C++ Python

# Гетерогенный плагин

## Представляем плагин Heterogeneous Plugin (Python)

Гетерогенный плагин позволяет вычислять выводы одной сети на нескольких устройствах. Целями выполнения сетей в гетерогенном режиме являются:

- Использование мощности ускорителей для обработки наиболее тяжелых частей сети и выполнения неподдерживаемых уровней на резервных устройствах, таких как CPU
- Более эффективное использование всего доступного оборудования во время одного умозаключения

Выполнение через гетерогенный плагин можно разделить на два независимых этапа:

1. Настройка сродства оборудования к слоям
2. Загрузка сети в плагин Heterogeneous, разбиение сети на части и выполнение их через плагин

Эти шаги разделены. Настройка сродства может быть выполнена автоматически с использованием политики отката или в ручном режиме.

Автоматическая политика отката вызывает "жадное" поведение и назначает все слои, которые могут быть выполнены на определенном устройстве, в соответствии с указанными вами приоритетами (например, HETERO:GPU,CPU). Автоматическая политика не учитывает особенности плагина, такие как невозможность инференции некоторых слоев без других специальных слоев, расположенных до или после этого слоя. За решение таких случаев отвечает плагин. Если плагин устройства не поддерживает топологию подграфа, построенного плагином HETERO, то следует задать аффинити вручную.

Некоторые топологии плохо поддерживаются для гетерогенного выполнения на некоторых устройствах или вообще не могут быть выполнены в этом режиме. Примерами таких сетей являются сети, имеющие уровни активации, которые не поддерживаются на основном устройстве. Если передача данных из одной части сети в другую в гетерогенном режиме занимает больше времени, чем в обычном режиме, возможно, не имеет смысла выполнять их в гетерогенном режиме. В этом случае можно определить наиболее требовательную к вычислениям часть вручную и установить сродство, чтобы избежать многократной пересылки данных туда и обратно в течение одного вывода.

## Использовать аффинити слоев по умолчанию

Чтобы использовать сродства по умолчанию, вызовите `load_network` с устройством "HETERO" и необязательным списком устройств для рассмотрения.

```
from openvino.inference_engine import IECore

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

# Аннотация уровней для каждого устройства и политика отката по умолчанию

Политика отката по умолчанию решает, какой слой идет на какое устройство автоматически в соответствии с поддержкой в специальных плагинах (GPU, CPU, MYRIAD).

Другой способ аннотирования сети - установить сродство вручную с помощью кода.

## Установите аффинити всех слоев на CPU

```
import ngraph as ng
из openvino.inference_engine import IECore

ie = IECore()
# Чтение сети в формате IR или ONNX
net = ie.read_network(path_to_model)
# Создайте функцию Ngraph (график) из сети
ng_func = ng.function_from_cnn(net)
for node in ng_func.get_ordered_ops():
    rt_info = node.get_rt_info()
    rt_info["affinity"] = "CPU"
```

Политика отката не работает, если хотя бы один слой имеет инициализированное сродство. Последовательность действий должна заключаться в вызове настроек сродства по умолчанию, а затем в установке слоев вручную.

> **Примечание**
>
> Если вы задаете сродство вручную, имейте в виду, что в настоящее время плагины Inference Engine не поддерживают постоянные (*Constant -> Result*) и пустые (*Parameter -> Result*) сети. Пожалуйста, избегайте этих подграфов, если вы задаете аффинити вручную.

## Пример - Ручная настройка аффинити слоев

```
import ngraph as ng
из openvino.inference_engine import IECore

ie = IECore()
# Читаем сеть в формате IR или
ONNX net =
ie.read_network(path_to_model)
ng_func = ng.function_from_cnn(net)

for node in ng_func.get_ordered_ops():
    rt_info = node.get_rt_info()
    rt_info["affinity"] = "CPU"

# Загрузите сеть на целевом устройстве
exec_net = ie.load_network(network=net, device_name='HETERO:FPGA,CPU')
```

> **Примечание**
>
> `ie.query_network` не зависит от аффинити, установленных пользователем, а запрашивает поддержку уровней на основе возможностей устройства.

## Детали разделения сети и исполнения

Во время загрузки сети в гетерогенный плагин сеть делится на отдельные части и загружается в выделенные плагины. Промежуточные блобы между этими подграфами распределяются автоматически наиболее эффективным способом.

## Точность исполнения

Точность для выводов в неоднородном плагине определяется:

- Точность ИК
- Способность конечных плагинов выполняться с точностью, определенной в IR

Пример:

- Если вы хотите выполнить GPU с CPU fallback с FP16 на GPU, вам нужно использовать только FP16 IR.

Образцы OpenVINO можно использовать с помощью следующей команды:

```
. /object_detection_sample_ssd -m <путь_к_модели>/ModelSSD.xml -i
<путь_к_картинкам>/picture.jpg -d HETERO:MYRIAD,CPU
```

где HETERO означает гетерогенный плагин

Вы можете указать более двух устройств, например: `-d HETERO:MYRIAD,GPU,CPU`

## Анализ гетерогенного исполнения

После включения конфигурационного ключа KEY_HETERO_DUMP_GRAPH_DOT вы можете создавать дамп GraphViz*.

файлы .dot с аннотациями устройств для

каждого слоя. Гетерогенный плагин может

генерировать два файла:

- `hetero_affinity_<имя сети>.dot` - аннотация сродств для каждого слоя. Этот файл записывается на диск только в том случае, если была выполнена политика отката по умолчанию
- `hetero_subgraphs_<имя сети>.dot` - аннотация сродств для каждого графа. Этот файл записывается на диск во время выполнения `ICNNNetwork::LoadNetwork()` для гетерогенного плагина

### Для создания файлов .dot

```
ie = IECore()
ie.set_config( config={'HETERO_DUMP_GRAPH_DOT' : 'YES'}, device_name='HETERO')
```

Для просмотра изображений можно использовать утилиту GraphViz* или конвертер файлов. В операционной системе Ubuntu* вы можете использовать xdot:

- `sudo apt-get install xdot`
- `xdot hetero_subgraphs.dot`

Вы можете использовать данные о производительности (в примерах приложений это опция `-pc`), чтобы получить данные о производительности каждого подграфа.

Вот пример вывода для Googlenet v1, работающего на HDDL с откатом на CPU:

```
subgraph1: 1. предварительная обработка входных данных (средние
данные/HDDL):EXECUTED layerType:
realTime: 129cpu   : 129 execType:
subgraph1: 2. передача входных данных в DDR:EXECUTEDlayerType :
realTime: 201cpu          : 0execType                          :
subgraph1: 3. Время       выполнения HDDL:EXECUTEDlayerType :
realTime: 3808 cpu: 0execType                                  :
subgraph1: 4. передача вывода из DDR:EXECUTEDlayerType         :
realTime: 55cpu : 0execType                                    :
subgraph1: 5. Постпроцессинг                                   вывода
HDDL:EXECUTEDlayerType    : realTime: 7cpu                     :
7execType       :         execType:
subgraph1: 6. скопировать в IE blob:EXECUTEDlayerType          :
realTime: 2cpu  : 2       NOT_RUN                              layerType: Выход
subgraph2: prob:          realTime: 0cpu  : 0
                          execType: unknown
subgraph2: out prob:          EXECUTED                         layerType: SoftMax
realTime: 10    процесс   execType: ref
                cpu: 10
Общее время: 4212
микросекунд
```

## См. также

[Поддерживаемые устройства]

Создан с использованием
Sphinx 3.2.1.