Wikiprint Book

Title: System usage

Subject: DEEP - Public/User_Guide/DEEP-EST_DAM

Version: 24

Date: 19.05.2024 15:06:32

Table of Contents

System usage

The DEEP-EST Data Analytics Module (DAM) can be used through the SLURM based batch system that is also used for (most of) the Software Development Vehicles (SDV). You can request DAM nodes (`dp-dam[01-16]`) with an interactive session like this:

```
srun -A deep -N 4 --tasks-per-node 2 -p dp-dam --time=1:0:0 --pty /bin/bash -i
[kreutz1@dp-dam01 ~]$ srun -n 8 hostname
dp-dam01
dp-dam01
dp-dam02
dp-dam02
dp-dam03
dp-dam03
dp-dam04
dp-dam04
```

When using a batch script, you have to adapt the partition option within your script: `--partition=dp-dam`

Persistent Memory

Each of the DAM nodes is equipped with ?Intel's Optane DC Persistent Memory Modules (DCPMM). Whereas the first two nodes (`dp-dam[01,02]`) expose 3 TB of persistent memory the remaining nodes (`dp-dam03-16]`) have 1.5 TB of persistent memory included.

The DCPMMs can be driven in different modes. For further information of the operation modes and how to use them, please refer to the following ?information

DCPMM modes have been added to check which nodes are running in which mode, you can use the `scontrol` command:

```
scontrol show node dp-dam01 | grep AvailableFeatures
```

To select a node using a certain memory mode, you can use the constraint option within SLURM, e.g.:

```
srun -p dp-dam -N 1 -c 24 -t 0:30:0 --constraint=dpcmm_mem --pty /bin/bash
```

Using Cuda

To compile and run Cuda applications on the Nvidia V100 cards included in the DAM nodes, it is necessary to load the CUDA module:

```
[deamicis1@deepv ~]$ ml CUDA
[deamicis1@deepv ~]$ ml

Currently Loaded Modules:
 1) GCCcore/.8.3.0 (H)   2) binutils/.2.32 (H)   3) nvidia/.driver (H,g)   4) CUDA/10.1.105 (g)

 Where:
  g:  built for GPU
  H:             Hidden Module
```

Using FPGAs

Each node is equipped with a Stratix 10 FPGA. For getting started using OpenCL with the FPGAs you can find some hints as well as the slides and exercises from the Intel FPGA workshop held at JSC in:

```
/usr/local/fpga
```

It is recommended to do the first steps in an interactive session on a DAM node. To set up and check the FPGA environment, do the following:

```
source /usr/local/fpga/FPGA_init.sh
lspci | grep -i 'accel'
aocl list-devices
aoc -list-boards
# -- optional for doing the exercises:
# export CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1
```

You can copy and untar the lab into your home directory to do the exercises step by step. The exercises use the emulator device instead of the actual FPGA device due to the long compilation time for the FPGAs. For using the FPGA device you will have to compile your OpenCL kernels using `-board=pac_s10_dc` option:

```
# compile for the emulator
aoc -march=emulator -fast-emulator kernel-file.cl

# compile for the FPGA device
aoc -board=pac_s10_dc kernel-file.cl
```

In addition, you will have to adapt the OpenCL host file to select the correct platform ("Intel® FPGA SDK for OpenCL™" or "Intel® FPGA Emulation Platform for OpenCL™ (preview)").

Attention: Compiling kernels for the FPGA device (instead of the emulator) might take several hours.

Although eclipse is available on the DAM nodes, compiling and running the example applications might not work out, so you have to fall back to the command line as described in the exercise manual and by using the provided `simple_compile.sh` scripts.

Vector addition tutorial

A short tutorial to use the Intel FPGA Stratix 10 in the DAM nodes for the vector addition kernel in C++/OpenCL and PyOpenCL can be found in the DEEP-EST Gitlab: ?https://gitlab.version.fz-juelich.de/DEEP-EST/fpga_usage

Filesystems and local storage

The home filesystem on the DEEP-EST Cluster Module is provided via GPFS/NFS and hence the same as on (most of) the remaining compute nodes. The local storage system of the DAM running BeeGFS is available at

```
/work
```

The file servers are reachable through the 40 GbE interface of the DAM nodes.

This is NOT the same storage being used on the DEEP-ER SDV system. Both, the DEEP-EST prototype system and the DEEP-ER SDV have their own local storage.

It's possible to access the local storage of the DEEP-ER SDV (`/sdv-work`), but you have to keep in mind that the file servers of that storage can just be accessed through 1 GbE ! Hence, it should not be used for performance relevant applications since it is much slower than the DEEP-EST local storages mounted to `/work`.

There is node local storage available for the DEEP-EST DAM node (2 x 1.5 TB NVMe SSD), it is mounted to `/nvme/scratch` and `/nvme/scratch2`. Additionally, there is a small (about 380 GB) scratch folder available in `/scratch`. Remember that the three **scratch folders** are not persistent and **will be cleaned after your job has finished** !

Multi-node Jobs

Multi-node MPI jobs can be launched on the DAM nodes with ParaStation MPI by loading the `Intel` (or `GCC`) and `ParaStationMPI` modules.

Extoll: As of 12.12.2019, the first half of the DAM nodes (`dp-dam[01-08]`) has only GbE connectivity, while the second half has also the faster Extoll interconnect active (nodes `dp-dam[09-16]`). To run multi-node MPI jobs on the DAM nodes, it is strongly recommended to use the `dp-dam-ext` partition, which includes only the nodes providing EXTOLL connectivity. If necessary, users can also run MPI jobs on the other DAM nodes (using the `dp-dam` partition) by setting the `PSP_TCP=1` environment variable in their scripts. This will cause any MPI communication to go through the slower 40 Gb Ethernet fabric.

A release-candidate version of ParaStationMPI with CUDA awareness and GPU direct support for Extoll is currently being tested. Once released it will become available on the DAM nodes with the modules environment. Further information on CUDA awareness can be found in the ParaStationMPI section. As a temporary workaround, the current version of ParaStationMPI automatically performs device-to-host, host-to-host and host-to-device copies transparently to the user, so it can be used to run applications requiring a CUDA-aware MPI implementation (with limited data transfer performance).

For using Cluster nodes in heterogeneous jobs together with CM and/or ESB nodes, please see info about heterogeneous jobs.