Getting Started with NVIDIA Parabricks

Table of contents

Installation Requirements

Hardware Requirements

- Any NVIDIA GPU that supports CUDA architecture 70, 75, 80, 86, 89 or 90 and has at least 16GB of GPU RAM. NVIDIA Parabricks has been tested on the following NVIDIA GPUs:
    - V100
    - T4
    - A10, A30, A40, A100, A6000
    - L4, L40
    - H100, H200
    - Grace Hopper Superchip
- The fq2bam tool requires at least 24 GB of GPU memory by default; the `--low-memory` option will reduce this to 16 GB of GPU memory at the cost of slower processing. All other tools require at least 16 GB of GPU memory per GPU.
- System Requirements:
    - A 2 GPU system should have at least 100GB CPU RAM and at least 24 CPU threads.
    - A 4 GPU system should have at least 196GB CPU RAM and at least 32 CPU threads.
    - A 8 GPU system should have at least 392GB CPU RAM and at least 48 CPU threads.

Note

Parabricks is supported on time-sliced virtual (vGPU) but not on Multi-Instance (MIG) GPUs.

Software Requirements

The following are software requirements for running Parabricks.

- An NVIDIA driver with version 525.60.13 or greater .
- Any Linux Operating System that supports nvidia-docker2 Docker version 20.10 (or higher)

Please see this page for more information on supported driver configurations.

Note

Parabricks is available as Docker image. For Singularity users, please check here about importing a Docker image into a Singularity Image,

Verifying Hardware and Software Requirements

Checking available NVIDIA hardware and driver

To check your NVIDIA hardware and driver version, use the `nvidia-smi` command:

```
$ nvidia-smi +-----------------------------------------------------------------------------+ | NVIDIA-SMI 525.60.13 Driver Version: 525.60.13 CUDA Version: 12.0 | |-------------------------------+----------------------+----------------------+ | 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 Tesla V100-DGXS... On | 00000000:07:00.0 Off | 0 | | N/A 44C P0 38W / 300W |
74MiB / 16155MiB | 0% Default | | | | N/A | +---------------------------------------------------
----------------------+ | Processes: | | GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|=============================================================================
| 0 N/A N/A 3019 G /usr/lib/xorg/Xorg 56MiB | +--------------------------------------------------
-------------------------+
```

This shows the following important information:

- The NVIDIA driver version is 525.60.13.
- The supported CUDA driver API is 12.0.
- The GPU has 16 GB of memory.

Checking available CPU RAM and threads

To see how much RAM and CPU threads in your machine, you can run the following:

```
# To check available memory $ cat /proc/meminfo | grep MemTotal # To check
available number of threads $ cat /proc/cpuinfo | grep processor | wc -l
```

Checking nvidia-docker2 installation

To make sure you have nvidia-docker2 installed, run this command:

```
$ docker run --rm --gpus all nvidia/cuda:12.0.0-base-ubuntu20.04 nvidia-smi
```

When it finishes downloading the container, it will run the `nvidia-smi` command and show you the same output as above.

Checking python version

To see which version of Python you have, enter the following command:

```
$ python3 --version
```

Make sure it's at least version 3 (3.6.9, 3.7, etc).

Getting the Software

The NVIDIA Parabricks Docker image can be obtained by running the following command:

```
$ docker pull nvcr.io/nvidia/clara/clara-parabricks:4.3.1-1
```

At this point the software is ready to use.

Running NVIDIA Parabricks

From the Command Line

Parabricks is deployed using a Docker image. There are two parts to customizing a Parabricks run:

- Customizing Docker container specific options: These are the options that are passed to the `docker` command before the name of the container. For example, the user should mount their data directories within the Docker container by passing the `-v` option to Docker. See the Tutorials for more detailed examples.
- Parabricks specific options: These options are passed to the Parabricks command line to customize the Parabricks run. For example, you can choose which tool to run and pass tool-specific options.

For example, use the following command to run the Parabricks fq2bam (BWA-MEM + GATK) tool using a Docker container. See the tutorial for further details on how this command works.

```
$ docker run \ --gpus all \ --rm \ --volume $(pwd):/workdir \ --volume
$(pwd):/outputdir \ nvcr.io/nvidia/clara/clara-parabricks:4.3.1-1 \ pbrun fq2bam \ --
ref /workdir/parabricks_sample/Ref/Homo_sapiens_assembly38.fasta \ --in-fq
/workdir/parabricks_sample/Data/sample_1.fq.gz
/workdir/parabricks_sample/Data/sample_2.fq.gz \ --out-bam
/outputdir/fq2bam_output.bam
```

Sample data is freely available. See the Getting The Sample Data section in the Tutorials for instructions on obtaining the sample data, and a step-by-step guide to using both fq2bam and Haplotype Caller.

Some useful Docker options to consider:

- `--gpus all` lets the Docker container use all the GPUs on the system. The GPUs available to Parabricks container can be limited using the `--gpus "device=&lt;list of GPUs&gt;"` option. Use `nvidia-smi` to see how many GPUs you have, and which one is which.
- `--rm` tells Docker to terminate the image once the command has finished.
- `--volume $(pwd):/image/data` mounts your current directory (a path on the server) on the Docker container in the `/image/data` directory (a path inside the Docker container). If your data is not in the current directory use an option similar to `--volume /path/to/your/data:/image/data`.
- `--workdir` tells Docker what working directory to execute the commands from (inside the container).
- The rest of the command is the Parabricks tool you want to run, followed by its arguments. For those familiar with pre-v4.0 versions of Parabricks and its `pbrun` command, this Docker invocation takes the place of `pbrun`.

Running Parabricks Using the Base Command Platform

An example command to launch a BaseCommand container on a single-GPU instance is:

```
ngc batch run --name "parabricks-germline" \ --instance dgxa100.80g.1.norm \ --
commandline "pbrun germline \ --ref
/workspace/parabricks_sample/Ref/Homo_sapiens_assembly38.fasta \ --in-fq
/Data/HG002-NA24385-pFDA_S2_L002_R1_001-30x.fastq.gz /Data/HG002-NA24385-
pFDA_S2_L002_R2_001-30x.fastq.gz \ --knownSites
/workspace/parabricks_sample/Ref/Homo_sapiens_assembly38.known_indels.vcf.gz
\ --out-bam output.bam \ --out-variants output.vcf \ --out-recal-file report.txt \ --run-
partition \ --no-alt-contigs" \ --result /results \ --image "nvcr.io/nvidia/clara/clara-
parabricks:4.3.1-1"
```

Note that for other Parabricks commands (i.e. fq2bam, HaplotypeCaller, DeepVariant) the `ngc batch run` command is similar. Make sure to use the correct paths for your workplace or dataset that contains the data you intend to use.

Uninstalling the software

Uninstalling NVIDIA Parabricks is as simple as removing the Docker image.

```
$ docker images REPOSITORY TAG IMAGE ID CREATED SIZE ...
nvcr.io/nvidia/clara/clara-parabricks 4.3.1-1 516740210042 2 months ago 3.23GB ...
$ docker rmi 516740210042
```

The exact value of the "IMAGE ID" will vary depending on your installation.

Note

- User guides and Reference manuals can be found on the NVIDIA Parabricks documentation page.

- Answers to many other FAQs can be found on the developer forum.