Инструкция по воспроизведению тестов MLPerf Inference GMNT, MLPerf Inference ResNet, Kaldi

Общие сведения и требования к системе

Для проведения всех этапов тестирования система должна соответствовать следующим требованиям:

1. В системе должны быть установлены только графические ускорители NVIDIA TESLA T4
2. Пользователь, осуществляющий тестирование, должен иметь возможность повышать привилегии до суперпользователя «**root»** через «**sudo»**
3. Операционная система на базе Ubuntu 18.04 LTS или RHEL 7.

Генератор нагрузки

Рекомендуется запускать генератор нагрузки на том же узле, где и проводятся тесты. Создаваемая им нагрузка на локальном узле меньше, чем вероятные помехи и нагрузка при запуске на удаленном хосте.

Хорошим критерием является наличие ядер CPU, загруженных менее чем на 20% во время теста.

Параметры команды «make» в MLPerf тестах

Действия для запуска тестов MLPerf содержат команды автоматизации следующего вида:

```
make run RUN_ARGS="…"
```

Данная команда производит компиляцию TensorRT engine и затем запускает генератор нагрузки и прочие элементы. При необходимости, данные действия можно разбить на два этапа: компиляция и генератор нагрузки. Далее приведены инструкции, как выполнять каждый этап раздельно.

Для выполнения этапа компиляции необходимо выполнить следующую команду:

```
make generate_engines
```

Для запуска генератора нагрузки при условии, что TensorRT engine уже скомпилированы, необходимо выполнить следующую команду:

```
make run_harness
```

По умолчанию, переменная **«RUN_ARGS»** содержит параметр, отвечающий за проверку и

производительности и скорости:

```
--test_mode=SubmissionRun
```

Для измерения производительности следует указать

```
--test_mode=PerformanceOnly
```

Для измерения аккуратности следует указать

```
--test_mode=AccuracyOnly
```

Первичная настройка системы

Настройка системы на базе Ubuntu 18.04 LTS

Перед началом проведения работ по тестированию необходимо произвести установку необходимых компонентов и первичную настройку:

1. Отключаем драйвер **«nouveau»**, согласно инструкции: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html#runfile-nouveau-ubuntu

   ```
   #runfile-nouveau-ubuntu
   ```

2. Устанавливаем драйвер NVIDIA GPU версии 440 и перезагружаем систему

   ```
   $ sudo add-apt-repository ppa:graphics-drivers/ppa
   $ sudo apt-get update
   $ sudo apt-get install nvidia-driver-418\\
   $ sudo reboot
   ```

3. Устанавливаем **«docker»** версии 19.03, согласно инструкции: https://docs.docker.com/install/linux/docker-ce/ubuntu/
4. Устанавливаем **«docker-compose»** версии 1.25.4, согласно инструкции: https://docs.docker.com/compose/install/#install-compose-on-linux-systems
5. Устанавливаем пакет **«git»**

   ```
   $ sudo apt install -y git
   ```

6. Устанавливаем пакет **«nvidia-container-runtime»** и перезапускаем службу **«docker»**

   ```
   $ curl -s -L \
   https://nvidia.github.io/nvidia-container-runtime/gpgkey | \
   sudo apt-key add –
   $ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
   $ curl -s -L \
   https://nvidia.github.io/nvidia-container-runtime/$distribution/nvidia-
   container-runtime.list | \
   sudo tee /etc/apt/sources.list.d/nvidia-container-runtime.list
   ```

```
$ sudo apt-get update
$ sudo apt install -y nvidia-container-runtime
$ sudo systemctl restart docker
```

Настройка системы на базе RHEL 7

Перед началом проведения работ по тестированию необходимо произвести установку необходимых компонентов и первичную настройку:

1. Устанавливаем драйвер NVIDIA GPU версии 440, согласно инструкции https://docs.nvidia.com/datacenter/tesla/tesla-installation-notes/index.html#unique_1068237798
2. Устанавливаем **«docker»** версии 19.03, согласно инструкцииhttps://docs.docker.com/install/linux/docker-ce/centos/
3. Устанавливаем **«docker-compose»** версии 1.25.4, согласно инструкции: https://docs.docker.com/compose/install/#install-compose-on-linux-systems
4. Устанавливаем пакет **«git»**

```
$ sudo yum install -y git
```

5. Устанавливаем пакет **«nvidia-container-runtime»** и перезапускаем службу **«docker»**

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -k -L \
https://nvidia.github.io/nvidia-container-runtime/$distribution/nvidia-container-runtime.repo | \\\
sudo tee /etc/yum.repos.d/nvidia-container-runtime.repo
$ sudo yum install -y nvidia-container-runtime\\
$ sudo systemctl restart docker
```

Настройки окружения запуска тестов

Для завершения первичной настройки необходимо выполнить следующие шаги:

1. Устанавливаем по умолчанию в значение **«nvidia»** docker runtime.
    1. Модифицируем файл

       /etc/docker/daemon.json

```
{
    "default-runtime":"nvidia",
    "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
```

```
            "runtimeArgs": []

        }

    }

}
```

2. При необходимости изменения места хранения образов docker следует добавить в файл:

```
 "data-root": "/path/to/docker/images"
```

2. После чего сервис **«docker»** должен быть перезагружен

```
 $ sudo systemctl restart docker
```

3. Переводим все GPU в режим **«persistent mode»** и отключаем ECC в памяти

```
$ sudo systemctl --now enable nvidia-persistenced\\
$ sudo nvidia-smi -e 0\\
$ sudo shutdown -r 0
```

4. Устанавливаем максимальные частоты работы GPU и памяти
    1. Для получения максимальной частоты памяти

```
$ nvidia-smi -i 0 -q -d SUPPORTED_CLOCKS | grep Memory | head -n 1
```

    2. Для получения максимальной частоты GPU

```
 $ nvidia-smi -i 0 -q -d SUPPORTED_CLOCKS |grep Graphics |head -n
1
```

    3. Задаем максимальные частоты

```
 $ sudo nvidia-smi -ac <частота памяти в МГц>,<частота GPU в МГц>
```

5. Пример получения и установки частот GPU

```
$ nvidia-smi -i 0 -q -d SUPPORTED_CLOCKS|grep Memory | head -n 1
Memory : 3003 MHz

$ nvidia-smi -i 0 -q -d SUPPORTED_CLOCKS|grep Graphics| head -n 1
Graphics : 1531 MHz

$ sudo nvidia-smi -ac 3003,1531\\
Applications clocks set to "(MEM 3003, SM 1531)" for GPU
00000000:03:00.0
All done.
```

Подготовка компонентов для запуска тестов

На данном этапе производятся подготовительные работы по созданию основным компонентов повторяющихся тестов:

1. Создаем папку **«compose»**

```
$ mkdir -p ~/mlperf_gnmt/compose
```

2. В папку **«~/mlperf_gnmt/compose»** должны быть скачаны все файлы из папки https://drive.google.com/open?id=1WUpUliRw75pBnJ-lmOgnsx_ruPPAgsGU
3. Переходим в общий каталог теста **«mlperf_gnmt»**

```
$ cd ~/mlperf_gnmt/
```

4. Клонируем основной репозиторий **«mlperf inference results»**

```
$ git clone https:%%//%%github.com/mlperf/inference_results_v0.5.git
```

5. Заменяем оригинальный Dockerfile

```
$ cp compose/Dockerfile.original.replacement
inference_results_v0.5/closed/NVIDIA/docker/Dockerfile
```

6. Переходим в папку

```
$ cd compose
```

7. Произвести сборку docker-образа

```
$ sudo docker-compose build
```

    1. При необходимости авторизации следует зарегистрироваться и получить API ключ и следовать инструкции https://ngc.nvidia.com/setup
    2. Если возникают ошибки проверки SSL сертификатов, необходимо добавить в inference_results_v0.5/closed/NVIDIA/docker/Dockerfile следующее:

```
ENV GIT_SSL_NO_VERIFY\\
RUN echo "check_certificate = off" >> ~/.wgetrc
```

Проведение тестирования для режима «Offline»

Для воспроизведения тестов в режиме **«Offline»** рекомендуется использовать эталонную конфигурацию оборудования с четырьмя, восьмью или двадцатью GPU NVIDIA Tesla T4. Полное описание систем доступно по ссылкам в таблице:

4 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/DellEMC/systems/R740_T4x4_tensorrt.json

8 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/NVIDIA/systems/T4x8.json
20 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/NVIDIA/systems/T4x20.json

Эталонные результаты теста MLPerf Inference для систем выше приведены в следующей таблице (полная таблица доступна по ссылке: https://mlperf.org/inference-results/)

	ResNet	**GNMT**
4 x Tesla T4	22438.00	1417.62
8 x Tesla T4	44977.80	2834.75
20 x Tesla T4	113592.00	7154.88

Для определения удельных результатов системы на одном GPU, в следующей таблице приведены результаты тестов, разделённые на количество GPU:

	ResNet	**GNMT**
4 x Tesla T4	5609.50	354.41
8 x Tesla T4	5622.23	354.34
20 x Tesla T4	5679.60	357.74

Таким образом наблюдается почти линейное масштабирование результатов бенчмарка с количеством GPU. Аппаратные отличия тестируемых конфигураций, от конфигураций выше, могут приводить к снижению производительности до 30%.

По завершению каждого теста необходимо остановить контейнер, используя следующие действия:

1. Переключиться в терминал, где запущена следующая команда

```
$ sudo docker-compose up
```

2. Прервать её исполнение нажав сочетание клавиш **«CTRL+C»**
3. Остановить контейнер, используя следующую команду

```
$ sudo docker-compose down
```

Тест «MLPerf Inference GNMT»

Для проведения теста «MLPerf Inference GNMT» в режиме «Offline» следует выполнить следующие действия:

1. Запускаем docker-образ

```
$ sudo docker-compose up
```

2. Открываем дополнительный терминал и переходим в папку

```
**«~/mlperf_gnmt/compose»**.
```

```
$ cd ~/mlperf_gnmt/compose
```

3. Подключаемся к контейнеру

```
 $ sudo docker-compose exec mlperf-gnmt /bin/bash
```

4. Скачиваем датасет, используя готовый скрипт

```
$ ./download_dataset.sh
```

5. Производим запуск теста

```
 make run RUN_ARGS="--benchmarks=gnmt --scenarios=Offline --
test_mode=SubmissionRun"
```

По окончанию тестирования будет выдан результат исполнения. В качестве единицы измерения принимается показатель «*количество сэмплов в секунду*». Будет выведено сообщение о статусе тестирования на точность.

По завершению всех операций останавливаем контейнер, как указано в разделе Проведение тестирования для режима «Offline».

Тест «MLPerf Inference ResNet»

Для проведения теста «MLPerf Inference ResNet» в режиме «Offline» следует выполнить следующие действия:

1. Создаём папку **/tmp/preprocessed_data**

```
$ mkdir -p /tmp/preprocessed_data
```

    1. Перед проведением тестирования система преобразует датасет в другой формат. Эта папка будет использоваться для хранения датасета в преобразованном формате. Убедитесь, что на диске, где хранится эта папка есть 20 ГБ свободного места.
2. Скопируем файлы с **ILSVRC2012_val_00000001.JPEG** по **ILSVRC2012_val_00050000.JPEG** из валидационного датасета **«Imagenet 2012»** в папку **/mnt/hdd/datasets/imagenet.**
    1. Получить файлы можно, пройдя регистрацию и получив одобрение от авторов датасета, по адресу: http://image-net.org/download-images
3. Переходим в общий каталог теста **«~/mlperf_gnmt/compose»**

```
 $ cd ~/mlperf_gnmt/compose
```

4. Открываем файл **«docker-compose.yml»** любым удобным редактором
5. Добавляем в раздел **«volumes»** следующие разделы

```
- /tmp/preprocessed_data:/work/build/preprocessed_data
- "/mnt/hdd/datasets/imagenet:/work/build/data/imagenet"
```

6. Сохраняем и закрываем файл.

7. Запускаем docker-образ

```
$ sudo docker-compose up
```

8. Открываем дополнительный терминал и переходим в папку **«~/mlperf_gnmt/compose»**.

```
$ cd ~/mlperf_gnmt/compose
```

9. Подключаемся к контейнеру

```
$ sudo docker-compose exec mlperf-gnmt /bin/bash
```

10. Переходим в папку **«/work»** внутри контейнера

```
$ cd /work
```

11. Производим подготовку данных для проведения тестирования

```
$ python3 scripts/preprocess_data.py
-d build/data -o build/preprocessed_data \
-b resnet --val_only -t fp32
$ python3 scripts/preprocess_data.py \
-d build/data -o build/preprocessed_data \
-b resnet --val_only
```

12. Производим запуск теста

```
$ make run RUN_ARGS="--benchmarks=resnet --scenarios=Offline --
test_mode=SubmissionRun "
```

По окончанию тестирования будет выдан результат исполнения. В качестве единицы измерения принимается показатель «*количество сэмплов в секунду*». Будет выведено сообщение о статусе тестирования на точность.

По завершению всех операций останавливаем контейнер, как указано в разделе Проведение тестирования для режима «Offline».

Проведение тестирования для режима «Server»

Для воспроизведения тестов в режиме **«Server»** рекомендуется использовать эталонную конфигурацию с четырьмя, восьмью или двадцатью GPU NVIDIA Tesla T4. Полное описание систем доступно по ссылкам в таблице:

4 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/DellEMC/systems/R740_T4x4_tensorrt.json
8 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/NVIDIA/systems/T4x8.json
20 x Tesla T4	https://github.com/mlperf/inference_results_v0.5/blob/master/closed/NVIDIA/systems/T4x20.json

Эталонные результаты теста MLPerf Inference для систем выше приведены в следующей таблице (полная таблица доступна по ссылке: https://mlperf.org/inference-results/)

	ResNet	**GNMT**
4 x Tesla T4	20742.83	828.57
8 x Tesla T4	41546.64	1581.20
20 x Tesla T4	103532.10	3776.07

Для определения удельных результатов системы на одном GPU, в следующей таблице приведены результаты тестов, разделённые на количество GPU:

	ResNet	**GNMT**
4 x Tesla T4	5185.71	207.14
8 x Tesla T4	5193.33	197.65
20 x Tesla T4	5176.61	188.80

Таблица 1. Удельные показатели системы на одном GPU

Таким образом наблюдается почти линейное масштабирование результатов бенчмарка с количеством GPU. Аппаратные отличия тестируемых конфигураций, от конфигураций выше, могут приводить к снижению производительности до 30%.

По завершению каждого теста необходимо остановить контейнер, используя следующие действия:

1. Переключиться в терминал, где запущена следующая команда

```
$ sudo docker-compose up
```

2. Прервать её исполнение нажав сочетание клавиш **«CTRL+C»**
3. Остановить контейнер, используя следующую команду

```
$ sudo docker-compose down
```

Валидность результатов тестирования в режиме «Server»

Тестирование в режиме «Server» подразумевает, что результаты валидны только если от постановки каждого сэмпла в очередь до поучения ответа проходило бы меньше времени, чем заданное пороговое значение, которое можно найти в таблице

> https://github.com/mlperf/inference_policies/blob/master/inference_rules.adoc#41-benchmarks

Основные параметры, влияющие на прохождение теста системой, описаны в данной секции, **Таблица 2. Файл gnmt/Server/config.json** и **Таблица 4. Файл resnet/Server/config.json.** Дополнительные параметры также могут иметь влияние на производительность и на валидность бенчмарка. Более подробное описание влияния параметров на производительность и валидность теста, и алгоритм подбора оптимальных содержатся в документе https://github.com/mlperf/inference_results_v0.5/blob/master/closed/NVIDIA/performance_tuning_guid

e.adoc

Валидность результатов обозначается словом **«VALID»** в выводе, невалидность словом **«INVALID»**. Для получения валидных результатов необходимо задать подходящие для системы параметры тестирования.

Основной параметр, влияющий на валидность результатов — «**server_target_qps»**, целевое количество запросов в секунду, которое будет отсылать генератор нагрузки. Частично код, который уточняет это значение с помощью бинарного поиска инкорпорирован в сам генератор нагрузки LoadGen[1)], однако результаты теста зависят от исходного значения.

Получение валидных результатов не гарантирует, что получены максимальные возможные результаты тестирования, так как они зависят от многих параметров, указанных в конфигурационных файлах, и при некоторых комбинациях могут быть валидными, но заниженными. Заниженными считаются результаты, при которых можно увеличить значение «**server_target_qps»** на 5% и подобрать такую конфигурацию параметров, что пять запусков теста подряд выдадут валидные результаты.

Тест «MLPerf Inference GNMT»

1. Выполняем действия 1-10 секции **Тест «MLPerf Inference GNMT»** в части **Проведение тестирования для режима «Offline».**
2. Запускаем тест командой

```
 make run RUN_ARGS="--benchmarks=gnmt --scenarios=Server --
test_mode=SubmissionRun "
```

3. **По окончанию тестирования будет выдан результат исполнения. В качестве единицы измерения принимается показатель «*количество сэмплов в секунду*». Будет выведено сообщение о том, успешно ли система прошла тест на точность и сообщение, валидны результаты или невалидны.**

Если результаты невалидны, изменим параметр **«server_target_qps»** и иные параметры в конфигурационных файлах. Основные параметры, влияющие на результат описаны в таблицах ниже.

Параметр	**Ключ**	**Описание**
server_target_qps	gnmt → server_target_qps	Целевое значение количества обрабатываемых семплов в секунду. Если указано слишком большое значение, не все запросы будут вовремя выполнены и результат тестирования окажется невалидным. Если указано слишком маленькое значение, результат тестирования будет валиден, но занижен. Изначальное значение рекомендуется выбирать по **Таблица 1. Удельные показатели системы на одном GPU.** Частично код, который уточняет это значение с помощью бинарного поиска инкорпорирован в сам генератор нагрузки LoadGen[2)], однако результаты теста зависят от исходного значения.

Параметр	Ключ	Описание
batch_sizes	gnmt → batch_sizes	Количество сэмплов, которые собираются в один батч для отправки на GPU. Могут компилироваться несколько возможных **batch_size** одновременно.
concurrency	gnmt → concurrency	Количество одновременно исполняющихся на GPU батчей инференса.
precision	gnmt → precision	Аппаратная точность вычислений.

Таблица 2. Файл gnmt/Server/config.json

Параметр	Ключ	Описание
server_target_qps	*.Server.target_qps	Значение необходимо устанавливать в точности равным значению в файле **gnmt/Server/config.json.** Смотри Таблица 2. Файл gnmt/Server/config.json
Остальные параметры		Заданы правилами MLPerf и не подлежат изменению.

Таблица 3. Файл gnmt/Server/user.conf

Для редактирования файлов:

1. Открываем дополнительный терминал и переходим в папку «**~/mlperf_gnmt/compose**».

```
$ cd ~/mlperf_gnmt/compose
```

2. Подключаемся к контейнеру

```
$ sudo docker-compose exec mlperf-gnmt /bin/bash
```

3. Открываем на редактирование json-файл любым удобным редактором /work/measurements/T4x8/gnmt/Server/config.json
4. Изменяем параметры согласно **Таблица 2. Файл gnmt/Server/config.json**.
5. Сохраняем и закрываем файл.
6. Открываем на редактирование conf-файл любым удобным редактором “/work/measurements/T4x8/gnmt/Server/user.conf”
7. Изменяем параметры согласно **Таблица 3. Файл gnmt/Server/user.conf**.
8. Сохраняем и закрываем файл.
9. - Изменение иных параметров в этих конфигурационных файлах тоже влияет на результаты теста. Для более тонкой настройки необходимо обратиться к документации теста MLPerf Inference, указанной в **Валидность результатов тестирования в режиме «Server».**
10. Перезапустим тест командой из пункта 1. Если результаты теста всё ещё невалидны, повторим процедуру 1 - 10.
11. Если результаты валидны, проверим, что они не занижены:
    1. Увеличим значение «**server_target_qps»** на 5% в конфигурационных файлах.
    2. Запустим тест пять раз подряд как в пункте 1 и посмотрим, выдала ли система хотя бы раз невалидные результаты.
    3. Если система выдала валидные результаты все пять раз, предыдущие результаты были занижены. Повторим пункты 1 - 11.
    4. Если система выдала невалидные результаты хотя бы один раз, изменим параметры, зафиксировав «**server_target_qps»** согласно пунктам 4 - 9 и повторим

действия из пункта 11 b

Тест «MLPerf Inference ResNet»

1. Выполняем все действия 4 из секции **Тест «MLPerf Inference ResNet»** в части **Проведение тестирования для режима «Offline».**
2. После этого необходимо запустить тест командой

```
make run RUN_ARGS="--benchmarks=resnet --scenarios=Server --
test_mode=SubmissionRun "
```

По окончанию тестирования будет выдан результат исполнения. В качестве единицы измерения принимается показатель «*количество сэмплов в секунду*». Будет выведено сообщение о том, успешно ли система прошла тест на точность и сообщение, валидны результаты или невалидны.

Если результаты невалидны, изменим параметр **«server_target_qps»** и иные параметры в конфигурационных файлах. Основные параметры, влияющие на результат описаны в таблицах ниже.

Параметр	**Ключи**	**Описание**
server_target_qps	resnet → server_target_qps	Целевое значение количества обрабатываемых семплов в секунду. Если указано слишком большое значение, не все запросы будут вовремя выполнены и результат тестирования окажется невалидным. Если указано слишком маленькое значение, результат тестирования будет валиден, но занижен. Изначальное значение рекомендуется выбирать по **Таблица 1. Удельные показатели системы на одном GPU.** Частично код, который уточняет это значение с помощью бинарного поиска инкорпорирован в сам генератор нагрузки LoadGen[3], однако результаты теста зависят от исходного значения.
deque_timeout_us	resnet → deque_timeout_us	Только для бенчмарка resnet. Число миллисекунд, после которого на GPU в любом случае отправляется батч, даже если он ещё не заполнился запросами целиком
gpu_batch_size	resnet → gpu_batch_size	Количество сэмплов, которые собираются в один батч для отправки на GPU.
gpu_inference _streams	resnet → gpu_inference _streams	Количество одновременно исполняющихся на GPU батчей инференса.

Таблица 4. Файл resnet/Server/config.json

Параметр	**Ключ**	**Описание**
server_target_qps	*.Server.target_qps	Значение необходимо устанавливать в точности равным значению в файле resnet/**Server/config.json.** Смотри **Таблица 4. Файл resnet/Server/config.json**

Параметр	Ключ	Описание
Остальные параметры		Заданы правилами MLPerf и не подлежат изменению.

Таблица 5. Файл resnet/Server/user.conf

Для редактирования файлов:

1. Открываем дополнительный терминал и переходим в папку **«~/mlperf_gnmt/compose»**.

```
$ cd ~/mlperf_gnmt/compose
```

2. Подключаемся к контейнеру

```
$ sudo docker-compose exec mlperf-gnmt /bin/bash
```

3. Открываем на редактирование json-файл любым удобным редактором /work/measurements/T4x8/resnet/Server/config.json
4. Изменяем параметры согласно **Таблица 4. Файл resnet/Server/config.json.**
5. Сохраняем и закрываем файл.
6. Открываем на редактирование conf-файл любым удобным редактором /work/measurements/T4x8/resnet/Server/user.conf
7. Изменяем параметры согласно **Таблица 5. Файл resnet/Server/user.conf**.
8. Сохраняем и закрываем файл.
9. Изменение иных параметров в этих конфигурационных файлах тоже влияет на результаты теста. Для более тонкой настройки необходимо обратиться к документации теста MLPerf Inference, указанной в **Валидность результатов тестирования в режиме «Server».**
10. Перезапустим тест командой из пункта 1. Если результаты теста всё ещё невалидны, повторим процедуру 1 - 10.
11. Если результаты валидны, проверим, что они не занижены:
    1. Увеличим значение «**server_target_qps»** на 5% в конфигурационных файлах.
    2. Запустим тест пять раз подряд как в пункте 1 и посмотрим, выдала ли система хотя бы раз невалидные результаты.
        1. Если система выдала валидные результаты все пять раз, предыдущие результаты были занижены. Повторим пункты 1 - 11.
        2. Если система выдала невалидные результаты хотя бы один раз, изменим параметры, зафиксировав «**server_target_qps»** согласно пунктам 4 - 9 и повторим действия из пункта 11 b

Тест «Kaldi»

Для выполнения теста «Kaldi» необходимо проделать следующие действия:

1. Запустить контейнер из репозитория NGC на локальном узле

```
 $ docker run --gpus all -it --ipc=host --name kaldi
nvcr.io/nvidia/kaldi:20.03-py3
```

2. После успешного скачивания и запуска образа автоматически пользователь попадет внутрь контейнера

3. Следует перейти в рабочую директорию теста

```
$ cd /workspace/nvidia-examples/librispeech/
```

4. Необходимо выполнить скрипт предварительной подготовки **«prepare.sh»**

```
$ ./prepare.sh
```

    1. Для запуска теста необходимо выполнить одну из следующих команд:
        1. Если используется один графический процессор в сервере

```
$ ./run_benchmark.sh
```

        2. Если используется несколько графических процессоров в сервере

```
$ ./run_multigpu_benchmark.sh
```

По окончанию тестирования будет выдан результат тестирования в режиме полной загрузки. В качестве единицы измерения пропускной способности принимается показатель «ускорение относительно реального времени» (RTF, Real Time Factor), который измеряет отношение между длительностью всех распознанных во время бенчмарка фраз и временем, затраченным на их распознавание. В качестве единицы измерения точности применяется WER (Word Error Rate) — среднее нормированное расстояние Левенштейна в словах между сказанными и предсказанными фразами. Будет выведено сообщение о том, успешно ли система прошла тест на точность и сообщение, валидны результаты или невалидны.

Проведём в том же контейнере тест «Kaldi» в онлайн сценарии, когда на сервер генерируется динамическая нагрузка и измеряется не только пропускная способность, но и задержки. Результаты теста зависят от параметра **«ONLINE_NUM_PARALLEL_STREAMING_CHANNELS» —** целое число, количество одновременно транслируемых на сервер каналов. Значение этого параметра необходимо вычислить по формуле

```
числоканалов > floor(RTF \ast 0.8)$**,**
```

где RTF ускорение относительно реального времени, измеренное в пункте 5.

1. Для запуска теста необходимо выполнить одну из следующих команд:
    1. Если используется один графический процессор в сервере

```
 $ ONLINE=1 ONLINE_NUM_PARALLEL_STREAMING_CHANNELS=<число каналов> \
./run_benchmark.sh
```

    2. Если используется несколько графических процессоров в сервере

```
 $ ONLINE=1 ONLINE_NUM_PARALLEL_STREAMING_CHANNELS=<число каналов> \
./run_multigpu_benchmark.sh
```

По окончанию тестирования будет выдан результат исполнения. В качестве единицы измерения задержки принимается показатель «95-й перцентиль задержки» (Latency @ 95%), который измеряет время между отправкой фрагмента на распознавание и

получением распознанного фрагмента в секундах при количестве запросов, установленном на 80% от пропускной способности в режиме полной загрузки. Будет выведено сообщение о том, успешно ли система прошла тест на точность и сообщение, валидны результаты или невалидны.

Дополнительная информация

Мониторинг

Для получения наилучших результатов тестов рекомендуется проводить измерения без использования мониторинга.

При необходимости получения данных по мониторингу, рекомендуется производить запуск тестов в два прохода: с включенным мониторингом и без него.

Рекомендуется использовать **DCGM** для наблюдения за GPU. Документация с описанием процесса установки доступна по ссылке (скачать данный пакет можно по адресу: https://developer.nvidia.com/dcgm): https://docs.nvidia.com/datacenter/dcgm/latest/dcgm-user-guide/getting-started.html#installation

Для получения результатов мониторинга в формате **Prometheus**, рекомендуется использовать следующий репозиторий и код: https://github.com/NVIDIA/gpu-monitoring-tools/tree/master/dcgm-exporter

Система мониторинга **Zabbix** версии **4.2** и выше поддерживает сбор данных в формате **Prometheus**: https://www.zabbix.com/documentation/4.2/manual/config/i"tems/itemtypes/prometheus

Сбор метрических данных можно осуществлять при помощи утилиты **«nvidia-smi»** . Пример команды сбора различных метрик раз в секунду в csv файл ~/mlperf_gnmt/nvidia-smi-metrics.csv приведен ниже:

```
 nvidia-smi \--query-
gpu=timestamp,name,pci.bus_id,driver_version,pstate,pcie.link.gen.max,pcie.l
ink.gen.current,temperature.gpu,power.draw,power.limit,utilization.gpu,utili
zation.memory,memory.total,memory.free,memory.used --format=csv -l 1 | \

tee -a ~/mlperf_gnmt/nvidia-smi-metrics.csv
```

Интерактивный вариант отображения метрических данных можно производить следующей командой:

```
nvidia-smi dmon
```

[1] [2] [3]
https://github.com/mlperf/inference_policies/blob/master/inference_rules.adoc#51-loadgen-operation, пункт Server