# Getting Started with the DPU Offload Service

## Pre-requisites

1. Access to the following repositories:  
   UCX (the most up to date dpu\_offload branch)  
   UCC-priv (the most up to date dpu\_offload branch)  
   DPU Offload Service (v0.0.4 or better)  
   OpenMPI
2. A machine with DPUs. At Nvidia the following machines will work:   
   thor  
   helios  
     
   Note: you must use the Santa Clara VPN at Nvidia to access the gateway to these clusters.

## Theory of Operation

The primary challenge in integrating the DPU offloaded MPI stack is the requirement to have multiple binary types (aarch64 and x86\_64) active and up-to-date at the same time. Mechanisms like Spack do not currently support this in a way that makes life easier rather than harder. To that end we are going to use a mix of environment variables, shell scripts, and module files to support keeping these binaries straight and up-to-date, but this is simply one mechanism for accomplishing this task and other methods may be preferred. We suggest following these instructions precisely before modifying your methods to use a better approach.

These instructions will create several environment variables (PLATFORM, SWHOME), several directories ($SWHOME, $SWHOME/modulefiles, ~/bin, ~/srun, ~/workspace), module files and shell scripts. As you proceed through these instructions you will understand why each step is required to construct the environment. It will likely be useful to read the entire instructions once before proceeding.

## Planning for Capacity Constraints

Many home spaces have insufficient capacity to handle the very large build environments needed to support the large number of code repositories we will simultaneously have active. To move these spaces to a larger scratch file system we will use symbolic links. Please confirm with your administrator that the scratch file system you are moving files to is not automatically purged.

Execute the following commands to create the symbolic links:

echo $USER #confirm this is your userid  
cd /global/scratch/users/$USER  
mkdir sw workspace  
cd

mkdir -p ~/srun  
ln -s /global/scratch/users/$USER scratch

ln -s /global/scratch/users/$USER/sw sw  
ln -s /global/scratch/users/$USER/workspace workspace

## Creating a multi-platform build environment

The first step in building an environment is creating environment variables that describes the current platform. In general, we need the instruction set and the binary versions captured in this environment. Add the following to your shell resource file to set an environment variable describing the platform (usually add this to .bashrc):  
  
#

# Platform detection

#

os=$(uname -s)

arch=$(uname -m)

if [ -d /etc/susehelp.d ]; then

os='sles'

elif [ '1' = "$(uname -r |cut -f 6 -d '.' |grep -c chaos)" ]; then

os='toss'

elif [ 'Linux' = "$os" ]; then

os=$(uname -r |grep -o -E '[a-z]+.' |head -n 1)

elif [ 'Darwin' = "$os" ]; then

os='osx'

fi

export PLATFORM=${os}-${arch}

After execution of this your PLATFORM environment variable should be set similarly to: el8-x86\_64

Second, we use this platform value to set the platform default binary location. Add the following to your shell’s sourced environment (usually .bashrc):  
  
export SWHOME=~/sw/$PLATFORM

mkdir -p $SWHOME

Now we setup some other variables that will be useful as a software developer:  
  
export MODULEPATH=~/sw/modulefiles:$MODULEPATH

export PKG\_CONFIG\_PATH="$SWHOME/lib:$PKG\_CONFIG\_PATH"

Next, be sure to reload your current environment with the variables modified above

exec $SHELL

At this point you should have a valid build environment to begin code development. **See Appendix A.1 for a complete version of this shell environment.**

## Creating a Proof-of-Concept (POC) sandbox

As a research software developer, it may be necessary to have multiple POC builds available that are simple to activate at any particular time. This means we need to have separate sandboxes for building and installing each of these POC demonstrations. To keep this straight, we will use modulefiles. The general naming of the pieces is poc-<user supplied name>. In this case the POC will be the dpu-alltoallv, and thus the POC will be called poc-dpu-alltoallv.

To create the POC sandbox execute the following commands:  
  
mkdir -p ~/workspace/poc-dpu-alltoallv  
mkdir -p ~/sw/modulefiles/poc-dpu-alltoallv  
cd ~/sw/modulefiles/poc-dpu-alltoallv

Now create a file named 1.0 with the contents shown in Appendix A.2.

To activate the sandbox, type the following:  
module load poc-dpu-alltoallv

## Building the POC software Stack

Note that you will have to build the software stack on all architectures you are using (e.g. in the case of a DPU offload you will need to build for the host (x86) and the DPU (aarch64).

First retrieve all the required repositories (Yong will grant access to UCX and UCC-priv, Geoffroy will grant access to DPU Offload service and MPI tests):

module load poc-dpu-alltoallv

poc # this will switch you to the POC build directory

git clone -b origin/topic/dpu\_offload\_v3 git@github.com:yqin/ucx

git clone git@github.com:gvallee/dpu\_offload\_service.git

git clone -b topic/dpu\_offload [git@github.com:yqin/ucc-priv](mailto:git@github.com:yqin/ucc-priv)

git clone --recurse-submodules https://github.com/open-mpi/ompi.git

git clone git@github.com:gvallee/mpi\_tests

Now you need to build these repositories. While this is simple enough to do, the build script can make this easier. It is important to note that we are going to use out of tree builds – this is important so that the same code base can be built for two different target architectures. For each repository, we will build a version into build\_x86 and a version into build\_aarch64. In the beginning you may lose track of this detail multiple times and it may be a source of difficulty. Just get used to this and try to remain diligent.

Also note that UCC requires OpenMPI and OpenMPI requires UCC. You will need to build OpenMPI at least once without UCC in order to bootstrap the UCC build the first time. (Running the build script without specifying a particular target will build and link the entire infrastructure from scratch and take care of this for you).

Create a script called build.sh with the contents shown in Appendix A.3. (Note: you may wish to remove –enable-debug flags)

You will undoubtedly edit this script many times during the development of proof-of-concept code, it’s just a starting point. To build the POC execute the following command once:  
chmod +x ./build.sh

And run the script on each platform type that you require (likely an x86 host and a DPU):

./build.sh

## Running the POC Software Stack

Note that to run the DPU prototype on a Slurm-based system we will need an allocation that includes the hosts and its DPU. There is nothing tricky to doing this, but we also provide a utility script that will help you formulate these commands.

Create a file in ~/bin named dpu-salloc-bfdev.sh with contents as shown in Appendix A.4.

This script will ensure you get an allocation of hosts and their DPUs. To print an salloc command to run type the following:

dpu-salloc-bfdev.sh -v <n>

where n is the host to use. To identify available hosts type the following:  
sinfo -p thor  
sinfo -p helios  
  
Note that nodes in the idle state which you can request. You need to get both a host AND its DPU. You can ssh to the DPU and build the DPU versions of the software by executing:

ssh thorbf00n  
module load poc-dpu-alltoallv  
poc  
./build.sh

As you gain familiarity with the stack you will come to understand which things must be rebuilt for the changes you have made.

Now it is necessary to run the software stack. For this you will need a SLURM script to run which will create all of the DPU config files and start/stop daemons as required.

In your srun directory, first create ~/srun/dpu\_utils.sh contents as shown in Appendix A.5.

Next, create ~/srun/dpu\_ucc-perf-alltoallv.sh with contents as shown in Appendix A.6.

Finally, lets run a test of the offloaded Alltoallv collective.

First, build the example using:

cd $HOME/workspace/poc-dpu-alltoallv/mpi\_tests/alltoallv && make && poc

Now submit the job using:

sbatch -N 2 -p thor --nodelist=<n> ~/srun/dpu\_alltoallv.sbatch

where <n> is an available host/dpu pair of the form “thor001, thorbf001”

# Appendix A Files

## A.1 Shell File Contents

Add to .bashrc, .profile, or similar.

#

# Platform stuff

#

os=$(uname -s)

arch=$(uname -m)

if [ -d /etc/susehelp.d ]; then

os='sles'

elif [ '1' = "$(uname -r |cut -f 6 -d '.' |grep -c chaos)" ]; then

os='toss'

elif [ 'Linux' = "$os" ]; then

os=$(uname -r |grep -o -E '[a-z]+.' |head -n 1)

elif [ 'Darwin' = "$os" ]; then

os='osx'

fi

export PLATFORM=${os}-${arch}

#

# Set binary location

#

export SWHOME=~/sw/$PLATFORM

mkdir -p $SWHOME

#

# Set sw location vars

#

export MODULEPATH=~/sw/modulefiles:$MODULEPATH

export PKG\_CONFIG\_PATH="$SWHOME/lib:$PKG\_CONFIG\_PATH"

#

# Add to PATH

#

export PATH="$HOME/bin:$SWHOME/bin:$PATH"

## A.2 Module File Contents

Contents of ~/sw/modulefiles/poc-dpu-alltoallv/1.0:

#%Module

proc ModulesHelp { } {

puts stderr "This module switches the SW environment to use the poc-dpu-alltoallv environment"

}

module-whatis "This module enables the poc-dpu-alltoallv environment"

# Retrieve environment variables

set home [getenv HOME]

set orig\_swhome [getenv SWHOME]

set platform [getenv PLATFORM]

# Setup the environment

set pocname "poc-dpu-alltoallv"

set basedir "${home}/sw/${pocname}"

set swhome "${basedir}/${platform}"

setenv SWHOME "${swhome}"

prepend-path PATH "${swhome}/bin"

prepend-path LD\_LIBRARY\_PATH "${swhome}/lib"

# Create aliases

set-alias poc "cd ${home}/workspace/${pocname}"

set-alias sw "echo ${swhome}"

## A.3 Build Script

Contents of ~/workspace/poc-dpu-alltoallv/build.sh

#!/bin/sh

# Environment setup stuff

module load poc-dpu-alltoallv

os=$(uname -r |grep -o -E '[a-z]+.' |head -n 1)

mtype=$(uname -m)

platform=${os}-${mtype}

export SWHOME=${HOME}/sw/poc-dpu-alltoallv/${platform}

export WORKSPACE=${HOME}/workspace/poc-dpu-alltoallv

# Identify what to build

if [ "x86\_64" = "$mtype" ]; then

builddir="build\_x86"

elif [ "aarch64" = "$mtype" ]; then

builddir="build\_aarch64"

else

echo "Error: Unknown machine type."

exit 1

fi

if [ ! -z "${1}" ]; then

echo "Requested targets for $SWHOME are $@"

targets="$@"

else

echo "Full build requested, deleting $SWHOME contents ..."

sleep 4s

rm -r $SWHOME/bin $SWHOME/lib $SWHOME/include $SWHOME/etc $SWHOME/share

targets="ucx dpu\_offload\_service ompi ucc ompi+ucc"

fi

# Perform builds

for t in $targets; do

echo "Building ${t} ..."

case $t in

ucx)

dir=ucx

cd ${dir} && ./autogen.sh >/dev/null

[ $? != 0 ] && exit 1

cd ..

dir=${dir}/${builddir}

mkdir -p ${dir} && cd ${dir}

../configure --prefix=$SWHOME >/dev/null

[ $? != 0 ] && exit 1

make -j $(nproc) && make install >/dev/null

[ $? != 0 ] && exit 1

cd ${WORKSPACE}

;;

dpu\_offload\_service)

dir=dpu\_offload\_service

cd ${dir} && ./autogen.sh >/dev/null

[ $? != 0 ] && exit 1

cd ..

dir=${dir}/${builddir}

mkdir -p ${dir} && cd ${dir}

../configure --prefix=$SWHOME --with-ucx=$SWHOME --enable-debug >/dev/null

[ $? != 0 ] && exit 1

make -j $(nproc) && make install >/dev/null

[ $? != 0 ] && exit 1

cd ${WORKSPACE}

;;

Contents of ~/workspace/poc-dpu-alltoallv/build.sh (continued)

ompi)

dir=ompi

cd ${dir} && ./autogen.pl >/dev/null

[ $? != 0 ] && exit 1

cd ..

dir=${dir}/${builddir}

mkdir -p ${dir} && cd ${dir}

../configure --prefix=$SWHOME --with-ucx=$SWHOME --with-pmix=internal >/dev/null

[ $? != 0 ] && exit 1

make -j $(nproc) && make install >/dev/null

[ $? != 0 ] && exit 1

cd ${WORKSPACE}

;;

ucc)

dir=ucc-priv

cd ${dir} && ./autogen.sh >/dev/null

[ $? != 0 ] && exit 1

cd ..

dir=${dir}/${builddir}

mkdir -p ${dir} && cd ${dir}

../configure --prefix=$SWHOME --enable-debug --with-ucx=$SWHOME --with-dpu-offload=$SWHOME --with-mpi=$SWHOME >/dev/null

[ $? != 0 ] && exit 1

make -j $(nproc) && make install >/dev/null

[ $? != 0 ] && exit 1

# Build the daemon

set -x

UCC\_SRC\_DIR=${WORKSPACE}/ucc-priv

DPU\_DAEMON\_BUILD=${UCC\_SRC\_DIR}/${builddir}

DPU\_DAEMON\_SRC\_DIR=${UCC\_SRC\_DIR}/src/components/tl/ucp/offload\_dpu\_daemon

echo "Daemon dirs: <$UCC\_SRC\_DIR> <${DPU\_DAEMON\_BUILD}> <${DPU\_DAEMON\_SRC\_DIR}>"

cd ${DPU\_DAEMON\_BUILD}

gcc ${DPU\_DAEMON\_SRC\_DIR}/offload\_dpu\_daemon.c -O0 -g -I${DPU\_DAEMON\_SRC\_DIR} -I${SWHOME}/include -L${SWHOME}/lib -ldpuoffloaddaemon -l ucp -l ucs -o ucc\_offload\_dpu\_daemon

[ $? != 0] && exit 1

cp ucc\_offload\_dpu\_daemon $SWHOME/bin

cd ${WORKSPACE}

;;

ompi+ucc)

dir=ompi

cd ${dir} && ./autogen.pl >/dev/null

[ $? != 0 ] && exit 1

cd ..

dir=${dir}/${builddir}

mkdir -p ${dir} && cd ${dir}

../configure --prefix=$SWHOME --with-ucx=$SWHOME --with-ucc=$SWHOME --with-pmix=internal >/dev/null

[ $? != 0 ] && exit 1

make -j $(nproc) && make install >/dev/null

[ $? != 0 ] && exit 1

cd ${WORKSPACE}

;;

\*)

echo "Unknown target ${t}!"

echo "Available targets are 'ucx', 'dpu\_offload\_service', 'ompi', 'ucc', & 'ompi+ucc'"

exit 2

;;

esac

done

echo "Completed building targets:${targets}"

## A.4 Salloc Script

## Contents of ~/bin/dpu-salloc-bfdev.sh

# Get an allocation for hosts and their DPUs

#

# Print usage

function usage {

echo "Usage: $0 <-v> <-p platform> <-n nnodes> <specific node>"

exit 1

}

#

# Main

#

# Determine if the user requested a specific platform, otherwise set default

nhosts="1"

platform="thor"

verbose="0"

while getopts "n:p:v" o; do

case ${o} in

p)

platoform=${OPTARG}

;;

n)

nhosts=${OPTARG}

;;

v)

echo "Dry run enabled"

verbose=1

;;

\*)

usage

;;

esac

done

shift $((OPTIND-1))

# Determine if the user requested a specific first node

if [ -z "$1" ]; then

begin=1

else

begin="$1"

fi

echo "$platform count=$nhosts first=$begin"

# Hosts aren't named consistently, so we have to build the names differently

for n in $(seq $begin $((begin + nhosts - 1)) ); do

if [ "thor" == "$platform" ]; then

hostnum=$(seq -w $n 999 999)

bfnum=$hostnum

nodes="${platform}${hostnum},${platform}bf${bfnum}"

elif [ "helios" == "$platform" ]; then

hostnum=$(seq -w $n 999 999)

bfnum=$hostnum

nodes="${platform}${hostnum},${platform}bf${bfnum}"

fi

if [ -z "${nodelist}" ]; then

nodelist="$nodes"

else

nodelist="${nodelist},${nodes}"

fi

done

cmd="salloc -N $((nhosts\*2)) -p ${platform} --nodelist ${nodelist} -t 120"

if [ "1" == "$verbose" ]; then

echo "$cmd"

else

$cmd

fi

## A.5 Slurm DPU Utility Script

Contents of ~/srun/dpu\_utils.sh:

#!/usr/bin/bash

#

# This script DPU utility functions

#

# Global settings (not used yet)

DPU\_PORT1=9999

DPU\_PORT2=11112

DPU\_CONFIGFILE\_PREFIX="/tmp/bws-dpu-cgf"

# Create the config file for the DPU offload engine

#

# Param dpulist Comma seperated list of hosts with DPUs

#

# Uses SLURM\_JOB\_PARTITION SLURM\_NODELIST

# Exports OFFLOAD\_CONFIG\_FILE\_PATH path to the DPU daemon config file

# Exports DPU\_OFFLOAD\_LIST\_DPUS Comma separated list of hosts with DPUs

#

function dpu\_create\_configfile

{

local dpulist="$1"

if [ -z "$dpulist" ]; then

echo "Usage: $0 <list of dpus>"

exit 1

else

echo "DEBUG: dpulist=$dpulist"

fi

local domain=".hpcadvisorycouncil.com"

if [ "thor" == "${SLURM\_JOB\_PARTITION}" ]; then

local bf\_ip\_prefix="192.168.131.1"

local hosts=$(scontrol show hostname ${SLURM\_NODELIST} |grep -v 'bf' |sed -e "s/\$/${domain}/g")

local bfs=$(scontrol show hostname ${SLURM\_NODELIST} |grep 'bf' |sed -e "s/\$/${domain}/g")

local bf\_ips=$(scontrol show hostname ${SLURM\_NODELIST} |grep 'bf' |sed -e "s/thorbf0/${bf\_ip\_prefix}/g")

local cfg\_names=$(paste <(echo "$hosts") <(echo "$bfs") -d ,)

local cfg\_names\_ips=$(paste <(echo "$cfg\_names") <(echo "$bf\_ips") -d :)

local dpu\_cfg=$(sed -e "s/\$/:${DPU\_PORT1}:${DPU\_PORT2}\\n/g" <(echo "$cfg\_names\_ips") )

elif [ "helios" == "${SLURM\_JOB\_PARTITION}" ]; then

local bf\_ip\_prefix="192.168.129.1"

local hosts=$(scontrol show hostname ${SLURM\_NODELIST} |grep -v 'bf' |sed -e "s/\$/${domain}/g")

local bfs=$(scontrol show hostname ${SLURM\_NODELIST} |grep 'bf' |sed -e "s/\$/${domain}/g")

local bf\_ips=$(scontrol show hostname ${SLURM\_NODELIST} |grep 'bf' |sed -e "s/heliosbf0/${bf\_ip\_prefix}/g")

local cfg\_names=$(paste <(echo "$hosts") <(echo "$bfs") -d ,)

local cfg\_names\_ips=$(paste <(echo "$cfg\_names") <(echo "$bf\_ips") -d :)

local dpu\_cfg=$(sed -e "s/\$/:${DPU\_PORT1}:${DPU\_PORT2}\\n/g" <(echo "$cfg\_names\_ips") )

else

echo "Unknown BF Testbed platform: ${SLURM\_JOB\_PARTITION}"

exit 1

fi

# Instead use a static file

#cp $HOME/workspace/poc-dpu-xgvmi/dpu\_offload\_service/etc/platforms/helios.cfg $OFFLOAD\_CONFIG\_FILE\_PATH

#export OFFLOAD\_CONFIG\_FILE\_PATH="$HOME/workspace/poc-dpu-xgvmi/dpu\_offload\_service/etc/platforms/helios.cfg"

)

# Create the config file

rm -f /tmp/bws-dpu-cfg.\*

DPU\_OFFLOAD\_LIST\_DPUS="$dpulist"

OFFLOAD\_CONFIG\_FILE\_PATH=$(mktemp /tmp/bws-dpu-cfg.XXXXXX)

echo "# Format version: 1" > $OFFLOAD\_CONFIG\_FILE\_PATH

echo "# <host name>,<dpu1\_hostname:dpu\_conn\_addr:interdpu-port:rank-conn-port>,..." >> $OFFLOAD\_CONFIG\_FILE\_PATH

for line in $dpu\_cfg; do

echo $line >> $OFFLOAD\_CONFIG\_FILE\_PATH

done

export OFFLOAD\_CONFIG\_FILE\_PATH

export DPU\_OFFLOAD\_LIST\_DPUS

}

# Distribute the DPU config file to al hosts

#

# Param configfile

#

# Uses SLURM\_NODELIST

#

function dpu\_send\_configfile

{

local configfile="$1"

local hostlist=$(scontrol show hostname ${SLURM\_NODELIST})

for host in $hostlist; do

scp $OFFLOAD\_CONFIG\_FILE\_PATH $host:$OFFLOAD\_CONFIG\_FILE\_PATH

[ $? != 0 ] && exit 1

done

}

# Start the DPU daemons

#

# Param Comma separated list of hosts on which to start DPU daemons

#

function dpu\_start\_daemons

{

local dpulist="$1"

local daemondir="$2"

if [ -z "$dpulist" -o -z "$daemondir" ]; then

echo "Usage: $0 <list of dpus> <daemon exec dir>"

exit 1

else

echo "DEBUG: dpulist=$dpulist"

echo "DEBUG: daemondir=$daemondir"

fi

dpu\_create\_configfile "$dpulist"

dpu\_send\_configfile "$OFFLOAD\_CONFIG\_FILE\_PATH"

# For each BF setup the environment and start it

daemonexe="${daemondir}/bin/ucc\_offload\_dpu\_daemon"

daemonenv="UCX\_NET\_DEVICES=mlx5\_0:1 \

UCX\_ZCOPY\_THRESH=0 \

UCX\_TLS=rc\_x \

UCX\_RC\_TIMEOUT=inf \

UCX\_LOG\_LEVEL=warn \

UCX\_LOG\_PRINT\_ENABLE=n \

DPU\_OFFLOAD\_DBG\_VERBOSE=1 \

OFFLOAD\_CONFIG\_FILE\_PATH=${OFFLOAD\_CONFIG\_FILE\_PATH} \

DPU\_OFFLOAD\_LIST\_DPUS=${DPU\_OFFLOAD\_LIST\_DPUS} \

LD\_LIBRARY\_PATH=${DPU\_SWHOME}/lib"

echo "DEBUG: Config file: ${OFFLOAD\_CONFIG\_FILE\_PATH}"

for dpu in $(echo $dpulist |sed "s/,/ /g"); do

daemonlog="$HOME/daemonlog-${SLURM\_JOBID}-${dpu}.out"

ssh "$dpu" "${daemonenv} nohup $daemonexe &> $daemonlog &"

#echo ssh "$dpu" "${daemonenv} gdb $daemonexe &> $daemonlog &"

echo "Daemon ($daemonexe) start status: $?"

done

~/srun/dpu\_utils.sh (continued)

local time=5s

echo "Wait $time for daemon wireup to complete"

sleep $time

}

# Stop the DPU daemons

#

# param Comma separated list of hosts with DPU

#

function dpu\_stop\_daemons {

local dpulist=$1

# Killall the daemons

for dpu in $(echo $dpulist |sed "s/,/ /g"); do

ssh $dpu "pkill -f ucc\_offload\_dpu\_daemon; [ "\$?" == "0" ] && echo \"$dpu: Daemon stopped\""

done

}

## A.7 Slurm DPU command script

Contents of ~/srun/dpu\_ucc\_perf\_alltoallv.sbatch:

##!/usr/bin/bash

#

# This script runs a set of benchmarks using DPUs

#

#SBATCH --job-name=dpu-alltoallv

##SBATCH --output=$HOME/jobs/%x-%j.out

#SBATCH --nodes=2

#SBATCH --time=00:60:00

#SBATCH --exclusive

#SBATCH -d singleton

### Platform Settings

nodes=$((SLURM\_NNODES/2))

slots=${SLURM\_CPUS\_ON\_NODE}

sockets=0

hostib="mlx5\_100:100"

case "${SLURM\_JOB\_PARTITION}" in

thor)

sockets=1

hostib=mlx5\_2:1

;;

helios)

sockets=1

hostib=mlx5\_4:1

;;

\*)

echo "This script has not been tested on partition: ${SLURM\_JOB\_PARTITION}"

exit 1

;;

esac

### Job Settings

JOB\_PPN=$slots

JOB\_NPROCS=$((nodes\*slots))

JOB\_ITER=1

JOB\_NPROCS=$((2))

### Platform/DPU software locations

export PRTE\_MCA\_plm=ssh

export SWHOME="${HOME}/sw/poc-dpu-alltoallv/el8-x86\_64"

export DPU\_SWHOME="${HOME}/sw/poc-dpu-alltoallv/bluefield-aarch64"

echo "SWHOME: $SWHOME"

echo "DPU\_SWHOME: ${DPU\_SWHOME}"

# Create a list of just the hosts and a list of just the DPUs

hostlist=$(scontrol show hostname ${SLURM\_NODELIST} |grep -v 'bf' |sed -e "s/\$/:${slots}/g" |paste -d , -s)

bflist=$(scontrol show hostname ${SLURM\_NODELIST} |grep 'bf' |sed -e "s/\$//g" |paste -d , -s)

echo "Hostname list: $hostlist"

echo "Bluefield list: $bflist"

# Import the DPU utility functions

if [ -f ~/srun/dpu\_utils.sh ]; then

source ~/srun/dpu\_utils.sh

else

echo "Unable to import: ~/srun/dpu\_utils.sh"

exit 1

fi

dpu\_stop\_daemons "$bflist"

dpu\_start\_daemons "$bflist" "${DPU\_SWHOME}"

# Run the dpu assisted MPI command

export LD\_LIBRARY\_PATH=$SWHOME/lib

echo "Offload Config file: $OFFLOAD\_CONFIG\_FILE\_PATH:"

cat $OFFLOAD\_CONFIG\_FILE\_PATH

which $SWHOME/bin/mpirun

unset SLURM\_JOBID

Contents of ~/srun/dpu\_ucc\_perf\_alltoallv.sbatch (continued)

# Run collective through an mpi test program (set coll\_ucc\_enable=1 and coll\_ucc\_priority=100)

time $SWHOME/bin/mpirun \

--np ${JOB\_NPROCS} \

--map-by ppr:${JOB\_PPN}:node:oversubscribe \

-H ${hostlist} \

--bind-to core \

--rank-by core \

--mca pml ucx \

-x UCX\_NET\_DEVICES=$hostib \

-x UCX\_TLS=rc\_x \

-x UCX\_LOG\_LEVEL=warn \

-x UCX\_LOG\_PRINT\_ENABLE=n \

--mca coll\_hcoll\_enable 0 \

--mca coll\_ucc\_enable 1 \

--mca coll\_ucc\_priority 100 \

-x UCC\_CL\_BASIC\_TLS=ucp \

-x UCC\_LOG\_LEVEL=debug \

-x OFFLOAD\_CONFIG\_FILE\_PATH=$OFFLOAD\_CONFIG\_FILE\_PATH \

-x DPU\_OFFLOAD\_DBG\_VERBOSE=1 \

stdbuf -e0 -o0 \

$HOME/workspace/poc-dpu-alltoallv/mpi\_tests/alltoallv/simple\_alltoallv 2>&1 | tee alltoallv-log

dpu\_stop\_daemons "$bflist"