如何在服务器中配置REFNeRF

服务器配置

```
$ uname -a
Linux admin1-SYS-7048GR-TR 4.4.0-21-generic #37-Ubuntu SMP Mon Apr 18 18:33:37
UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
```

NVIDIA GeForce GTX 1080 Ti × 3

跳过痛苦的摸索阶段

更新驱动

服务器三个1080一看就是比较古老的产品了，也不出意料地驱动版本比较低，最高支持10.2的CUDA。这里牵涉到了一个运行（runtime）版本和驱动（driver）版本的问题。这里**必须得更新到一个11.x的驱动**。为什么呢……这就是让我痛苦了几天的原因……

总之先从官网寻找对应的驱动安装。简单来说`wget`下载链接并运行即可。这个安装的过程可能会有很多问题，利用搜索引擎解决即可。另外一种方法是直接用命令行，利用`apt`来更新驱动，但是我在服务器上无法使用这个方法，获取到的驱动同样也很古老，因此最后还是从官网下载驱动。

更新过后如下：

```
$ nvidia-smi
Thu Aug 25 15:56:23 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.65.01    Driver Version: 515.65.01    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:02:00.0 Off |                  N/A |
| 28%   35C    P8     9W / 250W |     19MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  Off  | 00000000:82:00.0 Off |                  N/A |
| 29%   37C    P8     8W / 250W |      2MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  NVIDIA GeForce ...  Off  | 00000000:83:00.0 Off |                  N/A |
| 29%   36C    P8     9W / 250W |      2MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

更新gcc

使用命令行更新即可。实际使用中没有遇到关于gcc版本的报错，但是tensorflow似乎对gcc的版本做出了要求。在我意识到要求最高的其实不是jaxlib而是tf后，我升级了gcc。

升级后如下：

```
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-
3ubuntu1~16.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-
languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-
major-version-only --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --
enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-
gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap
--enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-
default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --
enable-libmpx --enable-plugin --with-system-zlib --with-target-system-zlib --
enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --
with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-
tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-
checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --
target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~16.04)
```

安装CUDA和cudnn

研究过jaxlib和tf的各种版本要求之后，我选择了安装11.2的CUDA和8.1的cudnn。CUDA和cudnn的版本之间没有什么必然要求和联系。安装流程如下：

- linux安装CUDA+cuDNN_Iron·Man的博客-CSDN博客_linux安装cuda和cudnn
- Linux下安装cuda和对应版本的cudnn*水哥很水的博客-CSDN博客*linux安装cudnn

其中cudnn的包是需要下载验证的，搞了半天没搞出来怎么用`wget`解决，因此直接用本地机器下载，下载完后用`scp`上传到服务器。安装后如下：

```
$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2021 NVIDIA Corporation
Built on Sun_Feb_14_21:12:58_PST_2021
Cuda compilation tools, release 11.2, V11.2.152
Build cuda_11.2.r11.2/compiler.29618528_0
```

配置仓库环境

从仓库中克隆代码到本地，跟着readme一直配置到`pip install -r requirements.txt`这一步。需要做出改变了。REFNeRF是基于谷歌开发的jax和jaxlib实现的，其中谷歌每发布一版jaxlib就要对应地发布一版jax。而jaxlib的版本是需要根据电脑的CUDA和cudnn版本进行选择的。这里我参考了JAX: 库安装和GPU使用，解决不能识别gpu问题*Papageno2018的博客-CSDN博客*安装jax。（这里安装不对的话，后面会报无法使用GPU只能使用CPU的警告）实际上，可以先故意输入一个不存在的版本，`pip`会提示有哪些jaxlib的版本可以安装。

安装后的版本如下：

```
flax                            0.6.0
jax                             0.3.16
jaxlib                          0.3.15+cuda11.cudnn805
tensorflow                      2.9.2
```

实际过程中，因为一开始服务器的驱动最多支持10.2，用10.2的话需要自己再找符合版本要求的jaxlib和jax，而其中另外一个包flax又对jax的版本作了要求。也就是说，如果使用的是10.2的CUDA，需要自己摸索flax、jax和jaxlib三个包的版本组合。

在新服务器上配置环境的时候，我本来想使用8.2的cudnn，**结果发现不行**，会报一个关于NCCL的错。看到这个错后，我马上将cudnn改为8.1版本，代码顺利地跑了起来。**这里应该还是tensorflow对cudnn的版本作出了要求的原因。**

这里还有一个问题，`run_all_unit_tests.sh`进行到数据集测试的时候，其正常运行与否与GPU的个数有关。如果直接运行脚本，则会提示`reshape()`函数出错。阅读报错信息，指定只使用两个GPU运行脚本则可以通过测试（理论上只用一个应该也可以）：`CUDA_VISIBLE_DEVICES=1,0 ./scripts/run_all_unit_tests.sh`。

至此，模块测试脚本应该可以全部通过。在这里，因为仓库限定了虚拟环境的python版本为3.9，这个较新的版本又对tensorflow的版本做了要求。因为tensorflow并不向下兼容，我是在解决完上述所有问题后才发现，10.2的CUDA应该是不能跑这个仓库代码的。**在更新驱动和换一个服务器之间我选择了更新驱动，并重新把上述过程走了一遍。**

数据

从Ref-NeRF首页下载shiny dataset和real dataset，分别保存到shiny和real目录下。创建一个`ref_results`保存训练结果。最后得到如下的文件结构：

```
/home/admin1/REFNeRF
├── multinerf
|   ├── configs
|   ├── internal
|   ├── __pycache__
|   ├── scripts
|   └── tests
├── real
├── ref_results
└── shiny
```

训练与渲染

稍微修改一下`train_shinyblender.sh`：

```
#!/bin/bash
# Copyright 2022 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

export CUDA_VISIBLE_DEVICES=1,2

SCENE=toaster
EXPERIMENT=shinyblender50w
# DATA_DIR=/usr/local/google/home/barron/tmp/nerf_data/dors_nerf_synthetic
#
CHECKPOINT_DIR=/usr/local/google/home/barron/tmp/nerf_results/"$EXPERIMENT"/"$SC
ENE"
DATA_DIR=~/REFNeRF/shiny/refnerf
CHECKPOINT_DIR=~/REFNeRF//ref_results/"$EXPERIMENT"/"$SCENE"

rm "$CHECKPOINT_DIR"/*
python -m train \
  --gin_configs=configs/blender_refnerf.gin \
  --gin_bindings="Config.data_dir = '${DATA_DIR}/${SCENE}'" \
  --gin_bindings="Config.checkpoint_dir = '${CHECKPOINT_DIR}'" \
  --logtostderr
```

稍微修改一下`eval_shinyblender.sh`：

```
#!/bin/bash
# Copyright 2022 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

export CUDA_VISIBLE_DEVICES=1,2
```

```
SCENE=toaster
EXPERIMENT=shinyblender
# DATA_DIR=/usr/local/google/home/barron/tmp/nerf_data/dors_nerf_synthetic
#
CHECKPOINT_DIR=/usr/local/google/home/barron/tmp/nerf_results/"$EXPERIMENT"/"$SC
ENE"
DATA_DIR=~/REFNeRF/shiny/refnerf
CHECKPOINT_DIR=~/REFNeRF//ref_results/"$EXPERIMENT"/"$SCENE"

python -m eval \
  --gin_configs=configs/blender_refnerf.gin \
  --gin_bindings="Config.data_dir = '${DATA_DIR}/${SCENE}'" \
  --gin_bindings="Config.checkpoint_dir = '${CHECKPOINT_DIR}'" \
  --logtostderr
```

根据readme的提示，直接进行训练或者渲染是有概率发生内存溢出错误（OOM）的。关于需要修改的地方，readme有误。readme写的应该是修改`debug.gin`中的`batch_size`，但是我查看了代码，似乎训练和渲染代码都没有用到这个文件。正确的修改之处是修改`internal/configs.py`中的配置：

```
class Config:
  """Configuration flags for everything."""
  ...
  batch_size: int = 2048  # The number of rays/pixels in each batch.   original:
16384
  ...
  render_chunk_size: int = 2048  # Chunk size for whole-image renderings.
original: 16384
  ...

  # Only used by train.py:
  max_steps: int = 500000  # The number of optimization steps.
  ...
```

在这里我已经修改成了最大的`batch_size`。

在命令行中输入以下命令即可完成训练和渲染，其中渲染会生成包括但不限于渲染图和法向量图等图片：

```
cd multinerf
conda activate multinerf
nohup ./scripts/train_shinyblender.sh &>../ref_results/terminal/2048+50w &
nohup ./scripts/eval_shinyblender.sh &>../ref_results/terminal/2048+50w+eval &
```