

Nvidia NGC kontejneri

Sadržaj

- [Uvod](#)
 - [Što su kontejneri?](#)
 - [Kako su kontejneri implementirani na Isabelli?](#)
- [Nvidia NGC \(Nvidia GPU Cloud\) kontejneri](#)
 - [Što su NGC kontejneri?](#)
 - [GPU podrška za vašu aplikaciju](#)
 - [Kako preuzeti kontejner?](#)
- [Primjeri korištenja NGC kontejnera putem Singularitya](#)
 - [NVIDIA HPC Benchmarks](#)
 - [Tensorflow 2](#)
 - [GROMACS](#)
- [Performanse vs. broj grafičkih procesora](#)

Uvod

Što su kontejneri?

Kontejneri su datoteke koje pružaju mogućnost stvaranja izoliranog korisničkog okruženja (sa svojim aplikacijama i njihovim ovisnostima) putem [virtualizacije na nivou operacijskog sustava](#). Virtualizacija kontejnerima na Linux poslužiteljima moguća je zbog njegove podjele (točnije: memoriskog prostora) na dvije glavne komponente: korisnički prostor i prostor jezgre.

Korisnički prostor je prostor više razine u kojem se nalaze aplikacije/knjizičice i upravlja jezgrom tzv. sistemskim pozivima. Prostor jezgre je prostor niže razine, rezerviran za naredbe koje upravljaju hardverom neovisno o platformi na kojoj se nalazi. Izmjena korisničkog prostora je stoga moguća je ako dva Linux operativna sistema dijele istu jezgru: funkcionalnost koju kontejneri rješavaju.

Na ovaj način, korisnik (zajednica ili projekt) može pripremiti kontejner kojim ostali mogu preskočiti cijelokupan proces instalacije/prilagođbe poslužitelju, i na taj način direktno prijesti na razvoj i/ili korištenje aplikacija. Osim što omogućuju efikasniju izvedbu aplikacija, kontejneri omogućuju pristup raznovrsnijem broju poslužitelja na način koji je konzistentan i prilagođen aplikaciji kojoj je namijenjen.

Kako su kontejneri implementirani na Isabelli?

Na Isabelli, kontejneri su implementirani putem [Singularitya](#): platforme koja je specifično prilagođena HPC okruženju. Više informacija o tehničkim osnovama kontejnera, njihovoj izgradnji i pogotovo načinu na koji ih podnosite na klasteru Isabella možete naći na [našem wikiju](#).

Nvidia NGC (Nvidia GPU Cloud) kontejneri

Što su NGC kontejneri?

[Nvidia NGC \(Nvidia GPU Cloud\) kontejneri](#) su namijenjeni korištenju Nvidia grafičkih procesora (poput V100 Tesli na [čvorovima Isabelle](#)) i sadrže optimizirane aplikacije, knjizičice ili programska sučelja koja se oslanjaju na ubrzanje GPU paralelizmom i [objavljuju na mjesecnoj bazi](#).

Tipovi kontejnera koji su dostupni:

- [Containers](#) - specifične aplikacije
- [Collections](#) - skup aplikacija ili sučelja koja su namijenjena određenom području primjene ili slučajevima
- [Models](#) - sadrže već istrenirane modele namijenjene dubokom učenju
- [Resources](#) - sadrže podatke, uputstva i rezultate za usporedbu modela dubokog učenja

GPU podrška za vašu aplikaciju

Postojanje podrške za grafičke procesore za vašu aplikaciju možete provjeriti na [sljedećoj poveznici](#) ili direktno na [NGC katalogu](#).

Kako preuzeti kontejner?

U većini slučajeva, kontejneri se mogu preuzeti naredbom:

```
$ singularity pull <ime_kontejnera>.sif docker://<url_kontejnera>
```

U određenim slučajevima potrebno je [otvoriti NGC račun \(izbor: NVIDIA Account\)](#) i pri stvaranju kontejnera definirati [NGC korisničke informacije](#):

```
$ export SINGULARITY_DOCKER_USERNAME='$oauthtoken'
$ export SINGULARITY_DOCKER_PASSWORD=<NGC API ključ>
$ singularity pull <ime_kontejnera>.sif docker://<url_kontejnera>
```

NGC API ključ može se stvoriti tek kada se otvor NGC korisnički račun kao što je [opisano u službenoj dokumentaciji](#).

Primjeri korištenja NGC kontejnera putem Singularitya

Primjeri korištenja NGC su prikazani na dvije aplikacije:

- [NVIDIA HPC Benchmarks - HPL](#) (tag: 21.4-hpl)
- [Tensorflow 2](#) (tag: 22.08-tf2-py3)
- [GROMACS](#) (tag: 2022.1)

Ispod se nalaze primjeri SGE skripti i ulaznih datoteka/skripti potrebnih za njihovo izvršavanje.

Korištenje grafičkih procesora

Pri korištenju grafičkih procesora, treba obratiti pažnju na koji se način se poslovi podnose kao što je [opisano u wikiju](#).

Posebnu pažnju treba dati uputama koje su navedene na stranicama NGC kataloga, koja obično sadrži detaljne upute o korištenju kontejnera.

NVIDIA HPC Benchmarks

hpl-singularity.sge

```
## -cwd
## -o output/
## -e output/
## -pe gpusingle 1

# module
module load mpi/openmpi41-x86_64

# run
export UCX_TLS="sm"
export CUDA_VISIBLE_DEVICES=$( cat $TMP/gpu )
mpirun -np $NSLOTS -bind-to none \
singularity run --nv hpc-benchmarks:21.4-hpl.sif \
hpl.sh \
--cpu-affinity $( python -c "print (','.join(str(i) for i in range(0,19,2)) + ':'*4" ) \
--gpu-affinity ${CUDA_VISIBLE_DEVICES//,/} \
--cpu-cores-per-rank 2 \
--dat $PWD/hpl-$NSLOTS.dat
```

hpl-1.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6           device out (6=stdout,7=stderr,file)
1           # of problems sizes (N)
40000       Ns
1           # of NBs
288         NBs
0           PMAP process mapping (0=Row-,1=Column-major)
1           # of process grids (P x Q)
1           Ps
1           Qs
16.0        threshold
1           # of panel fact
0 1 2       PFACTs (0=left, 1=Crout, 2=Right)
1           # of recursive stopping criterium
2 8         NBMINS (>= 1)
1           # of panels in recursion
2           NDIVs
1           # of recursive panel fact.
0 1 2       RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
3 2         BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lrg,5=LnM)
1           # of lookahead depth
1 0         DEPTHs (>=0)
1           SWAP (0=bin-exch,1=long,2=mix)
192         swapping threshold
1           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
0           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
```

hpl-2.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6           device out (6=stdout,7=stderr,file)
1           # of problems sizes (N)
55000       Ns
1           # of NBs
288         NBS
0           PMAP process mapping (0=Row-,1=Column-major)
1           # of process grids (P x Q)
2           Ps
1           Qs
16.0        threshold
1           # of panel fact
0 1 2       PFACTs (0=left, 1=Crout, 2=Right)
1           # of recursive stopping criterium
2 8         NBMINS (>= 1)
1           # of panels in recursion
2           NDIVs
1           # of recursive panel fact.
0 1 2       RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
3 2         BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lrg,5=LnM)
1           # of lookahead depth
1 0         DEPTHs (>=0)
1           SWAP (0=bin-exch,1=long,2=mix)
192         swapping threshold
1           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
0           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
```

hpl-4.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6           device out (6=stdout,7=stderr,file)
1           # of problems sizes (N)
70000       Ns
1           # of NBs
288         NBS
0           PMAP process mapping (0=Row-,1=Column-major)
1           # of process grids (P x Q)
4           Ps
1           Qs
16.0        threshold
1           # of panel fact
0 1 2       PFACTs (0=left, 1=Crout, 2=Right)
1           # of recursive stopping criterium
2 8         NBMINS (>= 1)
1           # of panels in recursion
2           NDIVS
1           # of recursive panel fact.
0 1 2       RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
3 2         BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lrg,5=LnM)
1           # of lookahead depth
1 0         DEPTHs (>=0)
1           SWAP (0=bin-exch,1=long,2=mix)
192         swapping threshold
1           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
0           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
```

Tensorflow 2

tensorflow2-singularity.sge

```
## -cwd
## -o output/
## -e output/
## -pe gpu 1

# environment
module load mpi/openmpi41-x86_64

# run
openmpi-wrapper.sh \
singularity run --nv tensorflow:22.08-tf2-py3 \
python3 $PWD/synthetic.py \
--batch-size 128 \
--num-warmup-batches 100 \
--num-batches-per-iter 30 \
--num-iters 100
```

synthetic.py

```
# Copyright 2019 Uber Technologies, Inc. All Rights Reserved.
#
# 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.
# =====
import argparse
import os
import numpy as np
import timeit
import sys
import time

import tensorflow as tf
import horovod.tensorflow as hvd
from tensorflow.keras import applications

# Benchmark settings
parser = argparse.ArgumentParser(description='TensorFlow Synthetic Benchmark',
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--fp16-allreduce', action='store_true', default=False,
                    help='use fp16 compression during allreduce')

parser.add_argument('--model', type=str, default='ResNet50',
                    help='model to benchmark')
parser.add_argument('--batch-size', type=int, default=32,
                    help='input batch size')

parser.add_argument('--num-warmup-batches', type=int, default=10,
                    help='number of warm-up batches that don\'t count towards benchmark')
parser.add_argument('--num-batches-per-iter', type=int, default=10,
                    help='number of batches per benchmark iteration')
parser.add_argument('--num-iters', type=int, default=10,
                    help='number of benchmark iterations')

parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')

args = parser.parse_args()
args.cuda = not args.no_cuda

# Horovod: initialize Horovod.
hvd.init()

# Horovod: pin GPU to be used to process local rank (one GPU per process)
if args.cuda:
    gpus = tf.config.experimental.list_physical_devices('GPU')
    for gpu in gpus:
        tf.config.experimental.set_memory_growth(gpu, True)
    if gpus:
        tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], 'GPU')
else:
    os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

# Set up standard model.
model = getattr(applications, args.model)(weights=None)
opt = tf.optimizers.SGD(0.01)

data = tf.random.uniform([args.batch_size, 224, 224, 3])
target = tf.random.uniform([args.batch_size, 1], minval=0, maxval=999, dtype=tf.int64)

@tf.function
def benchmark_step(first_batch):
    # Horovod: (optional) compression algorithm.
    compression = hvd.Compression.fp16 if args.fp16_allreduce else hvd.Compression.none

    # Horovod: use DistributedGradientTape
    with tf.GradientTape() as tape:

```

```

probs = model(data, training=True)
loss = tf.losses.sparse_categorical_crossentropy(target, probs)

# Horovod: add Horovod Distributed GradientTape.
tape = hvd.DistributedGradientTape(tape, compression=compression)

gradients = tape.gradient(loss, model.trainable_variables)
opt.apply_gradients(zip(gradients, model.trainable_variables))

# Horovod: broadcast initial variable states from rank 0 to all other processes.
# This is necessary to ensure consistent initialization of all workers when
# training is started with random weights or restored from a checkpoint.
#
# Note: broadcast should be done after the first gradient step to ensure optimizer
# initialization.
if first_batch:
    hvd.broadcast_variables(model.variables, root_rank=0)
    hvd.broadcast_variables(opt.variables(), root_rank=0)

def log(s, nl=True):
    if hvd.rank() != 0:
        return
    print(s, end='\n' if nl else '')

log('Model: %s' % args.model)
log('Batch size: %d' % args.batch_size)
device = 'GPU' if args.cuda else 'CPU'
log('Number of %ss: %d' % (device, hvd.size()))

with tf.device(device):

    # Warm-up
    log('Running warmup...')
    benchmark_step(first_batch=True)
    timeit.timeit(lambda: benchmark_step(first_batch=False),
                  number=args.num_warmup_batches)

    # Benchmark
    log('Running benchmark...')
    img_secs = []
    for x in range(args.num_iters):
        time = timeit.timeit(lambda: benchmark_step(first_batch=False),
                             number=args.num_batches_per_iter)
        img_sec = args.batch_size * args.num_batches_per_iter / time
        log('Iter #%d: %.1f img/sec per %s' % (x, img_sec, device))
        img_secs.append(img_sec)

    # Results
    img_sec_mean = np.mean(img_secs)
    img_sec_conf = 1.96 * np.std(img_secs)
    log('Img/sec per %s: %.1f +-%.1f' % (device, img_sec_mean, img_sec_conf))
    log('Total img/sec on %d %s(s): %.1f +-%.1f' %
        (hvd.size(), device, hvd.size() * img_sec_mean, hvd.size() * img_sec_conf))

```

GROMACS

gromacs-singularity.sge

```
## -cwd
## -o output/
## -e output/
## -pe gpusingle 1

# environment
module load mpi/openmpi41-x86_64

# download
DATA_SET=water_GMX50_bare
wget -c https://ftp.gromacs.org/pub/benchmarks/${DATA_SET}.tar.gz
tar xf ${DATA_SET}.tar.gz
cd ./water-cut1.0_GMX50_bare/1536

# run
GROMACS_SIF=$( readlink -f ../../gromacs:2022.1.sif )
SINGULARITY="singularity run --nv -B ${PWD}:/host_pwd --pwd /host_pwd ${GROMACS_SIF}"

${SINGULARITY} gmx grompp -f pme.mdp

export GMX_GPU_DD_COMM=true
export GMX_GPU_PME_PP_COMM=true
export GMX_FORCE_UPDATE_DEFAULT_GPU=true
cuda-wrapper.sh ${SINGULARITY} gmx mdrun \
    -ntmpi $NSLOTS -nb gpu -pme gpu -ntomp 2 \
    -pin on -v -noconfout -nsteps 3000 -s
```

Performanse vs. broj grafičkih procesora

Ispod su prikazane performanse prethodno navedenih primjera u ovisnosti o broju grafičkih procesora.

Aplikacija [jedinica]	GPU	Paralelna okolina	Performansa	Performansa/GPU
NVIDIA HPC Benchmarks - HPL [Gflops]	1	gpusingle	4184	4184
	2	gpusingle	7801	3900
	4	gpusingle	9532	3117
Tensorflow 2 [img/sec]	1	gpu	396	396
	2	gpu	768	384
	3	gpu	1160	387
GROMACS [ns/day]	1	gpusingle	9.4	9.4
	2	gpusingle	8.1	4.0
	3	gpusingle	12.2	4.0