C++ Python

Multi-Device Plugin

Introducing the Multi-Device Plugin (Python)

The Multi-Device plugin automatically assigns inference requests to available computational devices to execute the requests in parallel. By contrast, the Heterogeneous plugin can run different layers on different devices but not in parallel. The potential gains with the Multi-Device plugin are:

- Improved throughput from using multiple devices (compared to single-device execution)
- More consistent performance, since the devices share the inference burden (if one device is too busy, another can take more of the load)

Note that with Multi-Device the application logic is left unchanged, so you don't need to explicitly load the network to every device, create and balance the inference requests and so on. From the application point of view, this is just another device that handles the actual machinery. The only thing that is required to leverage performance is to provide the multi-device (and hence the underlying devices) with enough inference requests to process. For example, if you were processing 4 cameras on the CPU (with 4 inference requests), it might be desirable to process more cameras (with more requests in flight) to keep CPU and GPU busy via Multi-Device.

The setup of Multi-Device can be described in three major steps:

1. Configure each device as usual (using the conventional ie_api.IECore.set_config method
2. Load the network to the Multi-Device plugin created on top of a (prioritized) list of the configured devices. This is the only change needed in the application.
3. As with any other ExecutableNetwork call (resulting from `load_network`), you create as many requests as needed to saturate the devices.

These steps are covered below in detail.

Defining and Configuring the Multi-Device Plugin

Following the OpenVINO™ convention of labeling devices, the Multi-Device plugin uses the name "MULTI". The only configuration option for the Multi-Device plugin is a prioritized list of devices to use:

Parameter name	Parameter values	Default	Description
"MULTI_DEVICE_PRIORITIES"	comma-separated device names with no spaces	N/A	Prioritized list of devices

You can set the configuration directly as a string, or use the metric key `MULTI_DEVICE_PRIORITIES` from the `multi/multi_device_config.hpp` file, which defines the same string.

The Three Ways to Specify Devices Targets for the MULTI plugin

- Option 1 - Pass a Prioritized List as a Parameter in ie.load_network()

```
from openvino.inference_engine import IECore

ie = IECore()
# Read a network in IR or ONNX format
net = ie.read_network(model=path_to_model)
exec_net = ie.load_network(network=net, device_name="MULTI:CPU,GPU")
```

- Option 2 - Pass a List as a Parameter, and Dynamically Change Priorities during Execution Notice that the priorities of the devices can be changed in real time for the executable network:

```
from openvino.inference_engine import IECore

# Init the Inference Engine Core
ie = IECore()

# Read a network in IR or ONNX format
net = ie.read_network(model=path_to_model)

ie.set_config( config={"MULTI_DEVICE_PRIORITIES":"HDDL,GPU"},
device_name="MULTI")

# Change priorities
ie.set_config( config={"MULTI_DEVICE_PRIORITIES":"GPU,HDDL"},
device_name="MULTI")
ie.set_config( config={"MULTI_DEVICE_PRIORITIES":"GPU"}, device_name="MULTI")
ie.set_config( config={"MULTI_DEVICE_PRIORITIES":"HDDL,GPU"},
device_name="MULTI")
ie.set_config( config={"MULTI_DEVICE_PRIORITIES":"CPU,HDDL,GPU"},
device_name="MULTI")
```

- Option 3 - Use Explicit Hints for Controlling Request Numbers Executed by Devices There is a way to specify the number of requests that Multi-Device will internally keep for each device. If the original app was running 4 cameras with 4 inference requests, it might be best to share these 4 requests between 2 devices used in the MULTI. The easiest way is to specify a number of requests for each device using parentheses: “MULTI:CPU(2),GPU(2)” and use the same 4 requests in the app. However, such an explicit configuration is not performance-portable and not recommended. The better way is to configure the individual devices and query the resulting number of requests to be used at the application level. See Configuring the Individual Devices and Creating the Multi-Device On Top.

Enumerating Available Devices

The Inference Engine features a dedicated API to enumerate devices and their capabilities. See the Hello Query Device Python Sample. This is example output from the sample (truncated to device names only):

```
./hello_query_device
Available devices:
    Device: CPU
...
    Device: GPU.0
...
    Device: GPU.1
...
    Device: HDDL
```

A simple programmatic way to enumerate the devices and use with the multi-device is as follows:

```
from openvino.inference_engine import IECore

all_devices = "MULTI:"
ie = IECore()
net = ie.read_network(model=path_to_model)
all_devices += ",".join(ie.available_devices)
exec_net = ie.load_network(network=net, device_name=all_devices)
```

Beyond the trivial “CPU”, “GPU”, “HDDL” and so on, when multiple instances of a device are available the names are more qualified. For example, this is how two Intel® Movidius™ Myriad™ X sticks are listed with the hello_query_sample:

```
...
    Device: MYRIAD.1.2-ma2480
...
    Device: MYRIAD.1.4-ma2480
```

So the explicit configuration to use both would be “MULTI:MYRIAD.1.2-ma2480,MYRIAD.1.4-ma2480”. Accordingly, the code that loops over all available devices of “MYRIAD” type only is below:

```
from openvino.inference_engine import IECore

ie = IECore()
match_list = []
all_devices = "MULTI:"
dev_match_str = "MYRIAD"
net = ie.read_network(model=path_to_model)

for d in ie.available_devices:
    if dev_match_str in d:
        match_list.append(d)

all_devices += ",".join(match_list)
exec_net = ie.load_network(network=net, device_name=all_devices)
```

Configuring the Individual Devices and Creating the Multi-Device On Top

It is possible to configure each individual device as usual and then create the “MULTI” device on top:

```
from openvino.inference_engine import IECore

ie = IECore()
net = ie.read_network(model=path_to_model)

cpu_config = {}
gpu_config = {}

ie.set_config(config=cpu_config, device_name="CPU")
ie.set_config(config=gpu_config, device_name="GPU")

# Load the network to the multi-device, specifying the priorities
exec_net = ie.load_network(
    network=net, device_name="MULTI", config={"MULTI_DEVICE_PRIORITIES": "CPU,GPU"}
)
# Query the optimal number of requests
nireq = exec_net.get_metric("OPTIMAL_NUMBER_OF_INFER_REQUESTS")
```

An alternative is to combine all the individual device settings into a single config file and load that, allowing the Multi-Device plugin to parse and apply settings to the right devices. See the code example in the next section.

Note that while the performance of accelerators works well with Multi-Device, the CPU+GPU execution poses some performance caveats, as these devices share power, bandwidth and other resources. For example it is recommended to enable the GPU throttling hint (which saves another CPU thread for CPU inferencing). See the section below titled Using the Multi-Device with OpenVINO Samples and Benchmarking the Performance.

Using the Multi-Device with OpenVINO Samples and Benchmarking the Performance

Every OpenVINO sample that supports the `-d` (which stands for “device”) command-line option transparently accepts Multi-Device. The Benchmark application is the best reference for the optimal usage of Multi-Device. As discussed earlier, you do not need to set up the number of requests, CPU streams or threads because the application provides optimal performance out of the box. Below is an example command to evaluate CPU+GPU performance with the Benchmark application:

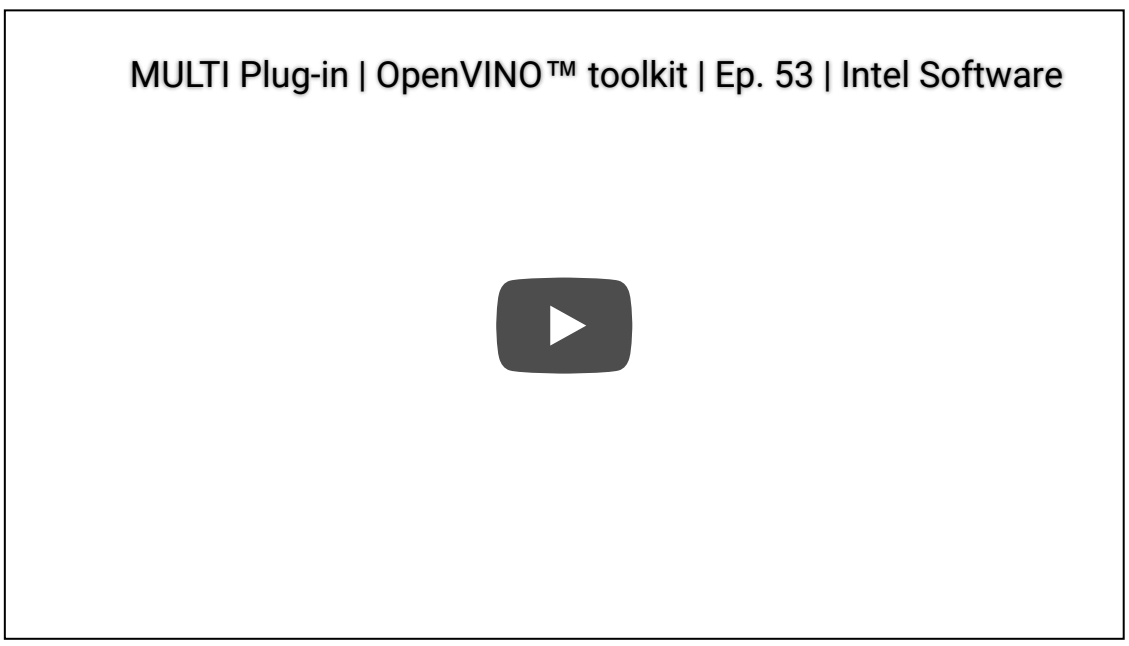
```
./benchmark_app.py –d MULTI:CPU,GPU –m <model>
```

> **Note**
>
> If you installed OpenVINO with pip, use `benchmark_app -d MULTI:CPU,GPU -m <model>`

The Multi-Device plugin supports FP16 IR files. The CPU plugin automatically upconverts it to FP32 and the other devices support it natively. Note that no demos are (yet) fully optimized for Multi-Device, by means of supporting the OPTIMAL_NUMBER_OF_INFER_REQUESTS metric, using the GPU streams/throttling, and so on.

Video: MULTI Plugin

> **Note**
>
> This video is currently available only for C++, but many of the same concepts apply to Python.



See Also

Supported Devices