cuBB on NVIDIA Cloud Native Stack

Table of contents

NVIDIA Cloud Native Stack (formerly known as Cloud Native Core) is a collection of software that runs cloud native workloads on NVIDIA GPUs. This section describes how to install and run the Aerial cuBB software examples on NVIDIA Cloud Native Stack and related components to run Aerial cuBB.

Installation of NVIDIA Cloud Native Stack

Prerequisite: The server must already have the OS, NVIDIA Driver, and other configuration as described in *Installing Tools on Aerial Devkit* or *Installing Tools on Dell R750*.

The steps to install NVIDIA Cloud Native Stack follows the NVIDIA Cloud Native Stack v9.3 installation guide on GitHub, starting with section “Installing Container Runtime”, with the following additional notes:

- Select containerd when given the choice between containerd or CRI-O in the install guide.
- For running an ru-emulator on a server without a GPU, it is necessary to remove/comment out the “BinaryName” field from /etc/containerd/config.toml on that server.

If this step is not done, an ru-emulator failed to start error message can occur

> State: Terminated Reason: StartError Message: failed to create containerd task: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: error during container init: error running hook *#0: error running hook: exit status 1, stdout: , stderr: Auto-detected mode as 'legacy'* nvidia-container-cli: initialization error: nvml error: driver not loaded: unknown Exit Code: 128 Started: Thu, 01 Jan 1970 00:00:00 +0000 Finished: Wed, 17 Jan 2024 05:25:27 +0000

- Enable k8s CPU Manager, Topology Manager, and Memory Manager.

1. Update each worker node’s /var/lib/kubelet/config.yaml. The file to use depends on the server type.

 # For Aerial Devkit servers $ cat <<EOF | sudo tee -a /var/lib/kubelet/config.yaml # Additional Configuration # Feature Gates featureGates: MemoryManager: true # CPU Manager Configuration cpuManagerPolicy: "static" cpuManagerPolicyOptions: full-pcpus-only: "true" reservedSystemCPUs: 0-2,22-23 # Topology Manager Configuration topologyManagerPolicy: "restricted" topologyManagerScope: "container" # Memory Manager Configuration memoryManagerPolicy: "Static" reservedMemory: - numaNode: 0 limits: memory: 100Mi EOF *# for Dell R750 servers* $ cat <<EOF | sudo tee -a /var/lib/kubelet/config.yaml # Additional Configuration # Feature Gates featureGates: MemoryManager: true # CPU Manager Configuration cpuManagerPolicy: "static" cpuManagerPolicyOptions: full-pcpus-only: "true" reservedSystemCPUs: 0-3 # Topology Manager Configuration topologyManagerPolicy: "restricted" topologyManagerScope: "pod" # Memory Manager Configuration memoryManagerPolicy: "Static" reservedMemory: - numaNode: 0 limits: memory: 50Mi - numaNode: 1 limits: memory: 50Mi EOF

2. Drain each worker node.

 # Run from k8s master or other server where you have the kube config kubectl drain $nodeName --force --ignore-daemonsets

3. Restart each worker node’s kubelet.

 # Run on worker node sudo systemctl stop kubelet sudo rm -f /var/lib/kubelet/cpu_manager_state sudo rm -f /var/lib/kubelet/memory_manager_state sudo systemctl start kubelet sudo systemctl status kubelet

4. Confirm kubelet status, verify that it is healthy.

$ systemctl status kubelet kubelet.service - kubelet: The Kubernetes Node Agent Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset:

```
enabled) Drop-In: /etc/systemd/system/kubelet.service.d      10-kubeadm.conf
Active: active (running) since Thu 2023-10-12 19:36:05 UTC; 5s ago
```

5. Uncordon the node.

```
# Run from k8s master or other server where you have the kube config kubectl
uncordon $nodeName
```

6. Setup a registry secret called “regcred” to be able to pull from $YourPrivateRegistry container registry. Follow the procedure described in the Kubernetes documentation. If you are using $YourPrivateRegistry=nvcr.io, please remember to generate an API Key from the NGC API-Key Setup Portal if you don’t already have one.

Building Aerial Binary Container

This section describes how to build an Aerial binary container for the cuphycontroller_scf, test_mac, and ru_emulator applications, along with some example scenario test vectors.

1. Extract the build script.

```
mkdir -p cuPHY-CP/ docker pull nvcr.io/ea_aerial_sdk/aerial:24-1-cubb docker run --
rm -u `id -u` -v .:/staging nvcr.io/ea_aerial_sdk/aerial:24-1-cubb cp -a
/opt/nvidia/cuBB/cuPHY-CP/container /staging/cuPHY-CP/
```

Install dependencies

```
sudo apt update sudo apt install python3-pip -y pip3 install hpccm
```

2. Build the binary container and push to your private container repository.

```
AERIAL_BUILD_IMAGE=nvcr.io/ea_aerial_sdk/aerial:24-1-cubb
AERIAL_RELEASE_REPO=$YourPrivateRepo/
AERIAL_RELEASE_VERSION_TAG=$YourTag ./cuPHY-CP/container/build_binary.sh
docker push $YourPrivateRepo/aerial_binary:$YourTag-amd64
```

Deploying Binary Container using Helm Chart

Configure the NGC cli tool - follow the steps in https://ngc.nvidia.com/setup/installers/cli

Login to NGC

```
$ ngc config set
```

2. Fetch the Helm Chart from NGC.

```
ngc registry chart pull ea_aerial_sdk/aerial-l1 ngc registry chart pull ea_aerial_sdk/aerial-ru-emulator
```

3. Create value overload files specific to your environment. You must change the following values:
    - YourRUEmulatorNodeName
    - YourPrivateRepo
    - YourTag
    - <MAC Address of DU's FH Port>
    - YourDUNodeName

```
$ cat <<EOF | tee override-ru-emulator-binary.yaml # Deployment customization extraSpec: nodeName: "YourRUEmulatorNodeName" image: repository: YourPrivateRepo/ name: aerial_binary pullPolicy: Always # Overrides the image tag whose default is the chart appVersion. tag: "YourTag" peerethaddr: "<MAC Address of DU's FH Port>" # Spacing is critical below extraSetup: | \`# ru-emulator extra setup\` sed -i "s/enable_beam_forming:.*/enable_beam_forming: 1/" ../cuPHY-CP/ru-emulator/config/config_dyn.yaml \`# Configure NIC PCIe Address\` sed -i "s/nic_interface.*/nic_interface: 0000:3b:00.0/" ../cuPHY-CP/ru-emulator/config/config_dyn.yaml EOF $ cat <<EOF | tee override-l1-binary.yaml #
```

```
Deployment customization extraSpec: nodeName: "YourDUNodeName" image: repository: YourPrivateRepo/ name: aerial_binary pullPolicy: Always # Overrides the image tag whose default is the chart appVersion. tag: "YourTag" enableTestMACContainer: 1 # Spacing is critical below extraSetup: | \`# Aerial L1 extra setup\` \`# Launch pattern related configuration\` sed -i "s/cell_group_num: .*/cell_group_num: 16/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; sed -i "s/pusch_nMaxPrb: .*/pusch_nMaxPrb: 136/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; \`# 3GPP conformance\` sed -i "s/pusch_tdi:.*/pusch_tdi: 1/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; sed -i "s/pusch_cfo:.*/pusch_cfo: 1/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; sed -i "s/pusch_to:.*/pusch_to: 1/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; sed -i "s/puxch_polarDcdrListSz:.*/puxch_polarDcdrListSz: 8/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; \`# Configure NIC PCIe Address\` sed -i "s/ nic:.*/ nic: 0000:cc:00.1/" ../cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml; # Spacing is critical below extraTestMACSetup: | \`# testMAC extra setup\` \`#sed -i "s/test_slots: 0/test_slots: 100000/" ../cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml;\` sed -i "s/schedule_total_time: 0/schedule_total_time: 470000/" ../cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml; sed -i "s/fapi_delay_bit_mask: 0/fapi_delay_bit_mask: 0xF/" ../cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml; sed -i "s/builder_thread_enable: 0/builder_thread_enable: 1/" ../cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml; EOF
```

4. Deploy the Helm Chart.

```
helm install aerial-ru-emulator-test aerial-ru-emulator-0.20234.0.tgz -f override-ru-emulator-binary.yaml helm install aerial-l1-test aerial-l1-0.20234.0.tgz -f override-l1-binary.yaml
```

5. View the logs for each container.

```
# Run in separate windows kubectl logs aerial-l1-test -f kubectl logs aerial-l1-test -c aerial-testmac-ctr -f kubectl logs aerial-ru-emulator-test -f
```

6. Remove the Helm Chart and destroy the pods when finished.

```
helm uninstall aerial-l1-test helm uninstall aerial-ru-emulator-test
```

Theory of Operation

At pod deployment time, k8s dynamically assigns dedicated CPU cores to the Aerial L1 cuphycontroller_scf container and the testMAC container (if it is deployed). When the container starts up, the $cuBB_SDK/cubb_scripts/autoconfig/auto_assign_cores.py script runs to map the k8s-assigned cores to the various Aerial functions. The following template configuration YAML files are used by the auto_assign_cores.py script:

- $cuBB_SDK/cuPHY-CP/cuphycontroller/config/cuphycontroller_$configL1.yaml -> $cuBB_SDK/cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml
- $cuBB_SDK/cuPHY-CP/cuphycontroller/config/$l2adapter_filename -> $cuBB_SDK/cuPHY-CP/cuphycontroller/config/l2_adapter_dyncore.yaml
- $cuBB_SDK/cuPHY-CP/testMAC/testMAC/$configMAC -> $cuBB_SDK/cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml

The variables used above come from:

- $cuBB_SDK: Environment variable defined in container
- $configL1: Helm chart aerial-l1/values.yaml (or override-l1-binary.yaml if overridden) variable ‘configL1’
- $l2adapter_filename: YAML configuration parameter ‘l2adapter_filename’ defined in the template cuphycontroller configuration yaml.
- $configMAC: Helm chart aerial-l1/values.yaml (or override-l1-binary.yaml if overridden) variable ‘configMAC’

An example run of the auto_assign_cores.py script for the aerial-l1-ctr container is:

Detected HT Enabled Detected Multiple NUMA Nodes: [0, 1]. Will use node 1 for scheduling. OS core affinity: [5, 7, 9, 11, 13, 15, 17, 53, 55, 57, 59, 61, 63, 65] OS core affinity for numa node 1: [5, 7, 9, 11, 13, 15, 17, 53, 55, 57, 59, 61, 63, 65] OS isolated cores: [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95] Tentative primary cores: [5, 7, 9, 11, 13, 15, 17] Cuphycontroller core assignment strategy for HT enabled: * 1 low priority primary core (shared with dpdk EAL), HT sibling for h2d_copy thread * {args.workers_ul_count} UL worker primary cores, HT siblings idle * {args.workers_dl_count} DL worker primary cores, HT siblings idle * 1 L2A timer thread primary core, HT sibling for L2A msg processing thread Need 7 physical cores (plus 0 reserved), potential affinity for 7 isolated physical cores +-------------+--+-------------+--+ | Primary | Primary Core | Sibling | Sibling Core | | Core Number | Uses | Core Number | Uses | +-------------+--+-------------+--+ | 5 | low priority threads (inc. DPDK EAL) | 53 | H2D copy | +-------------+--+-------------+--+ | 7 | UL Worker | 55 | [idle] | +-------------+--+-------------+--+ | 9 | UL Worker | 57 | [idle] | +-------------+--+-------------+--+ | 11 | DL Worker | 59 | [idle] | +-------------+--+-------------+--+ | 13 | DL Worker | 61 | [idle] | +-------------+--+-------------+--+ | 15 | DL Worker | 63 | [idle] | +-------------+--+-------------+--+ | 17 | L2A timer | 65 | L2A msg processing | +-------------+--+-------------+--+ Parsing cuphycontroller configuration template: /opt/nvidia/cuBB/cuPHY-CP/cuphycontroller/config/cuphycontroller_F08_R750.yaml Writing cuphycontroller configuration: /opt/nvidia/cuBB/cuPHY-CP/cuphycontroller/config/cuphycontroller_dyncore.yaml Parsing l2adapter configuration template: /opt/nvidia/cuBB/cuPHY-CP/cuphycontroller/config/l2_adapter_config_F08_R750.yaml Writing l2adapter configuration: /opt/nvidia/cuBB/cuPHY-CP/cuphycontroller/config/l2_adapter_config_dyncore.yaml

An example run of the auto_assign_cores.py script for the aerial-l1-ctr container is:

> Detected HT Enabled Detected Multiple NUMA Nodes: [0, 1]. Will use node 1 for scheduling. OS core affinity: [19, 21, 23, 67, 69, 71] OS core affinity for numa node 1: [19, 21, 23, 67, 69, 71] OS isolated cores: [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95] Tentative primary cores: [19, 21, 23] testMAC core assignment strategy: * 1 low priority primary core, HT sibling idle * 1 mac_recv thread primary core, HT sibling idle * 1 builder thread primary core, HT sibling idle Need 3 physical cores (plus 0 reserved), potential affinity for 3 isolated physical cores +-------------+--+-------------+--+ | Primary | Primary Core | Sibling | Sibling Core | | Core Number | Uses | Core Number | Uses | +-------------+--+-------------+--+ | 19 | [testmac] low priority threads | 67 | [idle] | +-------------+--+-------------+--+ | 21 | [testmac] recv | 69 | [idle] | +-------------+--+-------------+--+ | 23 | [testmac] builder | 71 | [idle] | +-------------+--+-------------+--+ Parsing testmac configuration template: /opt/nvidia/cuBB/cuPHY-CP/testMAC/testMAC/test_mac_config.yaml Writing testmac configuration: /opt/nvidia/cuBB/cuPHY-CP/testMAC/testMAC/test_mac_config_dyncore.yaml