ABCIでできる簡単マルチGPU実行

AnyTech 木村

（1）マルチGPUのメリット／デメリット

メリット	– 処理の高速化 – 理想的には、処理にかかる時間が、「1／GPU数」になる
デメリット	– プログラムが複雑化してしまう – 下手をすれば、その実装に、恩恵以上の時間を要してしまう

（1）マルチGPUのメリット／デメリット

メリット	– 処理の高速化 – 理想的には、処理にかかる時間が、「1／GPU数」になる
デメリット	– プログラムが複雑化してしまう – 下手をすれば、その実装に、恩恵以上の時間を要してしまう

極力簡単に、マルチGPUによる高速化を、
ABCI上で実現する方法を、紹介させて頂きます

(2) ABCIのマルチGPUノード

サービス	計算資源タイプ	計算資源概要	バッチ及びインタラクティブジョブ	予約
計算資源	<計算ノード(V)>			
	rt_F	4GPU, 40コア, 360GBメモリ	1.0 ポイント/時間	36 ポイント/日 (1.5 ポイント/時間相当)
	rt_G.large	4GPU, 20コア, 240GBメモリ	0.9 ポイント/時間	NA
	rt_G.small	1GPU, 5コア, 60GB メモリ	0.3 ポイント/時間	
	rt_C.large	20コア, 120GBメモリ	0.6 ポイント/時間	
	rt_C.small	5コア, 30GBメモリ	0.2 ポイント/時間	
	<メモリインテンシブノード>			
	rt_M.large	8コア, 800GBメモリ	0.4 ポイント/時間	NA
	rt_M.small	4コア, 400GBメモリ	0.2 ポイント/時間	
	<計算ノード(A)>			
	rt_AF	8GPU, 72コア, 480GBメモリ	3.0 ポイント/時間	108 ポイント/日 (4.5 ポイント/時間相当)
	rt_AG.small	1GPU, 9コア, 60GB メモリ	0.5 ポイント/時間	NA
ストレージ	グループ領域 (共有ディスク)	5 ポイント/TB・月 (ABCIアカウントに割当てられる200GBのホーム領域は無償)		
	ABCIクラウドストレージ	0.0001 ポイント/GB・日 (従量制)		

引用:https://abci.ai/ja/how_to_use/tariffs.html

(2) ABCIのマルチGPUノード

サービス	計算資源タイプ	計算資源概要	バッチ及びインタラクティブジョブ	予約
計算資源	<計算ノード(V)>			
	rt_F	4GPU, 40コア, 360GBメモリ	1.0 ポイント/時間	36 ポイント/日 (1.5 ポイント/時間相当)
	rt_G.large	4GPU, 20コア, 240GBメモリ	0.9 ポイント/時間	
	rt_G.small	1GPU, 5コア, 60GBメモリ	0.3 ポイント/時間	NA
	rt_C.large	20コア, 120GBメモリ	0.6 ポイント/時間	
	rt_C.small	5コア, 30GBメモリ	0.2 ポイント/時間	
	<メモリインテンシブノード>			
	rt_M.large	8コア, 800GBメモリ	0.4 ポイント/時間	NA
	rt_M.small	4コア, 400GBメモリ	0.2 ポイント/時間	
	<計算ノード(A)>			
	rt_AF	8GPU, 72コア, 480GBメモリ	3.0 ポイント/時間	108 ポイント/日 (4.5 ポイント/時間相当)
	rt_AG.small	1GPU, 9コア, 60GBメモリ	0.5 ポイント/時間	NA
ストレージ	グループ領域 (共有ディスク)	5 ポイント/TB・月 (ABCIアカウントに割当てられる200GBのホーム領域は無償)		
	ABCIクラウドストレージ	0.0001 ポイント/GB・日 (従量制)		

- Tesla V100 × 4
- メモリ容量計：16GB x 4 = 64GB
- メモリ共有可能

- Tesla A100 × 8
- メモリ容量計：40GB x 8 = 320GB
- メモリ共有可能

引用：https://abci.ai/ja/how_to_use/tariffs.html

(3) notebookのアサインコマンド例

rt_F (V100x4)	qrsh -l rt_F=1 -g xxxXXXXX -l h_rt=12:00:00 module load gcc/11.2.0 module load python/3.8/3.8.13 module load cuda/11.3/11.3.1 source venv38/bin/activate jupyter notebook --ip=`hostname` --port=8888 --no-browser
rt_AF (A100x8)	qrsh -l rt_AF=1 -g xxxXXXXX -l h_rt=12:00:00 module load python/3.8/3.8.13 module load cuda/11.5/11.5.2 source venv38a/bin/activate jupyter notebook --ip=`hostname` --port=8888 --no-browser

参考：https://docs.abci.ai/ja/tips/jupyter-notebook/

(4) yolov5の学習でマルチGPU

rt_G.small (V100x1)	python train.py --img 640 --batch 16 --epochs 20 ¥ --data data/hoge.yaml --weights yolov5s.pt
rt_F (V100x4)	python -m torch.distributed.run --nproc_per_node **4** ¥ train.py --img 640 --batch **64** --epochs 20 ¥ --data data/hoge.yaml --weights yolov5s.pt ¥ **--device 0,1,2,3**
rt_AF (A100x8)	python -m torch.distributed.run --nproc_per_node **8** ¥ train.py --img 640 --batch **320** --epochs 20 ¥ --data data/hoge.yaml --weights yolov5s.pt ¥ **--device 0,1,2,3,4,5,6,7**

参考：https://github.com/ultralytics/yolov5/issues/475

(5) pytorchにて、カスタムモデルの学習でマルチGPU

rt_G.small
(V100x1)

```
model = hogenet()

model.load_state_dict(torch.load(PATH))
```

rt_AF
(A100x8)

```
model = hogenet()

model.load_state_dict(torch.load(PATH))

from sync_batchnorm import convert_model,
                                    DataParallelWithCallback

model = convert_model(model)
model = DataParallelWithCallback(model,
                                 device_ids=[0, 1, 2, 3, 4, 5, 6, 7])
```

参考：https://github.com/vacancy/Synchronized-BatchNorm-PyTorch

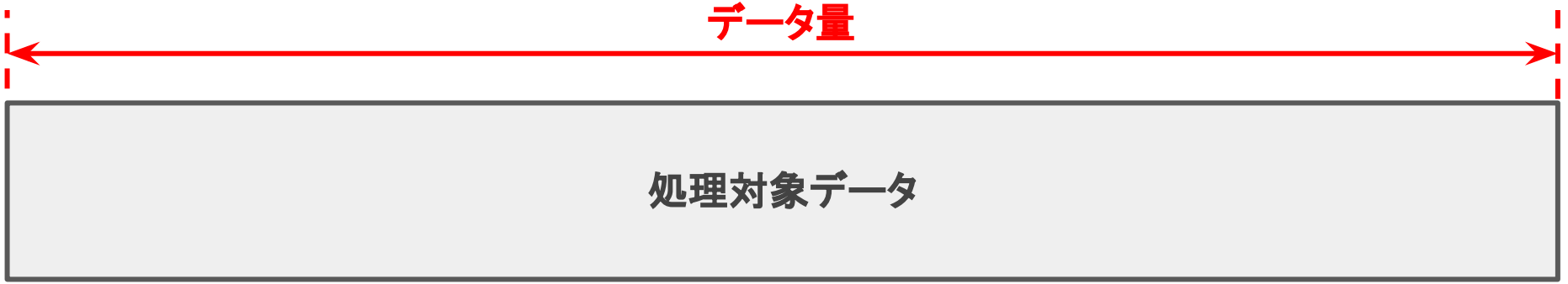
(5) pytorchにて、カスタムモデルの学習でマルチGPU

※save時だけ注意

通常の save	torch.save(model.state_dict(), PATH)
convert modelでの save	torch.save(model.**module**.state_dict(), PATH)

参考：https://qiita.com/conta_/items/c3e173e891145e87e668#解決策1-保存する前に気をつける

(6) 推論処理をマルチGPUで実践

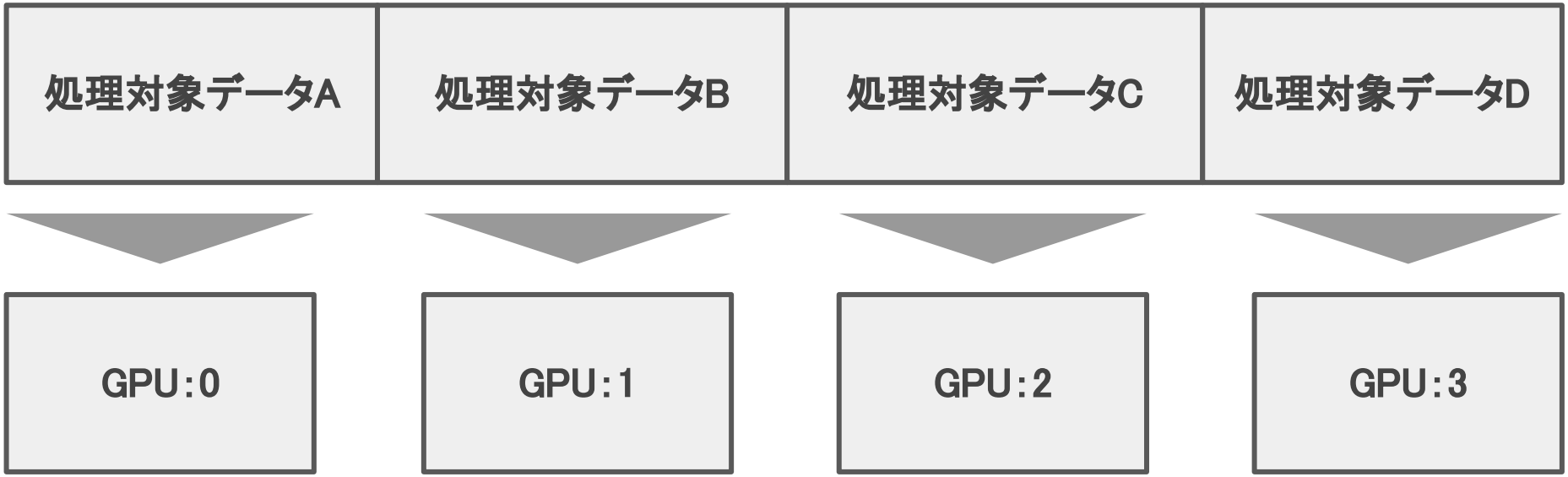


(6) 推論処理をマルチGPUで実践

@rt_G.small (V100x1)

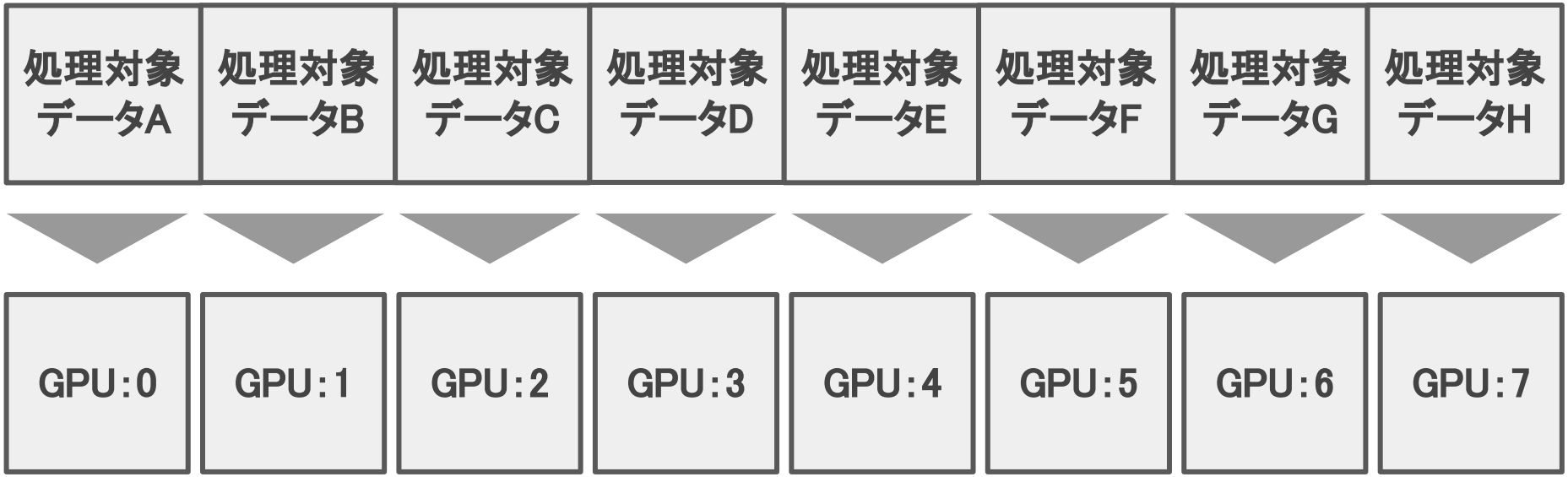
(6) 推論処理をマルチGPUで実践

@rt_F (V100x4)



(6) 推論処理をマルチGPUで実践

@rt_AF (A100x8)

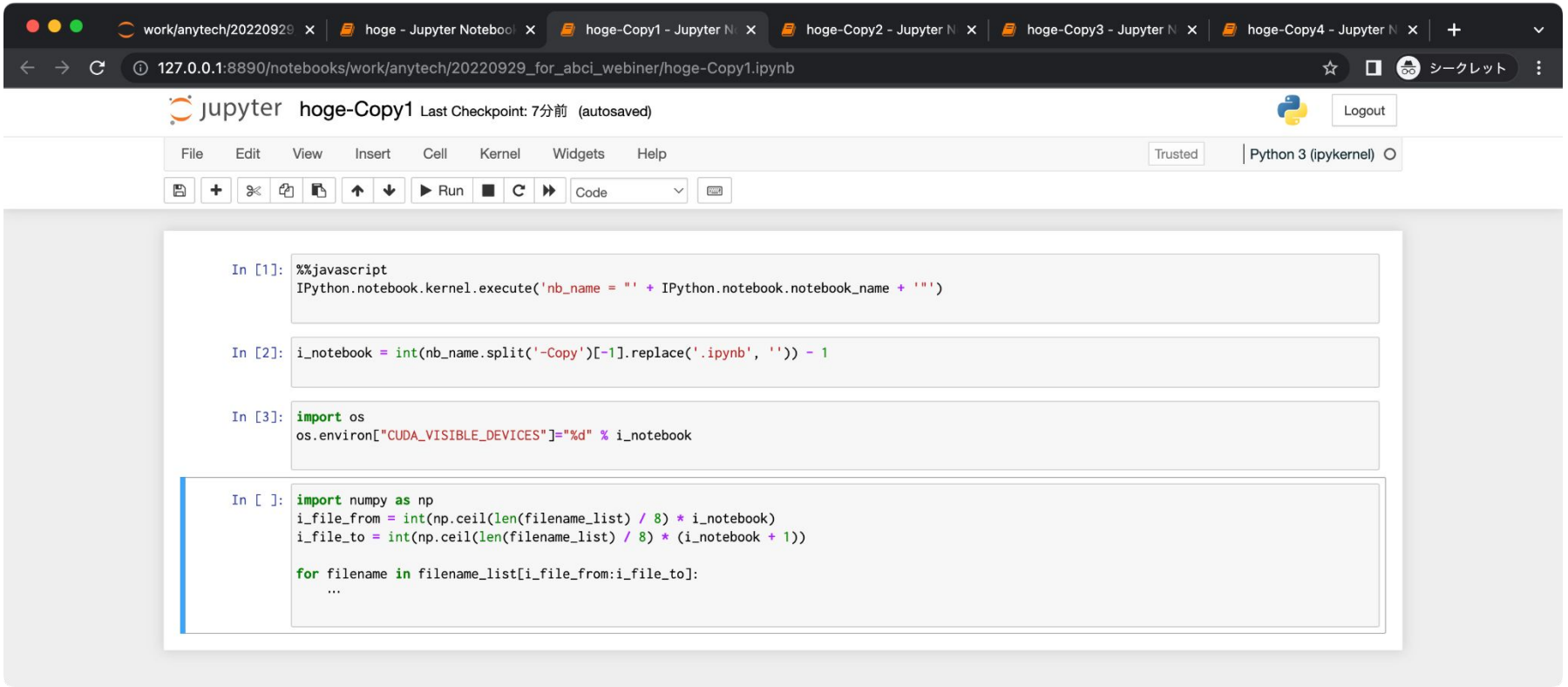


(6) 推論処理をマルチGPUで実践

notebookをコピーすると、「-CopyX.ipynb」という名称でインクリメントされる

(6) 推論処理をマルチGPUで実践

コピーしたnotebookを、ブラウザタブにて横展開して、一斉に実行



(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % i_notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    …
```

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

- 処理中のnotebook名称が、nb_nameに格納される
  （nb_name = hoge-Copy1.ipynb）
- waitがかかると失敗しやすい為、冒頭で処理すると良い

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', ''))
```

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % i_notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    ...
```

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

- 「hoge-Copy1.ipynb」から、
「1.ipynb」を切出す

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    ...
```

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

- 「1.ipynb」から「1」を切出す

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % i_notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    ...
```

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

- 文字列を整数型にキャストする

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % i_notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    ...
```

（6）推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

- pythonは、zeroオリジンの言語である為、
  「Copy1」が「0」となるように調整

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = str(i_notebook)
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 8) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    ...
```

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace('.ipynb', '')) - 1
```

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" % i_notebook
```

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 8) * i_notebook)
                                                  i_notebook + 1))
for filename in filename_list[i_file_from:i_file_to]:
    ...
```

- 「CopyX.ipynb」という名称に対応するGPUをアサインする

(6) 推論処理をマルチGPUで実践

```
%%javascript
IPython.notebook.kernel.execute('nb_name = "' + IPython.notebook.notebook_name + '"')
```

```
i_notebook = int(nb_name.split('-Copy')[-1].replace(
```

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="%d" %i_notebook
```

- データ分割は、ファイル名称で分けるなり、個別にアジャストすればOK
- 出力先が共通になってしまわないように、注意をする必要はあり

```
import numpy as np
i_file_from = int(np.ceil(len(filename_list) / 4) * i_notebook)
i_file_to = int(np.ceil(len(filename_list) / 4) * (i_notebook + 1))

for filename in filename_list[i_file_from:i_file_to]:
    …
```

(6) 推論処理をマルチGPUで実践

コピーしたnotebookを、ブラウザタブにて横展開して、一斉に実行