GRAPHCORE

PopDist and PopRun: User Guide

Version latest

Graphcore Ltd

Aug 25, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

PopRun is a command line utility to launch distributed applications on Graphcore IPU-POD compute systems. Specifically, PopRun is used to create multiple instances. Each instance can either be launched on a single host server or multiple host servers within the same IPU-POD, depending on the number of host servers available on the target IPU-POD. Typically, an IPU-POD$_{64}$ is configured with one, two or four host servers.

On large IPU-POD systems such as an IPU-POD$_{128}$, PopRun will automatically launch multiple instances on *remote* host servers. With remote host servers we mean servers that are physically located in another interconnected IPU-POD. This makes PopRun a powerful tool for running applications at scale.

For PopRun to launch applications, the actual application must be written in a way so that it is able to take advantage of the additional compute power provided by many IPUs inside an IPU-POD. Other motivating factors to use PopRun are:

1. PopRun lets you launch multiple instances of your application on one or more IPU-PODs.
2. Depending on your application, launching multiple instances may increase the performance of your application.
3. PopRun lets you perform careful placement of multiple instances on the host server to minimise NUMA effects.
4. PopRun is required if you wish to scale your application beyond a single IPU-POD$_{64}$.

The Poplar Distributed Configuration Library (PopDist) provides a set of APIs that you can use to make your application ready for distributed execution. Command line parameters passed to PopRun are exposed to the developer, which you can used to distribute the input/output data or other parts of the applications.

1.1 Installation

Both PopRun and PopDist come bundled with the Poplar SDK. No additional installation is required. The PopRun binary is called `poprun` and can be found in the `bin` directory of the Poplar SDK installation.

Before you proceed, make sure you have sourced the `enable.sh` script as described in the Getting Started Guide for your IPU system.

1.2 Validating the installation

To validate your Poplar SDK installation and thus make sure that `poprun` is available, run the following command:

```
$ poprun --num-instances 2 --num-replicas 2 --offline-mode=1 echo "Hello world!"
```

If PopRun is setup properly, you should see **Hello world!** printed out twice, once per instance:

```
Hello world!
Hello world!
```

As mentioned previously, PopDist requires an external Python package called Horovod. Please consult Section 4, Poplar distributed configuration library (PopDist) of for more information.

The following sections will provide more detailed information regarding the various PopRun and PopDist features.

CHAPTER

TWO

LAUNCHING APPLICATIONS WITH POPRUN

In order to understand how PopRun works, it is to important consider the following:

- The application to be distributed using the PopDist library
- Input data required by the application
- Output data generated by the application
- Virtual-IPU (V-IPU) utility for allocating IPUs on IPU-PODs

This guide assumes that you are familiar with the concepts mentioned above. More information on V-IPU can be found in the V-IPU User Guide.

The typical workflow of launching a distributed application on an IPU-POD involves the following steps:

- **Login**: All IPU-PODs are equipped with at least one host server. All application launches are made from a host server. Thus, the starting point for any application launch is logging on to a host server.
- **Distribution**: The actual application to be launched must be implemented in such a way so that it is capable of taking advantage of multiple IPUs, either within a single IPU-M2000 or across multiple IPU-M2000s. The PopDist API is used to make applications distributed. Typically, this involves dividing the computation in such a way so that it can be executed in parallel. The IPUs involved in the computation can either be located within an IPU-POD or in another interconnected IPU-POD.
- **Launch**: The actual application launch is made by calling PopRun from the command line on a host server.
- **Allocation**: Depending on the resources available to you, including both IPUs and host servers, PopRun will *automatically* interface with V-IPU to allocate and reserve the resources you need to execute your application.
- **Execution**: the final step involves the actual execution, where the application runs on the IPUs.

Note: Some IPU applications may involve the host as part of the computation or for host-side collective operations.

2.1 Launch modes

Applications distributed with PopDist can be launched in several ways. Below are the most common PopRun launch modes:

- **Single instance**: The simplest launch mode. Here, a single instance is launched on the host server to run an application on a single or multiple IPU-M2000s located in one IPU-POD. Each instance runs on a single graph compile domain (GCD), which is a subset of the available IPUs connected by IPU-Links (an IPU-Link domain)
- **Multi-instance/Single host**: In this launch mode, multiple instances are launched on the *same* host server. Typically, the application targets multiple IPU-M2000s. This mode is recommended for applications where the host CPU is used for pre-processing or other I/O tasks. Each instance can run on a separate GCD.

- **Multi-instance/Multiple hosts**: This mode applies to IPU-PODs that are equipped with multiple host servers. In this mode, multiple instances are launched on the multiple host servers located within a single IPU-POD. This mode is recommended, for example, for applications with special pre- or post-processing needs that must take place on the host CPU.
- **Multi-instance/Multiple IPU-PODs**: The most extensive mode where multiple instances are launched on one or more host servers across multiple IPU-PODs connected via GW-Links (known as a graph scaleout domain). This mode is recommended for highly scalable IPU applications. Each IPU-POD contains one or more IPU-Link domains (sets of IPUs connected via IPU-Links) The number of IPU-Link domains is specified with the `--num-ilds` option.

2.2 Multi-host setup

Launching applications on multiple hosts requires that you have an SSH key pair to authenticate the connections between the hosts. We recommend using the `ssh-keygen` tool to generate a new key pair for this purpose or copy a key pair from another machine.

Note: The steps shown in this section are only required if your user home directory is not located on the same host.

To authorize this key pair on all the hosts, use the `ssh-copy-id` command. In this example, we assume that we have the four hosts in the following IP address range: `10.1.3.10[1-4]`. To copy the SSH key to all of the mentioned host, issue the following command:

```
$ ssh-copy-id 10.1.3.101
$ ssh-copy-id 10.1.3.102
$ ssh-copy-id 10.1.3.103
$ ssh-copy-id 10.1.3.104
```

In order to verify that you have successfully copied your SSH key to all hosts, you can try to `ssh` into each of them. If you get access without being prompted for a password, you are ready to start using PopRun with that host.

Should you encounter problems accessing a host when using PopRun, an error will be reported. There are two typical errors that you could encounter:

- **Host key verification failed**: This indicates that the key of the remote host was not accepted by the local host. The key of the host typically needs to be placed in `~/.ssh/known_hosts` to be automatically verified. Note that when using `ssh` in interactive mode, you will be asked if you want to add the remote key the first time you connect. However, when using PopRun, this is not the case as PopRun uses `ssh` in non-interactive mode. So the easiest way to get the remote host key added is typically to use `ssh` to log into it the first time.
- **Permission denied**: This indicates that the remote host did not grant access to the local host. The key of the local host typically needs to be placed in `~/.ssh/authorized_keys` on the remote host in order to be automatically accepted. This can be done by using the `ssh-copy-id` command as explained above. Note that interactive password authentication is not supported by PopRun.

2.3 Application launches

In this section we will explain how sample IPU applications can be launched using PopRun using the various launch modes shown in section on Section 2.1, Launch modes. For simplicity, we will assume that our application is a Python application called `train.py`. The program arguments are also not shown, as they are irrelevant. Moreover, the IP address to the `--vipu-server-host` is fictional and used for the sake of the example. Replace it with the IP address of your own V-IPU server.

2.3.1 Single instance

```
$ poprun --numa-aware=yes --vipu-server-host=10.3.7.150 --vipu-partition=P8 \
        --vipu-cluster=A8 --mpi-global-args="--tag-output" \
        --num-replicas=4 --num-instances=1 python3 train.py
```

2.3.2 Multi instance / Single host

```
$ poprun --numa-aware=yes --vipu-server-host=10.3.7.150 --vipu-partition=A8 \
        --vipu-cluster=A8 --mpi-global-args="--tag-output" \
        --num-replicas=8 --num-instances=2 python3 train.py
```

2.3.3 Multi instance / Multi host

```
$ poprun --host 10.3.7.153,10.3.7.154 --numa-aware=yes \
     --vipu-server-host=10.3.7.150 --vipu-partition=P8 \
     --vipu-cluster=A8 --num-ilds=1 --mpi-global-args="--tag-output" \
     --mpi-local-args="-x PYTHONPATH" \
     --num-replicas=8 --num-instances=2 python3 train.py
```

The key takeaway from the command line shown above is that IP addresses of the hosts involved are passed to PopRun, and also any environment variables that you want to export from the local host to the remote hosts, in this example the `PYTHONPATH`.

2.3.4 Multi instance / Multi host / Multi IPU-PODs

```
$ poprun --host 10.3.7.153,10.3.8.153 --numa-aware=yes \
  --vipu-server-host=10.3.7.150 --vipu-partition=P8 --vipu-cluster=P8 \
  --num-ilds=2 --mpi-global-args="--tag-output" \
  --mpi-local-args="-x PYTHONPATH" \
  --num-replicas=8 --num-instances=2 python3 train.py
```

The command line shown above is similar to the one shown in Section 2.3.3, Multi instance / Multi host. One difference is that the number of IPU-Link domains (`--num-ilds`), is set to two, as opposed to one.

CHAPTER

THREE

POPRUN FEATURES

The PopRun launch format is as follows:

```
$ poprun [options] program [program args]
```

To list all the available command line options in PopRun, type:

```
$ poprun --help
```

This will print out the following table of options:

Option	Description	Default Value
`--help`	Produce help message	
`-H [ --host ] arg`	A comma-separated list of hosts on which to invoke instances	
`--num-replicas arg`	The total number of replicas	1
`--ipus-per-replica arg`	The number of IPUs per replica	1
`--num-instances arg`	The number of instances	No default
`--num-ilds arg`	The number IPU-Link domains	1
`--vipu-server-host arg`	The host of the V-IPU server	localhost
`--vipu-server-port arg`	The port number of the V-IPU server	8090
`--vipu-server-timeout arg`	The timeout for the V-IPU server requests (in seconds)	10
`--vipu-partition arg`	Create/use the specified V-IPU partition	No default
`--update-partition arg`	Update the V-IPU partition if needed	False
`--reset-partition arg`	Reset the V-IPU partition before execution. If not set, non-reconfigurable partitions are reset by default, while reconfigurable partitions are not	
`--remove-partition arg`	Remove a V-IPU partition after execution if the `--vipu-partition` options was used to create a partition	1
`--vipu-cluster arg`	Use the specified V-IPU cluster when creating a partition	
`--executable-cache-path arg`	Use the specified directory as an executable cache path	

continues on next page

Table 3.1 – continued from previous page

Option	Description	Default Value
`--mpi-global-args arg`	Global options to be passed to mpirun	
`--mpi-local-args arg`	Local (per-instance) options to be passed to mpirun	
`--numa-aware arg`	Bind instances to NUMA nodes in a round-robin fashion	0
`--print-topology arg`	Print topology table	0
`--only-output-from-instance arg`	Suppress the output from all other instances	
`--offline-mode arg`	Run without requiring or attaching to any IPUs	0
`-v [ --verbose ]`	Verbose output	0
`--vv`	Very verbose output	

Warning: You should be very careful when using the `--only-output-from-instance` option, as this will hide potential errors from the other instances.

3.1 Tips and tricks

In this section we share tips and tricks for more advanced users.

3.1.1 Passing environment variables

Often it can be useful to pass specific environment variables to each of the instances. To pass your own custom environment variables to the various instances, simply add the following option to PopRun:

```
$ poprun --mpi-local-args="-x POPLAR_ENGINE_OPTIONS=VALUE"
```

3.1.2 Character escaping

If you are passing custom environmental variables and its value contains special characters such as the quote (“) character, it might be cumbersome to escape such character. An alternative is to use a technique called environment variable forwarding. It can be applied like this:

```
$ export POPLAR_ENGINE_OPTIONS=".."
$ poprun --mpi-local-args="-x POPLAR_ENGINE_OPTIONS"
```

In the case shown above, the external value of `POPLAR_ENGINE_OPTIONS` is forwarded to *all* instances.

3.1.3 Tagging

It can sometimes be useful to trace what is printed out on the screen back to its originating instance. This is particularly true when debugging parallel and distributed applications. Luckily, PopRun provides a feature called **tagging**, which makes it possible to map what is printed out to an associated instance. This is done by prefixing the instance identifier to the text that is output to screen. Here is a simple example:

```
$ [1,1]<stdout> iteration: 6396, epoch: 81.79, lr: 0.1764, loss: 2.131
$ [1,0]<stdout> iteration: 6396, epoch: 81.79, lr: 0.1764, loss: 2.131
```

In the code excerpt shown above, the output is piped to `<stdout>` is prefixed with a pair of brackets ([]). The instance identifiers appear inside the brackets.

Tagging can be enabled by simply passing the following option to PopRun:

```
$ poprun --mpi-global-args="--tag-output"
```

3.1.4 File output forwarding

Another useful feature, especially in connection with debugging, is the ability to capture `<stdout>` and `<stderr>` and forward it to separate files. Preferably, where one file corresponds to each instance. With PopRun's file output forwarding this is possible. Simply pass the additional option to PopRun to enable this feature:

```
$ poprun --mpi-global-args="--output-filename output"
```

CHAPTER

FOUR

POPLAR DISTRIBUTED CONFIGURATION LIBRARY (POPDIST)

This section provides information on how you can use the PopDist library to make the appropriate changes to your application so that it can be launched in a distributed environment using PopRun. In short, PopDist provides a set of APIs which you can use to write a distributed application. As the examples later in this guide will demonstrate, very few modifications are needed in order to prepare an application for distributed execution.

As already mentioned in the introduction section, PopRun uses `mpirun` for multiple instance creation, while PopDist leverages `mpi` for communication between multiple hosts. This is not the be confused with communication between IPUs, which is realised using the Graphcore Communication Library (GCL) over either the IPU-Links or the GW-Links.

There are several packages that enable `mpi` communication in Python applications, but we recommend Horovod due to its simplicity. Broader details on the Horovod package can be found in the documentation here.

4.1 Installation

If your machine learning framework of choice is PopART, Graphcore's implementation of Horovod is recommended. The Graphcore TensorFlow wheel comes bundled with Horovod and thus no additional installation is needed. The Poplar SDK ships with a wheel for Graphcore Horovod and it can be installed using `pip`.

Please install and enable the Poplar SDK following the instructions in the Getting Started guide for your IPU system. Do this before installing Horovod to ensure that it uses the OpenMPI version that comes with the SDK.

Then Horovod can be installed with your virtual Python environment of choice activated:

```
$ pip install <sdk_path>/horovod.x.y.z.whl
```

PyTorch users on the other hand, must install the official Horovod package if they plan to use `mpi` for host side communication. This can be done using `pip` or simply typing:

```
$ pip install horovod
```

PyTorch is compatible with the latest version of Horovod. Be sure to have your virtual Python environment of choice activated first.

4.1.1 Validating Horovod

If you use TensorFlow, you can run the following code snippet in order to ensure that Horovod is installed correctly and working.

Listing 4.1: TensorFlow Horovod

```
1 # Copyright (c) 2021 Graphcore Ltd. All rights reserved.
2
3 import popdist
4 import popdist.tensorflow
5 from tensorflow.python.ipu import horovod as hvd
```

(continues on next page)

(continued from previous page)

```

hvd.init()
```

If you are using PyTorch, you can run the following code snippet to verify that Horovod is installed correctly.

Listing 4.2: PyTorch Horovod

```
# Copyright (c) 2021 Graphcore Ltd. All rights reserved.

import torch
import poptorch
import popdist
import horovod.torch as hvd

hvd.init()
```

4.2 The PopDist API

4.2.1 Core

popdist.popdist_core.**checkNumIpusPerReplica**(*expected*)
Check if the IPUs per replica in the context matches the expected number of IPUs.

False is returned if the environment variable is set and does not match the given value. On the other hand, if the environment variable is not set, a check is performed to see if the number of IPUs per replica corresponds to the expected number of IPUs.

Returns True if number of IPUs per replica equal the expected number of IPUs. False otherwise.

Parameters **expected** (int) –

Return type bool

popdist.popdist_core.**getDeviceId**(*ipus_per_replica=None*)
Gets the device id of device suitable for PopDist.

Returns Returns the device id of a suitable device.

Parameters **ipus_per_replica** (Optional[int]) –

Return type int

popdist.popdist_core.**getInstanceIndex**()
Gets the index of the current instance.

Can only be used with a uniform number of replicas per instance.

Returns instance index in `[0, getNumInstances())`.

Return type int

popdist.popdist_core.**getNumInstances**()
Gets the total number of instances.

Can only be used with a uniform number of replicas per instance.

Returns the total number of instances.

Return type int

popdist.popdist_core.**getNumIpusPerReplica**()
Get the number of IPUs per replica.

A function that will try to infer the number of IPUs per replica from the environment variables. Will default to 1 if the environment variable cannot be determined.

Returns 1 if the environment variable cannot be inferred.

Return type int

popdist.popdist_core.**getNumLocalReplicas**()
Gets the number of local replicas.

Infers the number of local replicas automatically from the environment variables.

Returns 1 if the environment variable is not set.

Return type int

popdist.popdist_core.**getNumTotalReplicas**()
Get the total number of replicas.

Infer the total number of replicas from environment variables. Will default to 1 if the environment variable cannot be determined.

Returns 1 if the environment variable cannot be inferred.

Return type int

popdist.popdist_core.**getReplicaIndexOffset**()
Gets the replica index offset.

The replica index corresponds to the offset of the first replica in an instance.

Returns 0 if no environment variable is set.

Return type int

popdist.popdist_core.**isPopdistEnvSet**()
Check if the PopDist environment is set.

Returns True if set.

Return type bool

popdist.popdist_core.**isUniformReplicasPerInstance**()
Checks if the number of replicas per instance is uniform.

Automatically inferred from the environment variables passed to PopDist.

Returns True if the number of replicas per instance is the same for all instances or if no environment is set.

Return type bool

4.2.2 PopART

popdist.popart.**configureSessionOptions**(*opts*)
Configure PopART session options to work with the PopDist context.

Parameters **opts** (popart.SessionOptions) –

Return type None

popdist.popart.**getDevice**(*ipusPerReplica*)
Get a PopART device that works with the PopDist context.

Returns An attached device.

Parameters **ipusPerReplica** (int) –

Return type popart.DeviceInfo

4.2.3 PopTorch

class popdist.poptorch.**Options**(**args*, ***kwargs*)
An extension to PopTorch's `Options` class so that it is easier to pass application-specific options to PopDist.

4.2.4 TensorFlow

popdist.tensorflow.**set_ipu_config**(*config*, *ipus_per_replica*, *configure_device=True*)
Set the PopDist configuration options for TensorFlow.

Parameters

- **config** – An IPUConfig instance created with `tensorflow.python.ipu.config.IPUConfig()` - or IpuOptions configuration protobuf created with `tensorflow.python.ipu.utils.create_ipu_config()` (deprecated) - to update.
- **ipus_per_replica** (`int`) – The number of IPUs per replica.
- **configure_device** (`bool`) – Whether to update `config` to select the IPU device for PopDist execution.

Returns The passed `config`.

4.3 PopDist examples

In this section we will detail how you can use PopDist to distribute your application. Examples for PyTorch and TensorFlow are shown.

4.3.1 PyTorch

The code example below outlines the most common steps involved in adding PopDist to an application. For the sake of brevity, the example code shown below assumes that the application is launched using PopRun like this:

```
$ poprun --vipu-partition=MyPartition --vipu-server-host=127.0.0.1
         --num-replicas=4 \
         --num-instances=1 --numa-aware=yes -v python main.py
```

Listing 4.3: Simple PopDist Example with PyTorch

```
# Copyright (c) 2021 Graphcore Ltd. All rights reserved.

import torch
import poptorch
import popdist
import popdist.poptorch
import horovod.torch as hvd


def init_popdist(args):
    hvd.init()
    args.use_popdist = True
    if popdist.getNumTotalReplicas() != args.replicas:
        print(f"The number of replicas is overridden by PopRun. "
              f"The new value is {popdist.getNumTotalReplicas()}.")
    args.replicas = int(popdist.getNumLocalReplicas())

    args.popdist_rank = popdist.getInstanceIndex()
    args.popdist_size = popdist.getNumInstances()


def create_model(opts):
```

(continues on next page)

(continued from previous page)

```
23      if opts.use_popdist:
24          model_opts = popdist.poptorch.Options()
25      else:
26          model_opts = poptorch.Options()
27
28      return model_opts
29
30
31  if __name__ == '__main__':
32
33      opts = command_line_arguments()
34
35      # Initialise PopDist
36      if popdist.isPopdistEnvSet():
37          init_popdist(opts)
38      else:
39          opts.use_popdist = False
40
41      create_model(opts)
```

The PopTorch PopDist package is imported on line 5-6, while Horovod is imported on line 7. Next, the PopRun command line parameters are parsed and passed to `init_popdist`, which is used to initialise both Horovod and PopDist.

Note: The use of Horovod is optional. It is **not** a requirement to initialise PopDist.

Some of the applications in our GitHub examples repository, such as our PyTorch CNNs training application make use of PopDist and PopRun.

4.3.2 TensorFlow

See the TensorFlow PopDist feature example.

Our TensorFlow CNNs training application also makes use of PopDist and PopRun.

4.4 Conclusion

A PopDist application can be organised into the following parts:

1. **Parsing phase**: where PopRun runtime command line parameters are parsed and handled.

2. **Initialisation phase**: where much needed variables such as *rank* and *size* are stored by making PopDist API calls. If MPI is to be used between the various instances, Horovod is also initialised.

3. **Digestion phase**: where PopDist variables are actively used to make the application distributed.

The steps above are just rough outlines and may vary from application to application.

CHAPTER

FIVE

POPRUN CHANGELOG

5.1 v2.2 (Poplar SDK 2.2)

5.1.1 New features

- Added checks for IPU/GW-link routing and sync type of existing partititons. The existing partition is checked against the values passed to `--ipu-link-routing-type`, `--gw-link-routing-type` and `--sync-type`. In case of a mismatch, the partition will be updated if `--update-partition=yes` is provided.
- Improved error message when the application was terminated by SIGKILL.

5.2 v2.1 (Poplar SDK 2.1)

5.2.1 New features

- Show full hostnames after the topology table if they cannot fit inside the table.
- Added command-line arguments for additional V-IPU options: `--ipu-link-routing-type`, `--gw-link-routing-type` and `--sync-type`.
- Improved error reporting when user program is missing from the command-line invocation.
- Added support for passing an environment variable to a specific instance by using `--instance-mpi-local-args=<instance-index>:-x VAR=VALUE`.
- Added initial support for the Slurm workload manager. All the resources allocated by Slurm are made available to PopRun.
- Removed dependency on the user locale. Avoids crashing in the case of an incorrectly configured user locale.

- Improved NUMA node binding when using cpusets. Only the NUMA nodes allowed by the current cpuset are used.
- Forward V-IPU timeout argument `--vipu-server-timeout` to IPUoF by internally passing the environment variable `IPUOF_VIPU_API_TIMEOUT`.
- Improved SSH error reporting. Instead of hanging on authentication issues, a clear error is reported.
- Automatically enable the gateway mode target option when using V-IPU.
- Added support for running programs in the current working directory without a `./` prefix for consistency with `mpirun`.
- Automatically enable NUMA awareness when there is more than one instance per host.
- Support passing `--mpi-local-args` and `--mpi-global-args` multiple times by merging the values.
- Verify the final state of partition after creation/reset. An error is reported if the partition was not created/reset correctly.
- Get V-IPU server address from local V-IPU configuration if not specified as command-line argument.
- Set the target options based on values reported by the V-IPU server.

5.3 v2.0 (Poplar SDK 2.0)

5.3.1 New features

- Added documentation
- POD native synchronisation support
- Improved input validation
- Offline mode support (running application without requiring IPUs)
- Support multi IPU-Link domain and multi-host in offline mode
- Newly created V-IPU partitions are not reset
- Ability to specify a timeout for V-IPU server requests
- Partitions created by PopRun will be automatically evicted
- PopRun will provide interactive progress status while running
- All available NUMA nodes may be used and pinned consecutively
- OpenMPI 4.0 is now bundled with the Poplar SDK, removing OpenMPI as an external dependency.
- Temporary executable caching to avoid redundant compilations on the same host
- Added verification of the number of replicas in existing partitions

5.4 v1.0 (Poplar SDK 1.4)

5.4.1 New features

- First release

CHAPTER

SIX

POPDIST CHANGELOG

6.1 v2.2 (Poplar SDK 2.2)

6.1.1 New features

- Improved error reporting in case of a missing IPU device.

6.2 v2.1 (Poplar SDK 2.1)

6.2.1 New features

- Support offline mode with PopTorch without attaching to device.
- Prevent poptorch.Options.Distributed being changed when using PopDist.
- Update to use new TensorFlow IPUConfig option configuration API.

6.3 v2.0 (Poplar SDK 2.0)

6.3.1 New features

- Added documentation
- PopTorch support
- Improved all user error messages
- ipus_per_replica is now optional when calling getDeviceId

6.4 v1.0 (Poplar SDK 1.4)

6.4.1 New features

- First release

CHAPTER

SEVEN

KNOWN ISSUES

7.1 Race condition with multiple users of the same partition

If there are multiple users of the same reconfigurable V-IPU partition, there is a potential race condition that can cause another user of the system to interfere with a PopRun launch, even if there are sufficient resources available for both users.

A PopRun launch on a reconfigurable partition consists of two phases:

1. PopRun first acquires and configures a Poplar parent device with all the IPUs required.
2. Next, each instance acquires its designated Poplar child device of the given parent device consisting of the IPUs allocated to that instance.

The race condition happens if there is another process that acquires any of the required IPUs between phase 1 and 2, and this will cause a failure when an instance tries to attach to it in phase 2 because it is already in use.

CHAPTER

EIGHT

INDEX

CHAPTER

NINE

TRADEMARKS & COPYRIGHT

PYTHON MODULE INDEX

INDEX