Documentación - Tutorial

Descripción:

En este tutorial se documentan aspectos técnicos acerca del flujo de trabajo seguido para lograr implementar la imagen de Linux a la medida con Yocto Project para correr una aplicación desarrollada, capaz de ser ejecutada en un amaquina virtual corriendo sobre una arquitectura x86, partiendo de una imagen mínima de Yocto Project.

Computador Host

El proyecto fue desarrollado en un computador host con las siguientes características:

12 cores con 12 Gbytes de RAM con sistema Operativo Ubuntu y 120 Gbytes de almacenamiento.

Características de los toolkits y herramientas utilizados

Yocto Project: versión scarthgap.

90 Gbytes o más de espacio libre de memoria en disco, al menos 8 Gbytes de memoria RAM, sin embargo se recomienda tener tantos Gbytes de memoria RAM como cores tenga la computadora hos para mejorar el proceso. Tener un sistema operativo soportado (Fedora, openSUSE, CentOS, Debian, o Ubuntu) [1].

Ciertas dependencias de software:

Git 1.8.3.1+, tar 1.28+, Python 3.8.0+, gcc 8.0+, GNU make 4.0+.

Docker

Este toolkit fue utilizado como paso intermedio, previamente a la construccion de la imagen en Yocto Project para tener un escapsulado en el que se instalaran todas las dependencias necesarias y verificar el funcionamiento de la aplicación. La versión utilizada fue 28.0.2.

OpenVINO

La versión implementada en la aplicación sigue la versión 2023.3 con la guia oficial encontrada en [2].

Gstreamer

Dentro del archivo de configuración de Yocto Project fueron agregados plugins de Gstreamer bajo la versión 1.22.12.

Virtual Box

La imagen a medida finalmente se corrió en una máquina virtual usando VirtualBox.

Otros

Dentro de los toolkits necesarios para la aplicación está presente python3, mumpy y pillow.

Descripción flujo de trabajo de los toolkits utilizados

Se revisó el ejemplo proporcionado en el taller de OpenVINO como referencia y punto de partida. Con estos recursos se tomaron como referencia y se generó un contenedor Docker. Se desarrolló un Dockerfile donde se describieron todas las dependencias y requerimientos tanto a nivel de software/toolkits así como del modelo a probar.

Posteriormente se hizo la configuración inicial del entorno de Yocto Project y la verificación de sus requerimientos como se mencionó anteriormente. Se añadieron las layers correspondientes a OpenVINO, luego se modificó el archivo local.config para agregar Gstreamer.

Se hizo una layer personalizada para agregar la aplicación a probar. Se intentó agregar una capa personalizada para agregar los elementos específicar de dlstreamer que no incluye gstreamer por defecto, sin embargo, este paso no fue éxitoso.

Tras varios procesos de compilación se probó la correcta instalación tanto de OpenVINO, Gstreamer, la layer personalizada y se probó la aplicación con una modificación para que no dependiera de los elementos de dlstreamer (gvaclasify y gvawaterark).

Finalmente, se utilizí VirtualBox y se cargó la imagen de linux para correrlo en una maquina virtual.

Selección de la aplicación

La aplicación escogida para probar en la imagen de Linux se basó en el modelo llamado 'human-pose-estimation' a partir de un programa en python.

```
import sys
from utils.gst_utils import gst_launch

# Configuración
DEVICE = "AUTO"
MODELS_PATH = "ruta al modelo"
MODELS_PROC_PATH = "ruta al model_proc"
MODEL_1 = "nombre del modelo"
HPE_MODEL_PROC = f"{MODELS_PROC_PATH}/{MODEL_1}.json"
HPE_MODEL = f"{MODELS_PATH}/intel/{MODEL_1}/FP32/{MODEL_1}.xml"
INPUT = "video fuente"

# Pipeline GStreamer
pipeline_str = (
    f'urisourcebin buffer-size=4096 uri={INPUT} ! '
    f'decodebin ! '
    #f'v4l2src !'
    #f'videoconvert !'
    f'gvaclassify model={HPE_MODEL} model-proc={HPE_MODEL_PROC} device={DEVICE} inference-
        region=full-frame ! queue ! '
    f'gvawatermark ! videoconvert ! autovideosink'
)
# Ejecutar la pipeline
print(f"Ejecutando pipeline de estimación de pose humana...")
gst_launch(pipeline_str)
```

Esta aplicación tiene dos partes principales: descripción de las rutas a buscar para encontrar todos los archivos necesarios y el pipeline a ejecutar por gstreamer. Como se puede observar, esta aplicación busca el nombre del modelo que se le diga, en este caso: human-pose-estimation.

Lista de dependencias

Se requiere clonar los siguientes paquetes de software contenidos en repositorios: Meta-poky, mental-intel, meta-openmebeded, meta-networking, layer personalizada y se debe incluir por supuesto una versión de python. En cuanto a gstreamer, se hizo append de los siguientes elementos, dentro del archivo loca.conf:

```
IMAGE_INSTALL:append = " \
   gstreamer1.0 \
   gstreamer1.0-plugins-base \
   gstreamer1.0-plugins-good \
   gstreamer1.0-plugins-bad \
   gstreamer1.0-libav \
   gstreamer1.0-python \
   gstreamer1.0-rtsp-server \
   gstreamer1.0-meta-base \
"
```

```
LICENSE_FLAGS_ACCEPTED += "commercial"
```

Donde esta última línea es la aceptación de licencia.

En cuanto a dependencias directas de python y openssh para la conexión ssh, se tiene:

```
IMAGE_INSTALL:append = " openssh \
    python3-pillow \
    python3-matplotlib \
    python3-opencv \
    python3-numpy \
    python3-dev \
"
```

Mapeo de dependencias

Las layers utilizadas son:

1. meta-poky y core: proporcionan el sistema base de Linux, incluyen herramientas esenciales del sistema y bibliotecas base, además de que proveen el framework BitBake para la construcción de imágenes.
2. meta-intel: proporciona OpenVINO y sus componentes.
3. meta-openembedded/meta-oe: incluye dependencias generales para multimedia e IA y ofrece bibliotecas de utilidad necesarias para procesamiento de imagen.
4. meta-openembedded/meta-python: proporciona módulos Python adicionales como numpy.
5. meta-openembedded/meta-multimedia: bibliotecas adicionales para GStreamer y herramientas de procesamiento multimedia.

6)meta-networking: dispone de los servicios de conexión ssh.

7. meta-myapp: contiene la aplicación principal de detectión de pose humana, el video fuente y el modelo preentrenado.

GStreamer (de poky/meta) utiliza OpenCV (de meta-oe), la capa que se intentó añadir dlstreamer (de meta-dlstreamer) depende de OpenVINO (de meta-intel). Y la aplicación principal (de meta-myapp) depende de todos los componentes anteriores.

Las layers que vienen dadas por un repositorio de github son integradas de la siguiente manera: clonar el repositorio, cambiarse al brnahc de la versión de Yocto correspondiente y con bitbake se puede agregar la layer automáticamente:

```
git clone https://git.yoctoproject.org/meta-intel

cd meta-intel; git fetch; git check out scarthgap
```

Esto clona el repo para tenerlo a disposición, para agregar dichos repositorios (los cuales ya contienen la estructura y archivos para ser reconocidos como layer dentro de Yocto) se ejecuta el siguiente comando para que bitbake lo agregue:

```
bitbake-layers add-layers <name_layer>
```

Recetas de Yocto:

myapp_1.0.bb: esta es la receta de la capa personalizada que se realizó.

```
SUMMARY = "Human pose estimation app with OpenVINO"
LICENSE = "CLOSED"
SRC_URI = "file://run_pose.py \
           file://utils \
           file://models \
           file://model_proc \
           file://face-demographics-walking.mp4"
```

```
# Instala todo dentro de /usr/share/myapp
do_install() {
    install -d ${D}${datadir}/myapp
    cp -r ${WORKDIR}/run_pose.py ${D}${datadir}/myapp/
    cp -r ${WORKDIR}/utils ${D}${datadir}/myapp/
    cp -r ${WORKDIR}/models ${D}${datadir}/myapp/
    cp -r ${WORKDIR}/model_proc ${D}${datadir}/myapp/
    cp -r ${WORKDIR}/face-demographics-walking.mp4 ${D}${datadir}/myapp/
}

FILES:${PN} += "${datadir}/myapp"
```

Esta fue la receta, en donde se especifican las rutas de los archivos necesarios así como la indicación de los elementos que debe copiar al crear la imagen.

Target Machine

Dentro del archivo conf/local.conf se específicó esta linea:

```
MACHINE ??= "qemux86-64"
```

Esta configuración define que la imagen se construirá para una arquitectura x86 de 64 bits virtualizada en QEMU.

Proceso síntesis de la imagen

Al tener listas las layers y el archivo local.conf correctamente configurados, se ejecuta:

```
bitbake core-image-minimal
```

Esto genera los archivos necesarios para hacer las pruebas con el virtualizador qemu usando:

```
runqemu qemux86-64 nographic serialstdio
```

Instalación de imagen sobre máquina virtual

Como requisito previo, en el archivo de Yocto Project, en local.config se agrega la linea:

```
IMAGE_FSTYPES = "wic wic.vdi"
```

Lo que convierte el forma de salida en una directamente compatible para cargarle a la máquina virtual al momento de crearla. Teniendo instalado virtualbox, se crea una nueva máquina con sistema operativo linux y se seleccionada la opción de carga un disco duro existente, en este apartado seleccionamos el archivo .vdi que nos genera el proceso de ‘cocinado’. Esto nos permite acceder a una máquina virtual con el sistema operativo a la medida.

Internamente hacemos:

```
dhcpcd eth0
```

Lo que asigna un ip automáticamente, que podemos visualizar con:

```
ip a
```

COn eso hacer la conexión ssh.

Problema y aplicación implementada

La aplicación desarrollada está pensada para utilizar técnicas de visión por computadora para detectar y estimar la pose de personas en un video en tiempo real. El sistema integra GStreamer para el procesamiento de video, OpenVINO para la aceleración de inferencia, y dlstreamer, idealmente,para conectar ambos componentes. La aplicación resuelve el desafío de detectar la postura corporal humana en video, lo que implica: procesar cada fotograma del video, identificar personas en la escena, determinar la posición de

articulaciones clave (cabeza, hombros, codos, etc.) y poder visualizar los resultados en tiempo real.

Referencias

[1] https://docs.yoctoproject.org/5.0.7/brief-yoctoprojectqs/index.html

[2] https://docs.openvino.ai/2023.3/openvino_docs_install_guides_installing_openvino_yocto.html

[3] https://github.com/Tobiasfonseca/DemoYocto