Exercise_FPGA_and_the_DevCloud

July 2, 2020

1 Exercise: FPGA and the DevCloud

Now that we've walked through the process of requesting an edge node with a CPU and Intelő Arria 10 FPGA on Intel's DevCloud and loading a model on the Intelő Arria 10 FPGA, you will have the opportunity to do this yourself with the addition of running inference on an image.

In this exercise, you will do the following: 1. Write a Python script to load a model and run inference 10 times on a device on Intel's DevCloud. * Calculate the time it takes to load the model. * Calculate the time it takes to run inference 10 times. 2. Write a shell script to submit a job to Intel's DevCloud. 3. Submit a job using `qsub` on an **IEI Tank-870** edge node with an **Intelő Arria 10 FPGA**. 4. Run `liveQStat` to view the status of your submitted jobs. 5. Retrieve the results from your job. 6. View the results.

Click the **Exercise Overview** button below for a demonstration.

Exercise Overview

IMPORTANT: Set up paths so we can run Dev Cloud utilities You *must* run this every time you enter a Workspace session.

```
In [1]: %env PATH=/opt/conda/bin:/opt/spark-2.4.3-bin-hadoop2.7/bin:/opt/conda/bin:/usr/local/sb
        import os
        import sys
        sys.path.insert(0, os.path.abspath('/opt/intel_devcloud_support'))
        sys.path.insert(0, os.path.abspath('/opt/intel'))
```

```
env: PATH=/opt/conda/bin:/opt/spark-2.4.3-bin-hadoop2.7/bin:/opt/conda/bin:/usr/local/sbin:/usr/
```

1.1 The Model

We will be using the `vehicle-license-plate-detection-barrier-0106` model for this exercise. Remember that to run a model on the FPGA, we need to use `FP16` as the model precision.

The model has already been downloaded for you in the `/data/models/intel` directory on Intel's DevCloud.

We will be running inference on an image of a car. The path to the image is `/data/resources/car.png`

2 Step 1: Creating a Python Script

The first step is to create a Python script that you can use to load the model and perform inference. We'll use the `%%writefile` magic to create a Python file called `inference_on_device.py`. In the next cell, you will need to complete the `TODO` items for this Python script.

`TODO` items:

1. Load the model
2. Get the name of the input node
3. Prepare the model for inference (create an input dictionary)
4. Run inference 10 times in a loop

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [2]: %%writefile inference_on_device.py

        import time
        import cv2
        import numpy as np
        from openvino.inference_engine import IENetwork
        from openvino.inference_engine import IECore
        import argparse

        def main(args):
            model=args.model_path
            model_weights=model+'.bin'
            model_structure=model+'.xml'

            start=time.time()
            model=IENetwork(model_structure, model_weights)

            core = IECore()
            net = core.load_network(network=model, device_name=args.device, num_requests=1)
            print(f"Time taken to load model = {time.time()-start} seconds")

            # Get the name of the input node
            input_name=next(iter(model.inputs))

            # Reading and Preprocessing Image
            input_img=cv2.imread('/data/resources/car.png')
            input_img=cv2.resize(input_img, (300,300), interpolation = cv2.INTER_AREA)
            input_img=np.moveaxis(input_img, -1, 0)

            # Running Inference in a loop on the same image
            input_dict={input_name:input_img}
```

```
    start=time.time()
    for _ in range(10):
        net.infer(input_dict)

    print(f"Time Taken to run 10 Inference on FPGA is = {time.time()-start} seconds")

if __name__=='__main__':
    parser=argparse.ArgumentParser()
    parser.add_argument('--model_path', required=True)
    parser.add_argument('--device', default=None)

    args=parser.parse_args()
    main(args)
```

```
Overwriting inference_on_device.py
```

Show Solution

2.1 Step 2: Creating a Job Submission Script

To submit a job to the DevCloud, you'll need to create a shell script. Similar to the Python script above, we'll use the `%%writefile` magic command to create a shell script called `inference_fpga_model_job.sh`. In the next cell, you will need to complete the `TODO` items for this shell script.

`TODO` items: 1. Create three variables: * `DEVICE` - Assign the value as the first argument passed into the shell script. * `MODELPATH` - Assign the value as the second argument passed into the shell script. 2. Call the Python script using the three variable values as the command line argument

If you get stuck, you can click on the **Show Solution** button below for a walkthrough with the solution code.

```
In [3]: %%writefile inference_fpga_model_job.sh
        #!/bin/bash

        exec 1>/output/stdout.log 2>/output/stderr.log

        mkdir -p /output

        DEVICE=$1
        MODELPATH=$2


        source /opt/intel/init_openvino.sh
        aocl program acl0 /opt/intel/openvino/bitstreams/a10_vision_design_sg1_bitstreams/2019R4


        # Run the load model python script
        python3 inference_on_device.py  --model_path ${MODELPATH} --device ${DEVICE}
        cd /output

        tar zcvf output.tgz stdout.log stderr.log
```

```
Overwriting inference_fpga_model_job.sh
```

Show Solution

2.2 Step 3: Submitting a Job to Intel's DevCloud

In the next cell, you will write your `!qsub` command to load your model and run inference on the **IEI Tank-870** edge node with an **Intel Core i5** CPU and an **Intelő Arria 10 FPGA**.

Your `!qsub` command should take the following flags and arguments: 1. The first argument should be the shell script filename 2. `-d` flag - This argument should be `.` 3. `-l` flag - This argument should request an edge node with an **IEI Tank-870**. The default quantity is 1, so the **1** after `nodes` is optional. * **Intel Core i5 6500TE** for your `CPU`. * **Intelő Arria 10** for your `FPGA`.

To get the queue labels for these devices, you can go to this link

4. `-F` flag - This argument should contain the two values to assign to the variables of the shell script:
   - **DEVICE** - Device type for the job: `FPGA`. Remember that we need to use the **Heterogenous plugin** (HETERO) to run inference on the FPGA.
   - **MODELPATH** - Full path to the model for the job. As a reminder, the model is located in `/data/models/intel`.

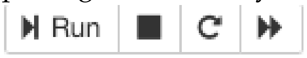
Note: There is an optional flag, `-N`, you may see in a few exercises. This is an argument that only works on Intel's DevCloud that allows you to name your job submission. This argument doesn't work in Udacity's workspace integration with Intel's DevCloud.

```
In [4]: job_id_core = !qsub inference_fpga_model_job.sh -d . -l nodes=1:tank-870:i5-6500te:iei-m
        print(job_id_core[0])
```

```
OczoupyOueOXm7GXWW8SatOAYixK82C5
```

Show Solution

2.3 Step 4: Running liveQStat

Running the `liveQStat` function, we can see the live status of our job. Running the this function will lock the cell and poll the job status 10 times. The cell is locked until this finishes polling 10 times or you can interrupt the kernel to stop it by pressing the stop button at the top:


- `Q` status means our job is currently awaiting an available node
- `R` status means our job is currently running on the requested node

Note: In the demonstration, it is pointed out that `W` status means your job is done. This is no longer accurate. Once a job has finished running, it will no longer show in the list when running the `liveQStat` function.

Click the **Running liveQStat** button below for a demonstration.

Running liveQStat

```
In [5]: import liveQStat
        liveQStat.liveQStat()
```

2.4 Step 5: Retrieving Output Files

In this step, we'll be using the `getResults` function to retrieve our job's results. This function takes a few arguments.

1. `job id` - This value is stored in the `job_id_core` variable we created during **Step 3**. Remember that this value is an array with a single string, so we access the string value using `job_id_core[0]`.
2. `filename` - This value should match the filename of the compressed file we have in our `inference_fpga_model_job.sh` shell script.
3. `blocking` - This is an optional argument and is set to `False` by default. If this is set to `True`, the cell is locked while waiting for the results to come back. There is a status indicator showing the cell is waiting on results.

Note: The `getResults` function is unique to Udacity's workspace integration with Intel's DevCloud. When working on Intel's DevCloud environment, your job's results are automatically retrieved and placed in your working directory.

Click the **Retrieving Output Files** button below for a demonstration.

Retrieving Output Files

```
In [6]: import get_results

        get_results.getResults(job_id_core[0], filename="output.tgz", blocking=True)
```

```
getResults() is blocking until results of the job (id:0czoupy0ue0Xm7GXWW8Sat0AYixK82C5) are read
Please wait...Success!
output.tgz was downloaded in the same folder as this notebook.
```

```
In [7]: !tar zxf output.tgz
```

```
In [8]: !cat stdout.log
```

```
INTELFPGAOCLSDKROOT is set to /opt/altera/aocl-pro-rte/aclrte-linux64. Using that.

aoc was not found, but aocl was found. Assuming only RTE is installed.

AOCL_BOARD_PACKAGE_ROOT is set to /opt/intel/openvino/bitstreams/a10_vision_design_sg2_bitstream
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/bin to PATH
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/linux64/lib to LD_LIBRARY_PATH
Adding /opt/altera/aocl-pro-rte/aclrte-linux64/host/linux64/lib to LD_LIBRARY_PATH
Adding /opt/intel/openvino/bitstreams/a10_vision_design_sg2_bitstreams/BSP/a10_1150_sg2/linux64/
[setupvars.sh] OpenVINO environment initialized
aocl program: Running program from /opt/intel/openvino/bitstreams/a10_vision_design_sg2_bitstrea
Failed to open file: /opt/intel/openvino/bitstreams/a10_vision_design_sg1_bitstreams/2019R4_PL1_
Error: Failed to find aocx
aocl program: Program failed.
Time taken to load model = 4.5110766887664795 seconds
Time Taken to run 10 Inference on FPGA is = 0.08702898025512695 seconds
```

In [9]: !cat stderr.log

```
Couldn't open file /opt/intel/openvino/bitstreams/a10_vision_design_sg1_bitstreams/2019R4_PL1_FP
inference_on_device.py:15: DeprecationWarning: Reading network using constructor is deprecated.
  model=IENetwork(model_structure, model_weights)
tar: stdout.log: file changed as we read it
```

In [ ]: