --total-ops 10000000 --burst-sz 32 --buffer-sz 64
```

The output will be similar as below:

```
EAL: Detected 72 lcore(s)
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: VFIO support initialized
EAL: Cannot obtain physical addresses: Success. Only vfio will function.
EAL: PCI device 0000:3f:02.5 on NUMA socket 0
EAL: probe driver: 8086:37c9 crypto_qat
EAL: using IOMMU type 1 (Type 1)
# Crypto Performance Application Options:
#
# cperf test: throughput
#
# size of crypto op / mbuf pool: 8192
# total number of ops: 10000000
# buffer sizes: 64
# burst sizes: 32

# segments per buffer: 1
#
# cryptodev type: crypto_qat
#
# crypto operation: cipher-only
# sessionless: no
# out of place: no
#
# cipher algorithm: aes-cbc
# cipher operation: encrypt
# cipher key size: 16
# cipher iv size: 16
#
Allocated session pool on socket 0

plaintext =
0x71, 0x75, 0x83, 0x98, 0x75, 0x42, 0x51, 0x09, 0x94, 0x02, 0x13, 0x20, 0x15, 0x64, 0x46, 0x32, 0x08, 0x18, 0x91, 0x82, 0x86, 0x52, 0x23,
0x93, 0x44, 0x54, 0x28, 0x68, 0x78, 0x78, 0x70, 0x06,
0x42, 0x74, 0x41, 0x27, 0x73, 0x38, 0x53, 0x77, 0x51, 0x96, 0x53, 0x24, 0x03, 0x88, 0x74, 0x14, 0x70, 0x23, 0x88, 0x30, 0x85, 0x18, 0x89,
0x27, 0x41, 0x71, 0x61, 0x23, 0x04, 0x83, 0x30, 0x57

cipher_key =
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f

cipher_iv =
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f

ciphertext =
0xe2, 0x19, 0x24, 0x56, 0x13, 0x59, 0xa6, 0x5d, 0xdf, 0xd0, 0x72, 0xaa, 0x23, 0xc7, 0x36, 0x3a, 0xbb, 0x3e, 0x8b, 0x64, 0xd5, 0xbf, 0xde,
0x65, 0xa2, 0x75, 0xd9, 0x45, 0x6c, 0x3c, 0xd2, 0x6a,
0xc7, 0xd0, 0x9a, 0xd0, 0x87, 0xb8, 0xe4, 0x94, 0x11, 0x62, 0x5a, 0xc3, 0xc3, 0x01, 0xa3, 0x86, 0xbc, 0xbc, 0x9c, 0xc0, 0x81, 0x9f, 0xbf, 0x5c,
0x6f, 0x3f, 0x13, 0xf1, 0xae, 0xcf, 0x26, 0xb3
lcore id Buf Size Burst Size Enqueued Dequeued Failed Enq Failed Deq MOps Gbps Cycles/Buf

7 64 32 10000000 10000000 11076837 9786964 6.2719 3.2112 365.86
```

References:

https://github.com/intel/intel-device-plugins-for-kubernetes

https://01.org/zh/intel-quick-assist-technology