conda build 系列教程·deepmd 的构建①|deepmd-kit

https://zhuanlan.zhihu.com/p/189019413

None Mon Jun, 21 00:12

此前，本专栏介绍了 conda install 的使用方法，以及 conda install 安装 deepmd-kit 的方法。但是，目前对 conda build 构建 conda packages 的讨论甚少。授之以鱼不如授之以渔，这一系列将以 deepmd-kit 的 packages 为例，介绍 conda 包的构建方法。

本文先介绍 python 包的构建配方（recipe）。在 deepmd-kit-feedstock 这一项目中，`recipe` 文件夹下包括 `conda_build_config.yaml` 和 `meta.yaml` 两个 YAML 文件，其中 `conda_build_config.yaml` 包含了构建变量，而 `meta.yaml` 包含了构建的元信息。

我们来看一看 `conda_build_config.yaml` 这个文件：

```
float_prec:
- low
- high
channel_sources:
- deepmodeling,defaults
channel_targets:
- deepmodeling main
c_compiler_version:
- 4  # [osx]
cxx_compiler_version:
- 4  # [osx]
numpy:
- 1.18
cuda_compiler_version:
- None
- 9.2
- 10.0
- 10.1
python:
- 3.6
- 3.7
- 3.8
```

这里，`float_prec` 表示程序是用低精度还是高精度编译的，`channel_sources` 和 `channel_targets` 表示 channel 的来源和目标，剩下的则是版本号。这里，`float_prec`、`cuda_compiler_version` 和 `python` 实质上构成了编译矩阵，一共有 $2 \times 4 \times 3 = 24$ 个序列。

我们再来看 `meta.yaml`，从头部看起：

```
{% set name = 'deepmd-kit' %}
{% set version = '1.2.0' %}
{% set py = environ.get('CONDA_PY', '') %}

{% if cuda_compiler_version == 'None' %}
{% set dp_variant = 'cpu' %}
{% set tf_version = '2.2' %}

{% else %}
{% set dp_variant = 'gpu' %}

{% if cuda_compiler_version == '10.1' %}
{% set tf_version = '2.2' %}
{% elif cuda_compiler_version == '10.0' %}
{% set tf_version = '2.0' %}
{% else %}
{% set tf_version = '1.14' %}
{% endif %}

{% endif %}
```

这是 jinja2 语法的运用。前三行定义了这一文件需要用到的变量。之后的若干行，则对 cuda_compiler_version 进行了选择，如果是 None，则认为编译的是 CPU 版本，使用 TensorFlow 2.2 作为依赖；如果 CUDA 版本分别是 10.1、10.0、9.2，则根据 TensorFlow 各版本支持的 CUDA 版本，分别选择 2.2、2.0 和 1.14 作为依赖。

```
package:
  name: {{ name|lower }}
  version: {{ version }}
```

这一部分定义了 package 的名称和版本，我们直接使用之前已经设置好的变量。

```
source:
  git_url: https://github.com/deepmodeling/deepmd-kit
  git_rev: 567bcff2a99ab50b9b1ba5066575da1a99a4c1b8
  patches:
  - low_prec.diff  # [float_prec == 'low']
  - osx.patch  # [osx]
```

这一部分是源代码的来源，`git_url` 和 `git_rev` 表示源代码来自的 git 项目的链接和分支，而 patches 表示对程序打补丁。值得注意的是，`low_prec.diff` 的后面有 `# [float_prec == 'low']` 的标记，表示仅当 `float_prec` 变量设置为 `low` 时，才打上 `low_prec.diff` 这个补丁。我们看看 `low_prec.diff` 的内容：

```
diff --git a/setup.py b/setup.py
index 2f56794..c62ce76 100644
--- a/setup.py
+++ b/setup.py
@@ -55,7 +55,7 @@ setup(
     cmake_args=['-DTENSORFLOW_ROOT:STRING=%s' % tf_install_dir,
                 '-DBUILD_PY_IF:BOOL=TRUE',
                 '-DBUILD_CPP_IF:BOOL=FALSE',
-                '-DFLOAT_PREC:STRING=high',
+                '-DFLOAT_PREC:STRING=low',
     ],
     cmake_source_dir='source',
     cmake_minimum_required_version='3.0',
```

原来，这就是一个典型的 diff 文件，对 `setup.py` 里的某行进行了修改，把 cmake 的 `FLOAT_PREC` 参数从 `high` 改成了 `low`。之后的 `osx.patch` 有着类似的作用。

我们接着来看 `meta.yaml` 的下一个部分。

```
build:
  number: 2
  string: 'py{{ py }}_{{ PKG_BUILDNUM }}_cuda{{ cuda_compiler_version }}_{{ dp_variant }}'  # [float_prec == 'high']
  string: 'py{{ py }}_{{ PKG_BUILDNUM }}_cuda{{ cuda_compiler_version }}_{{ dp_variant }}_{{float_prec}}'  # [float_prec != 'high']
  script:
    - 'SETUPTOOLS_SCM_PRETEND_VERSION={{ version }} {{ PYTHON }} -m pip install . -vv'  # [unix]
    - 'set SETUPTOOLS_SCM_PRETEND_VERSION=%PKG_VERSION%'  # [win]
    - 'pip install . --no-deps -vv'  # [win]
  skip: true  # [osx and cuda_compiler_version != 'None']
  skip: true  # [not linux]
  skip: true  # [py==38 and cuda_compiler_version == '9.2' ]
  skip: true  # [py==38 and cuda_compiler_version == '10.0' ]
```

这一部分是构建信息，`number` 表示该版本程序的构建版本号；`string` 表示构建出来的 package 名称，我们为不同的 `float_prec` 设置了不同的名称，将 python 版本、构建版本、CUDA 版本、CPU 还是 GPU、高精度还是低精度全部包含在内。

`script` 表示执行什么脚本可以构建程序，[之前笔者介绍过](#)，deepmd-kit 的 python 包可以用 `pip` 直接安装：

```
pip install git+htts://github.com/deepmodeling/deepmd-kit
```

这里正是用 `pip` 直接从源代码安装。将 `skip` 设为 `true`，又在后面设置了若干条件语句，表示在这些情况下跳过构建。例如，我们目前之构建 Linux 环境下的程序，当 CUDA 为 9.2 和 10.0 时，我们跳过 Python 3.8 的构建，因为没有对应的 TensorFlow。

```
requirements:
  build:
    - {{ compiler('c') }}
    - {{ compiler('cxx') }}
    - cmake  >=3.7
    - make  # [unix]
    - ninja

  host:
    - python
    - pip
    - numpy
    - setuptools_scm
    - tensorflow {{ tf_version }}*  # [cuda_compiler_version == 'None']
    - tensorflow-gpu {{ tf_version }}*  # [cuda_compiler_version != 'None']
    - cudatoolkit {{ cuda_compiler_version }}*  # [cuda_compiler_version != 'None']
    - scikit-build
    - m2r

  run:
    - python
    - numpy
    - scipy
    - {{ pin_compatible('tensorflow', max_pin='x.x') }}  # [cuda_compiler_version == 'None']
    - {{ pin_compatible('tensorflow-gpu', max_pin='x.x') }}  # [cuda_compiler_version != 'None']
    - {{ pin_compatible('cudatoolkit', max_pin='x.x') }}*  # [cuda_compiler_version != 'None']
```

下一部分就是依赖信息了，分别 `build`、`host` 和 `run` 三块。`build` 和 `host` 都是构建时需要的模块，但与 `build` 不同的是，`host` 可以添加共享库，可以让 package 跨平台使用。`run` 部分则是 package 运行时需要的依赖，安装 package 时将自动安装这些依赖。
`{{ compiler('c') }}` 和 `{{ compiler('cxx') }}` 将自动选择合适的编译器；`tf_version` 和 `cuda_compiler_version` 我们之前已经设置好，可以根据不同的 CUDA 版本自动选择相应的依赖；`pin_compatible` 则表示程序运行时的依赖版本，和构建时的依赖版本，在 `x.x` 版本号上保持一致。例如，构建时使用 CUDA 10.1 作为依赖，则运行时同样需要选择 10.1 作为依赖。

```
test:
  imports:
    - deepmd
  commands:
    - dp -h
```

`test` 部分比较简单，`imports` 是针对 Python 包的，而 commands 则针对 bash 命令。这一测试用于检测构建的 conda package 有无问题。

```
about:
  home: https://github.com/deepmodeling/deepmd-kit
  license: LGPL-3.0
  license_family: LGPL
  license_file: LICENSE
  summary: 'A deep learning package for many-body potential energy representation and molecular dynamics'
  doc_url: https://github.com/deepmodeling/deepmd-kit
  dev_url: https://github.com/deepmodeling/deepmd-kit

extra:
  recipe-maintainers:
    - njzjz
```

最后一部分则是程序的一些基本信息，比如网站、授权协议、介绍之类，可以显示在 anaconda.org 上。

本文就介绍到这里。相信大部分人都看得云里雾里，不过没关系，本文的初衷就不是让人看懂。下一期将介绍 `libtensorflow` 的构建，之后也会绍，有了配方后，如何搭建自动构建的平台。

发布于 2020-08-19

文章被以下专栏收录