HLS4ML

Introducción

En la búsqueda constante de mejorar el rendimiento y la eficiencia de los modelos de aprendizaje automático, HLS4ML emerge como una herramienta innovadora que combina la potencia del aprendizaje automático con la velocidad de la lógica programable. HLS4ML significa "*High-Level Synthesis for Machine Learning*," esta permite la implementación rápida y eficiente de modelos de aprendizaje automático en FPGAs (*Field-Programmable Gate Arrays*).

A medida que los desafíos de procesamiento de datos continúan creciendo en la era de la inteligencia artificial, la capacidad de acelerar los cálculos de ML (*Machine Learning*) se ha vuelto esencial. HLS4ML ofrece una solución al traducir modelos de alto nivel desarrollados en lenguajes como Python y TensorFlow a código RTL (Register Transfer Level) que se ejecuta directamente en FPGAs. Esta conversión habilita la aceleración de inferencias y entrenamientos de modelos de ML en tiempo real, abriendo nuevas oportunidades en aplicaciones de alto rendimiento, desde la detección de objetos hasta el procesamiento de señales.

¿Qué es HLS?

HLS significa High-Level Synthesis, esta es una técnica de diseño digital utilizada en la industria de la electrónica y la ingeniería de hardware para crear circuitos electrónicos personalizados de manera más eficiente a partir de descripciones de alto nivel.

Con HLS, los diseñadores pueden describir la funcionalidad deseada de un circuito en un lenguaje de alto nivel, como C, C++, o SystemC, en lugar de tener que especificar la lógica a un nivel de detalle tan bajo. El proceso de HLS toma esta descripción de alto nivel y genera automáticamente un diseño de hardware que cumple con los requisitos de funcionalidad, esto ahorra tiempo y esfuerzo en el diseño de hardware, ya que los diseñadores pueden enfocarse en la funcionalidad y el comportamiento deseado del circuito en lugar de preocuparse por los detalles de implementación a nivel de compuertas.

El paquete hls4ml crea un código HLS eficiente a partir de redes neuronales comprimidas, que se pueden utilizar para generar módulos de hardware a partir de código escrito en lenguajes de programación de alto nivel. Esto permite a los

desarrolladores escribir código utilizando bibliotecas Python ML y convertirlo a código C compatible con Vivado HLS.

Características de HLS4ML

- Soporta una amplia gama de modelos de aprendizaje automático, incluidos CNN, RNN y MLP.
- Permite a los desarrolladores personalizar el rendimiento y el consumo de energía de sus implementaciones.
- Es fácil de usar, con una API intuitiva.

Beneficios de HLS4ML

- Rendimiento: HLS4ML puede proporcionar un rendimiento de aprendizaje automático significativamente superior al de las CPU y GPU.
- Eficiencia energética: HLS4ML puede reducir significativamente el consumo de energía de las implementaciones de aprendizaje automático.
- Flexibilidad: HLS4ML permite a los desarrolladores personalizar sus implementaciones para satisfacer sus necesidades específicas.
- Fácil de usar: HLS4ML tiene una API intuitiva y proporciona documentación completa.

Casos de uso de HLS4ML

HLS4ML se puede utilizar para una amplia gama de aplicaciones de aprendizaje automático, que incluyen:

- Reconocimiento de imágenes y videos
- Detección de objetos
- Clasificación de imágenes
- Segmentación de imágenes
- Procesamiento de lenguaje natural
- Recomendación
- Control de robots
- Sistemas de conducción autónoma

Cómo empezar con HLS4ML

Antes de empezar necesitamos Docker, que es Docker se preguntaran, eso lo explicaremos a continuación:

Docker es una plataforma de código abierto que se utiliza para desarrollar, empaquetar y ejecutar aplicaciones en contenedores. Los contenedores son entornos aislados y ligeros que permiten ejecutar aplicaciones y sus dependencias de manera consistente en cualquier entorno, ya sea en una computadora local, un servidor en la nube o en un clúster de servidores.

Docker se puede implementar de dos formas, la primer implica tener una de estas versiones de Ubuntu: Ubuntu Lunar 23.04, Ubuntu cinético 22.10, Ubuntu Jammy 22.04 (LTS), Ubuntu Focal 20.04 (LTS), o usar el docker desktop este nos ayudará a implementar docker en Windows, Mac u otra distribución de Linux.

Ya con una de las dos implementaciones de Docker podemos continuar, aquí nuevamente hay dos opciones: crear una imagen que contenga Vivado o no, por experiencia la mejor opción es si tienes uno de los sistemas operativos de Ubuntu mencionados anteriormente, instales en ella Vivado (aquí es muy importante que sea una versión entre la 2018.2 a 2020.1) y utilices la versión de sin Vivado y así todo lo que se realice se hará sobre su propia máquina pero en el caso que utilices el docker desktop utilices la imagen con Vivado. Ya sabiendo eso abre un terminal y sigue las siguientes instrucciones:

Con Vivado:

Extraiga la imagen prediseñadas del Registro de contenedores de GitHub:

```
docker pull ghcr.io/fastmachinelearning/hls4ml-tutorial/hls4ml-0.7.1-vivado-2019.2:latest
```

Para crear la imagen con Vivado, ejecute (Advertencia: lleva mucho tiempo y requiere mucho espacio en disco):

```
docker build -f docker/Dockerfile.vivado -t ghcr.io/fastmachinelearning/hls4ml-tutorial/hls4ml-0.7.1-vivado-2019.2:latest .
```

Inicie el Contenedor:

```
docker run -p 8888:8888 ghcr.io/fastmachinelearning/hls4ml-tutorial/hls4ml-0.7.1-vivado-2019.2:latest
```

Sin Vivado:

Extraiga la imagen prediseñadas del Registro de contenedores de GitHub:

```
docker pull ghcr.io/fastmachinelearning/hls4ml-tutorial/hls4ml-0.7.1:latest
```

Para crear la imagen sin Vivado, ejecute :

```
docker build https://github.com/fastmachinelearning/hls4ml-tutorial -f docker/Dockerfile.
```

Inicie el Contenedor:

```
docker run -p 8888:8888 ghcr.io/fastmachinelearning/hls4ml-tutorial/hls4ml-0.7.1:latest
```

Después de implementar el comando para iniciar el contenedor te dara un link que te llevará a Jupyter, lo ideal es que intentes implementar los ejemplos que se encuentran aqui:
https://github.com/fastmachinelearning/hls4ml-tutorial/blob/main/part1_getting_started.ipynb

Estos ejemplos aunque útiles tienen algunos problemas u otras cosas que no especifican correctamente por eso trataremos aquí de resolver algunos de los problemas con los que podrías toparte.

Nota: No es necesario correr todos los ejemplos sobre la tarjeta porque hay librerías que las tarjetas no tienen incluidas por lo que seguramente intentarán instalarlas esto no es lo más recomendable ya que se demorara según la libreria de 4 horas en adelante y es un proceso que se puede colgar fácilmente por lo cual es mejor evitarlo, basta con correr solo el ejemplo 7b sobre la tarjeta.

Problemas u otros

1. Docker desktop requires a newer wsl kernel version

Si es la primera vez que intentas utilizar docker desktop en Windows lo mas seguro es que te encontrarás con este error, el cual no permitirá que se abra Docker, se puede solucionar fácilmente siguiendo los pasos del siguiente link:

https://learn.microsoft.com/es-es/windows/wsl/install-manual#step-4---download-the-linux-kernel-update-package

2. ¿Qué es callbacks / plotting ?

Si solo bajas los ejemplos del GitHub te presentaras con el problema de que no se puede importar all_callbacks o plotting o algo parecido, esto se resuelve fácilmente bajando también del GitHub dos archivos llamados así callbacks.py e plotting.py, es un error muy tonto pero que te puede hacer perder mucho tiempo si no sabes de que se esta hablando.

3. Vivado Path

Puede llegar a presentarse el problema con la siguiente línea de código que está en los ejemplos:

os.environ['PATH'] = os.environ['XILINX_VIVADO'] + '/bin:' + os.environ['PATH']

Esta tal como está solo funcionará si instalaste la versión de la imagen que contiene Vivado pero si no tienes que cambiar la ruta a donde está instalado en tu equipo Vivado, y quedara algo asi:

os.environ['PATH'] += os.pathsep + '/home/gegerin/Vivado_HLS/Vivado/2019.2/bin'

4. No se genera el bitfile

Esto es un problema de Vivado como tal, del cuaderno 7a después del PATH se le tiene que añadir la siguiente línea de código:

os.environ['LD_PRELOAD'] = '/lib/x86_64-linux-gnu/libudev.so.1'

Quedaria asi:

```
from tensorflow.keras.models import load_model
from qkeras.utils import _add_supported_quantized_objects

co = {}
_add_supported_quantized_objects(co)
import os

os.environ['PATH'] = os.environ['XILINX_VIVADO'] + '/bin:' + os.environ['PATH']
os.environ['LD_PRELOAD'] = '/lib/x86_64-linux-gnu/libudev.so.1'
```

5. hls_model

A la hora de crear el modelo HLS este se puede abordar de muchas maneras, el backend='VivadoAccelerator', solo funciona para la siguientes tarjetas:

- pynq-z2 (part: `xc7z020clg400-1`)
- zcu102 (part: `xczu9eg-ffvb1156-2-e`)
- alveo-u50 (part: `xcu50-fsvh2104-2-e`)
- alveo-u250 (part: `xcu250-figd2104-2L-e`)
- alveo-u200 (part: `xcu200-fsgd2104-2-e`)
- alveo-u280 (part: `xcu280-fsvh2892-2L-e`)

Estos se pueden referenciar por su nombre como: board='pynq-z2' o por su número de serie part= xc7z020clg400-1.

También hay que tener en cuenta que si se está utilizando la versión de la imagen sin Vivado debes haber instalado los archivos de la tarjeta que vas a utilizar en Vivado.

Nota: Aunque antes se dijo que solo funciona con las tarjetas que se especifican se comprobó que lo relacionado con la tarjeta PYNQ-Z2 funciona para la PYNQ-Z1.