DOCS: updated docs with InferenceEngine::Core
Lavrenov, Ilya authored 1 week ago

c3cb036c

MULTI.md 11 KB

Multi-Device Plugin

Introducing Multi-Device Execution

Multi-Device plugin automatically assigns inference requests to available computational devices to execute the requests in parallel. Potential gains are as follows

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

Notice that with multi-device the application logic 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 crunch. For example if you were processing 4 cameras on the CPU (with 4 inference requests), you may now want to process more cameras (with more requests in flight) to keep CPU+GPU busy via multi-device.

The "setup" of multi-device can be described in three major steps:

- First is configuration of each device as usual (e.g. via conventional SetConfig method)
- Second is loading of a network to the Multi-Device plugin created on top of (prioritized) list of the configured devices. This is the only change that you need in your application.
- Finally, just like with any other ExecutableNetwork (resulted from LoadNetwork) you just create as many requests as needed to saturate the devices. These steps are covered below in details.

Defining and Configuring the Multi-Device

Following the OpenVINO notions of "devices", the multi-device has a "MULTI" name. The only configuration option for the multi-device is 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 use name of the configuration directly as a string, or use MultiDeviceConfigParams::KEY_MULTI_DEVICE_PRIORITIES from the multi/multi_device_config.hpp that defines the same string.

Basically, there are three ways to specify the devices to be use by the "MULTI":

```
Core ie;
//NEW IE-CENTRIC API, the "MULTI" plugin is (globally) pre-configured with the explicit option:
ie.SetConfig({{"MULTI_DEVICE_PRIORITIES", "HDDL,GPU"}}, "MULTI");
ExecutableNetwork exec0 = ie.LoadNetwork(network, "MULTI", {});

//NEW IE-CENTRIC API, configuration of the "MULTI" is part of the network configuration (and hence specific to the netwo
ExecutableNetwork exec1 = ie.LoadNetwork(network, "MULTI", {{"MULTI_DEVICE_PRIORITIES", "HDDL,GPU"}});
//NEW IE-CENTRIC API, same as previous, but configuration of the "MULTI" is part of the name (so config is empty), also
ExecutableNetwork exec2 = ie.LoadNetwork(network, "MULTI:HDDL,GPU", {});

//Similarly for the deprecated (plugin-centric) API
//for example globally pre-configuring the plugin with the explicit option:
//auto plugin0 = PluginDispatcher().getPluginByDevice("MULTI");
//plugin0.SetConfig({{"MULTI_DEVICE_PRIORITIES", "HDDL,GPU"}});
//ExecutableNetwork exec3 = plugin.LoadNetwork(network, {});
// part of the config for the LoadNetwork or device name
//ExecutableNetwork exec4 = plugin0.LoadNetwork(network, {{"MULTI_DEVICE_PRIORITIES", "HDDL,GPU"}});
// part of the device name
//auto plugin1 = PluginDispatcher().getPluginByDevice("MULTI:HDDL,GPU");
//ExecutableNetwork exec5 = plugin1.LoadNetwork(network, {});
```

Notice that the priorities of the devices can be changed in real-time for the executable network:

```
Core ie;
ExecutableNetwork exec = ie.LoadNetwork(network, "MULTI:HDDL,GPU", {});
//...
exec.SetConfig({{"MULTI_DEVICE_PRIORITIES", "GPU,HDDL"}});
// you can even exclude some device
exec.SetConfig({{"MULTI_DEVICE_PRIORITIES", "GPU"}});
//...
// and then return it back
exec.SetConfig({{"MULTI_DEVICE_PRIORITIES", "GPU,HDDL"}});
//but you cannot add new devices on the fly, the next line will trigger the following exception:
//[ ERROR ] [NOT_FOUND] You can only change device priorities but not add new devices with the Network's SetConfig(Multi
//CPU device was not in the original device list!
exec.SetConfig({{"MULTI_DEVICE_PRIORITIES", "CPU,GPU,HDDL"}});
```

Finally, there is a way to specify number of requests that the multi-device will internally keep for each device. Say if your original app was running 4 cameras with 4 inference requests now you would probably want to share these 4 requests between 2 devices used in the MULTI. The easiest way is to specify #requests for each device using brackets: "MULTI:CPU(2),GPU(2)" and use the same 4 requests in your app. However, such an explicit configuration is not performance portable and hence not recommended. Instead, the better way is to configure the individual devices and query the resulting number of requests to be used in the application level (see Configuring the Individual Devices and Creating the Multi-Device On Top).

Enumerating Available Devices

Inference Engine now features a dedicated API to enumerate devices and their capabilities. See Hello Query Device C++ Sample. This is example output of the sample (truncated to the devices' names only):

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

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

```
        Core ie;
        std::string allDevices = "MULTI:";
        std::vector<std::string> availableDevices = ie.GetAvailableDevices();
        for (auto && device : availableDevices) {
            allDevices += device;
            allDevices += ((device == availableDevices[availableDevices.size()-1]) ? "" : ",");
        }
        ExecutableNetwork exeNetwork = ie.LoadNetwork(cnnNetwork, allDevices, {});
```

Beyond 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:

```
       Core ie;
        std::string allDevices = "MULTI:";
        std::vector<std::string> myriadDevices = ie->GetMetric("MYRIAD", METRIC_KEY(myriadDevices)));
        for (int i = 0; i < myriadDevices.size(); ++i) {
            allDevices += std::string("MYRIAD.")
                                    + myriadDevices[i]
                                    + std::string(i < (myriadDevices.size() -1) ? "," : "");
        }
        ExecutableNetwork exeNetwork = ie.LoadNetwork(cnnNetwork, allDevices, {});
```

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

As discussed in the first section, you shall configure each individual device as usual and then just create the "MULTI" device on top:

```
#include <multi/multi_device_config.hpp>
// configure the HDDL device first
Core ie;
ie.SetConfig(hddl_config, "HDDL");
// configure the GPU device
ie.SetConfig(gpu_config, "GPU");
// load the network to the multi-device, while specifying the configuration (devices along with priorities):
ExecutableNetwork exeNetwork = ie.LoadNetwork(cnnNetwork, "MULTI", {{MultiDeviceConfigParams::KEY_MULTI_DEVICE_PRIORITIES, "
// new metric allows to query the optimal number of requests:
uint32_t nireq = exeNetwork.GetMetric(METRIC_KEY(OPTIMAL_NUMBER_OF_INFER_REQUESTS)).as<unsigned int>();
```

Alternatively, you can combine all the individual device settings into single config and load that, allowing the multi-device plugin to parse and apply that to the right devices. See code example in the next section.

Notice that while the performance of accelerators combines really well with multi-device, the CPU+GPU execution poses some performance caveats, as these devices share the power, bandwidth and other resources. For example it is recommended to enable the GPU throttling hint (which save another CPU thread for the CPU inference). See section of the Using the multi-device with OpenVINO samples and benchmarking the performance below.

Querying the Optimal Number of Inference Requests

Notice that until R2 you had to calculate number of requests in your application for any device, e.g. you had to know that Intel® Vision Accelerator Design with Intel® Movidius™ VPUs required at least 32 inference requests to perform well. Now you can use the new GetMetric API to query the optimal number of requests. Similarly, when using the multi-device you don't need to sum over included devices yourself, you can query metric directly:

```
// 'device_name' can be "MULTI:HDDL,GPU" to configure the multi-device to use HDDL and GPU
ExecutableNetwork exeNetwork = ie.LoadNetwork(cnnNetwork, device_name, full_config);
// new metric allows to query the optimal number of requests:
uint32_t nireq = exeNetwork.GetMetric(METRIC_KEY(OPTIMAL_NUMBER_OF_INFER_REQUESTS)).as<unsigned int>();
```

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

Notice that every OpenVINO sample that supports "-d" (which stays for "device") command-line option transparently accepts the multi-device. The Benchmark Application is the best reference to the optimal usage of the multi-device. As discussed multiple times earlier, you don't need to setup number of requests, CPU streams or threads as the application provides optimal out of the box performance. Below is example command-line to evaluate HDDL+GPU performance with that:

```
$ ./benchmark_app –d MULTI:HDDL,GPU –m <model> -i <input> -niter 1000
```

Notice that you can use the FP16 IR to work with multi-device (as CPU automatically upconverts it to the fp32) and rest of devices support it naturally. Also notice that no demos are (yet) fully optimized for the multi-device, by means of supporting the OPTIMAL_NUMBER_OF_INFER_REQUESTS metric, using the GPU streams/throttling, and so on.

See Also

- Supported Devices