C++ Python

# Плагин для нескольких устройств

## Представляем плагин Multi-Device Plugin (Python)

Плагин Multi-Device автоматически распределяет запросы на выводы по доступным вычислительным устройствам для параллельного выполнения запросов. В отличие от этого, плагин Heterogeneous может запускать разные уровни на разных устройствах, но не параллельно. Потенциальными преимуществами плагина Multi-Device являются:

- Повышение пропускной способности за счет использования нескольких устройств
- (по сравнению с выполнением на одном устройстве) Более стабильная производительность, так как устройства разделяют нагрузку на выводы (если одно устройство слишком занято, другое может взять на себя больше нагрузки)

Обратите внимание, что при использовании Multi-Device логика приложения остается неизменной, поэтому вам не нужно явно загружать сеть на каждое устройство, создавать и балансировать запросы на вывод и так далее. С точки зрения приложения, это просто еще одно устройство, которое обрабатывает фактический механизм. Единственное, что требуется для повышения производительности, - это обеспечить мультиустройство (и, следовательно, базовые устройства) достаточным количеством запросов на вывод для обработки. Например, если вы обрабатываете 4 камеры на CPU (с 4 запросами на вывод), может быть желательно обрабатывать больше камер (с большим количеством запросов в полете), чтобы держать CPU и GPU занятыми через Multi-Device.

Настройка Multi-Device может быть описана тремя основными шагами:

1. Настройте каждое устройство как обычно (используя обычный метод ie_api.IECore.set_config
2. Загрузите сеть в плагин Multi-Device, созданный поверх (приоритетного) списка настроенных устройств. Это единственное изменение, необходимое в приложении.
3. Как и при любом другом вызове ExecutableNetwork (в результате `load_network`), вы создаете столько запросов, сколько необходимо для насыщения устройств.

Эти шаги подробно описаны ниже.

## Определение и настройка плагина для нескольких устройств

Следуя конвенции OpenVINO™ для маркировки устройств, плагин Multi-Device использует символ
имя "MULTI". Единственным параметром конфигурации для плагина Multi-Device является приоритетный список устройств для использования:

| Имя параметра | Параметр valuesDefaultDescription | |
|---|---|---|
| CE_PRIORITIES"запятую. | "MULTI_DEVI<br><br>разделенные имена устройств без пробелов | N/Приоритетный списокустройств |

Вы можете задать конфигурацию непосредственно в виде строки или использовать метрический ключ `MULTI_DEVICE_PRIORITIES`

из файла `multi/multi_device_config.hpp`, который определяет ту же строку.

## Три способа указания целевых устройств для плагина MULTI

- Вариант 1 - Передать список приоритетов в качестве параметра в ie.load_network()

```
from openvino.inference_engine import IECore

ie = IECore()
# Чтение сети в формате IR или ONNX
net = ie.read_network(model=path_to_model)
exec_net = ie.load_network(network=net, device_name="MULTI:CPU,GPU")
```

- Вариант 2 - Передача списка в качестве параметра и динамическое изменение приоритетов во время выполнения Обратите внимание, что приоритеты устройств могут быть изменены в реальном времени для исполняемой сети:

```
from openvino.inference_engine import IECore

# Инициировать ядро механизма вывода
ie = IECore()

# Чтение сети в формате IR или ONNX
net = ie.read_network(model=path_to_model)

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

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

- Вариант 3 - Использование явных подсказок для управления количеством запросов, выполняемых устройствами Существует способ указать количество запросов, которые Multi-Device будет внутренне хранить для каждого устройства. Если в исходном приложении использовалось 4 камеры с 4 запросами на вывод, возможно, лучше разделить эти 4 запроса между 2 устройствами, используемыми в MULTI. Самый простой способ - указать количество запросов для каждого устройства, используя круглые скобки: "MULTI:CPU(2),GPU(2)" и использовать те же 4 запроса в приложении. Однако такая явная конфигурация не является переносимой по производительности и не рекомендуется. Лучшим способом является настройка отдельных устройств и запрос результирующего количества запросов для использования на уровне приложения. См. раздел "Настройка отдельных устройств и создание многоуровневого устройства сверху".

## Перечисление доступных устройств

Inference Engine имеет специальный API для перечисления устройств и их возможностей. См. пример Hello Query Device Python Sample. Это пример вывода из примера (усеченный только до имен устройств):

```
.
/hello_query_device
Доступные
устройства:
    Устройство: CPU
...
    Устройство: GPU.0
...
    Устройство: GPU.1
...
    Устройство: HDDL
```

Простой программный способ перечисления устройств и использования с мультиустройством заключается в следующем:

```
from openvino.inference_engine import IECore

all_devices = "MULTI:"ė
= 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)
```

Помимо тривиальных "CPU", "GPU", "HDDL" и т.д., когда доступно несколько экземпляров устройства, имена становятся более конкретными. Например, вот как в примере hello_query_sample перечислены две палочки Intel® Movidius™ Myriad™ X:

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

Поэтому явная конфигурация для использования обоих устройств будет выглядеть так: "MULTI:MYRIAD.1.2-ma2480,MYRIAD.1.4- ma2480". Соответственно, ниже приведен код, который перебирает все доступные устройства только типа "MYRIAD":

```
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)

для d в 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)
```

## Настройка отдельных устройств и создание нескольких устройств сверху

Можно настроить каждое отдельное устройство как обычно, а затем создать сверху устройство "MULTI":

```
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,
имя_устройства="CPU")ėset_config(config=gpu_config,
имя_устройства="GPU")

# Загрузите сеть на мультиустройство, указав приоритеты
exec_net = ie.load_network(
    network=net, device_name="MULTI", config={"MULTI_DEVICE_PRIORITIES": "CPU,GPU"}
)
# Запрос оптимального количества запросов
nireq = exec_net.get_metric("OPTIMAL_NUMBER_OF_INFER_REQUESTS")
```

Альтернативный вариант - объединить все настройки отдельных устройств в один конфигурационный файл и загрузить его, что позволит плагину Multi-Device разобрать и применить настройки к нужным устройствам. Смотрите пример кода в следующем разделе.

Обратите внимание, что хотя производительность ускорителей хорошо работает в Multi-Device, выполнение CPU+GPU создает некоторые ограничения по производительности, поскольку эти устройства совместно используют энергию, пропускную способность и другие ресурсы. Например, рекомендуется включить подсказку GPU throttling (которая сохраняет еще один поток CPU для CPU inferencing). См. раздел ниже, озаглавленный

"Использование нескольких устройств с образцами OpenVINO и бенчмаркинг производительности".

## Использование образцов Multi-Device with OpenVINO и сравнительный анализ производительности

Каждый пример OpenVINO, поддерживающий опцию командной строки `-d` (что означает "устройство"), прозрачно принимает Multi-Device. Приложение Benchmark является лучшим эталоном для оптимального использования Multi-Device. Как говорилось ранее, вам не нужно настраивать количество запросов, потоков ЦП или потоков, поскольку приложение обеспечивает оптимальную производительность "из коробки". Ниже приведен пример команды для оценки производительности CPU+GPU с помощью приложения Benchmark:

```
. /benchmark_app.py -d MULTI:CPU,GPU -m <модель>
```

**Примечание**

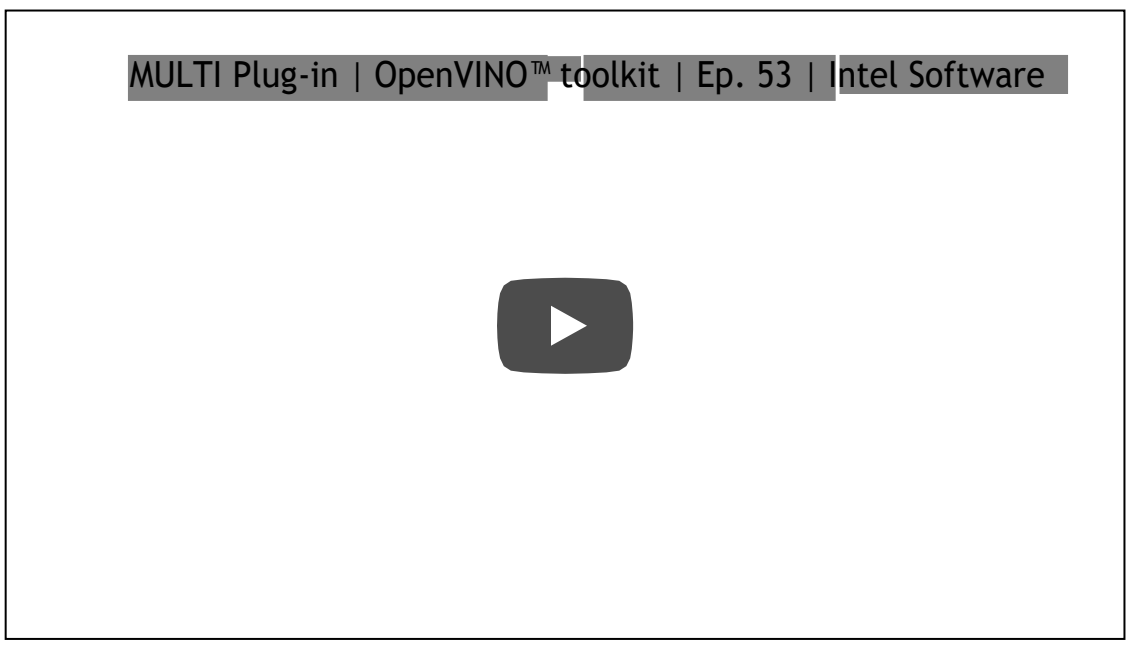
Если вы установили OpenVINO с помощью pip, используйте `benchmark_app -d MULTI:CPU,GPU -m <модель>`.

Плагин Multi-Device поддерживает файлы FP16 IR. Плагин CPU автоматически преобразует их в FP32, а остальные устройства поддерживают их нативно. Обратите внимание, что ни одна демонстрация (пока) не оптимизирована полностью для Multi-Device, посредством поддержки метрики OPTIMAL_NUMBER_OF_INFER_REQUESTS, использования потоков GPU/тротлинга и так далее.

## Видео: MULTI Plugin

**Примечание**

В настоящее время это видео доступно только для C++, но многие из тех же концепций применимы и к Python.




## См. также

Поддерживаемые устройства