# Using GP GPUs on HPC systems

UM Spring HPC Workshop

Grigory Shamov, May 19, 2023

University of Manitoba

# What are GP GPUs and what are they good for

- GP GPU ( general-purpose Graphical Processor Unit) can be used for HPC
  - Originally developed for video rendering (Games, Visualization, etc.)
    - GPUs have thousands of specialized computing units
  - NVidia is the pioneer, has largest scale of the market (AMD, etc.)
  - Accelerating math (integer, floating point in SP,  DP): MD, QM
  - Accelerating Machine Learning w tensor cores (AI/ML)

- Software needs to be rewritten for GPUs
  - Need dev  tools (CUDA, Nvidia HPC pack, libraries like CUDNN)
  - ML packages (TensorFlow, etc.)

- On HPC or cloud, users need to be able to find and specify the GPUs


NVIDIA TESLA V100 FOR PCIe

| | Tesla V100 for NVLink | Tesla V100 for PCIe | Tesla V100S for PCIe |
|---|---|---|---|
| PERFORMANCE with NVIDIA GPU Boost™ | DOUBLE-PRECISION 7.8 teraFLOPS<br>SINGLE-PRECISION 15.7 teraFLOPS<br>DEEP LEARNING 125 teraFLOPS | DOUBLE-PRECISION 7 teraFLOPS<br>SINGLE-PRECISION 14 teraFLOPS<br>DEEP LEARNING 112 teraFLOPS | DOUBLE-PRECISION 8.2 teraFLOPS<br>SINGLE-PRECISION 16.4 teraFLOPS<br>DEEP LEARNING 130 teraFLOPS |
| INTERCONNECT BANDWIDTH Bi-Directional | NVLINK 300 GB/s | PCIE 32 GB/s | PCIE 32 GB/s |
| MEMORY CoWoS Stacked HBM2 | CAPACITY 32/16 GB HBM2<br>BANDWIDTH 900 GB/s | | CAPACITY 32 GB HBM2<br>BANDWIDTH 1134 GB/s |
| POWER Max Consumption | 300 WATTS | 250 WATTS | |

# Prerequisites for a GPU calculation

- A physical GPU present ! (we will mostly talk about NVidia)
- GPU kernel drivers and libraries installed and working (check with **nvidia-smi**)
- Applications that use GPUs.
  - Some Ready-made GPU software, (Gaussian, LAMMPS, Tensorflow, etc.) or
  - CUDA for code development
    - CUDA must match the supported GPU driver version and GPU capabilities
  - NVidia HPC suite for OpenACC etc., or other GPU-based high level coding language
  - Libraries (cuBLAS, ML, Magma/Plasma, PETSc) that use GPUs
- The application might need to be told in input to use GPUs, how many, etc.

# GPU hardware available on Grex

- Grex has several GPU compute nodes, some in common use, some in “contributed” hardware
  - PI’s of the contributed hardware have priority access to their resources
  - PIs of the contributed hardware have OnDemand virtual desktop on their GPU nodes

- Local HPC resource (Grex, general use nodes)
  - Two nodes (**--partition=gpu**) of 4xV100 32GB VRAM each, NVLink, 192GB RAM, 32 CPU cores (Intel 5128)

- Local HPC resource (Grex, user-contributed  nodes)
  - Three nodes (**--partition=stamps-b**) of 4xV100 16GB VRAM each, NVLink, 192GB RAM, 32 CPU cores (Intel 5128)
  - One HGX-2 node (**--partition=livi-b**) 16x V100 32GB VRAM each, NVSwitch 1.5TB RAM, 48 CPU cores (Intel 6248R)
  - Two nodes (**--partition=aggro-b**) 2x A30 24GB VRAM each, 512 GB RAM, 24 CPU AMD 7402

- Contributors would use **livi** , **stamps** and **aggro** partitions to have preemptive access with 1h delay

# GPU capacity available on the National Systems

- National systems: every HPC and Arbutus cloud has one or more GPU “partitions”
  - GPUs span multiple generations of hardware from 2015-2021
  - Cedar, Graham, Beluga, Narval have a mix of P100s , V100s, T4, and A100s
  - Niagara has a sister GPU cluster, Mist which is a separate system
  - Arbutus OpenStack cloud provides  virtual GPUs (V100s)
  - https://docs.alliancecan.ca/wiki/Using_GPUs_with_Slurm

- GPUs on the Arbutus Cloud are chosen by using a VM “flavour” and installing NVidia software into the guest VM
  - https://docs.alliancecan.ca/wiki/Using_cloud_GPUs

# How to request a GPU on Grex HPC system

- SLURM syntax for GPUs
  - You will need to select a Partition that has GPUs! (**--partition=stamps-b**)
  - You will need to specify number of GPUs and other resources (CPU, mem, time)
  - Something called cons_TRES; **–gpus=N**; **–gpus-per-node=N**; **–mem-per-gpu=M**
  - Not all combinations of –nodes, –ntasks and –X-per-Y are sensible!

- How much CPUs and memory per GPU is to ask?
  - Start with average (i.e., on the 4x V100 node of 32 GPU, **–cpus-per-gpu=8 –mem-per-cpu=4000M**)

- Interactive job example w salloc
  - ***salloc --partition=stamps-b --gpus=1 --cpus-per-gpu=8 --mem-per-cpu=4000***
  - Try **nvidia-smi** ; try a sample from /global/software/cuda/11.4.3-gcc48/samples

- Batch job example with sbatch : same, needs a job script with cuda modules loaded

# How to request a GPU on DRAC HPC systems

- SLURM syntax for GPUs https://docs.alliancecan.ca/wiki/Using_GPUs_with_Slurm .
  - No partitions on DRAC systems!
- There are two variants of SLURM syntax, the newer "gres" and the older "cons_tres". DRAC doc recommends the new one; but the systems still support the old one

```
--gpus-per-node=[type:]number or  --gres=gpu[[:type]:number]
```

- GPUs of very different characteristics (V100 vs T4 on Graham, for example) have to be specified explicitly in SLURM:
  - `--gres=gpu:v100:1    # One V100 card per job`
  - `--gres=gpu:t4:2      # Two T4 cards`

- On Cedar too: depending on memory:
  - P100 with 12 GB is "p100" `--gpus-per-node=p100:1`
  - P100 with 16 GB is "p100l" `--gpus-per-node=p100l:1`

- How much CPUs and memory per GPU is to ask?
  - Recommended values: **GPU-equivalent** of the other resources (CPU, memory).
  - On a 32CPU node with 4 GPUs, ask for 8 CPUs for a 1-GPU job

# GPU software

- Canned GPU codes, commercial : Gaussian, etc.; Guppy (bioinformatics); Matlab
- ML Packages (Tensorflow etc.) – can installed via a Python packaging like Conda.
- Compiling your own software;
  - Need “**module load gcc/$ver**” or “**module load intel/$ver**” first, then “**module load cuda/$ver**”
  - CUDA versions 10.2, 11.3 and 11.7 are available on Grex (**module spider cuda**) . Gives **nvcc**
  - After loading the modules, proceed with cmake  or configure, make etc. as per package’s instruction
  - Some GPU codes need NVidia HPC toolkit (Portland Group compilers for OpenACC)
- Containers: Singularity (now Apptainer) and NVidia NGC repository
  - https://catalog.ngc.nvidia.com/
  - Get package from NGC Cloud using **singularity pull**
  - Run **singularity exec** as described (often requires bind-mounting container directories)

# Demo on Grex

Follow along if have a CCDB account! Instructions: /global/software/ws2023 on Grex.
(SSH to **yak.hpc.umanutoba.ca**, *ls /global/software/ws2023*)

- Building and running a CUDA example
  - *cat /global/software/ws2023/1-gpu-nvidia-sample.txt*
- Fetching and Running an NGC container
  - *cat /global/software/ws2023/2-gpu-NGC-lammps-container.txt*
- Running an ML Python package in a Jupyter notebook/tunnel
  - *cat /global/software/ws2023/3-gpu-OpenAI-diffusion.txt*

*1. Connect to Grex*
ssh -Y user_name@yak.hpc.umanitoba.ca

*2. Get a GPU in an interactive job*
salloc --partition=gpu --gpus=1 --cpus-per-gpu=6 --mem=12000

*3. Load modules for CUDA*
module load gcc/11.2 cuda/11.7 cmake git

*4. run a NVidia sample*
nvidia-smi
$CUDA_HOME/extras/demo_suite/deviceQuery
$CUDA_HOME/extras/demo_suite/bandwidthTest

*5. check options with --help, try different jobs with different devices, more than one GPU*

*1. Connect to Grex*
ssh -Y user_name@yak.hpc.umanitoba.ca

*2. Get a GPU in an interactive job*
salloc --partition=gpu --gpus=1 --cpus-per-gpu=6 --mem=12000

*3. Load modules for CUDA and singularity*
module load gcc/11.2 cuda/11.7 singularity

*4. Check it all works*
singularity version
nvidia-smi

*5. Get the LAMMPS code from NGC cloud. Also, get the Lennard Jones input and the run script.*

singularity pull docker://nvcr.io/hpc/lammps:patch_3Nov2022

wget https://lammps.sandia.gov/inputs/in.lj.txt
wget https://gitlab.com/NVHPC/ngc-examples/-/raw/master/lammps/single-node/run_lammps.sh

*Edit the run_lammps to set device count properly, or use the one in this directory*

*6. Actually run the LAMMPS using singularity, container image, and input from 5.*
singularity run --nv -B $PWD:/host_pwd --pwd /host_pwd ./lammps_patch_3Nov2022.sif ./run_lammps.sh

*1. Connect to Grex, grab a GPU similar to the above slides*
*2. Load modules for CUDA and Python*
module load gcc/11.2 cuda/11.7 cmake git python/3.11 cudnn
*3. Create a Virtualenv called* openai, *install dependencies such as Torch and Pytorch3D, also Jupyter for notebooks. Takes time…*
virtualenv --no-download openai
source openai/bin/activate
pip install jupyter
pip install torch==2.0.0+cu117 torchvision==0.15.1+cu117 torchaudio==2.0.1 --index-url https://download.pytorch.org/whl/cu117
pip install "git+https://github.com/facebookresearch/pytorch3d.git"

*4. Obtain OpenAI models from github, install into the Virtualenv*
git clone https://github.com/openai/shap-e && cd shap-e && pip install -e .
cd ..
git clone https://github.com/openai/point-e && cd point-e && pip install -e .
cd ..

*6. Start Jupyter Notebook Server on the node, using the Virtualenv. Note the access token, Do not close the interactive job session!*
# know your node's ip; assume it shows 10.22.33.44 in the output. Pick a free port like 8888 when starting the Jupyter
ping -c 1 `hostname`.local
jupyter notebook --ip  10.22.33.44 --port 8888

*7. In another terminal, open an SSH Tunnel to the host/port. Use IP and port from the point6! Do not close the terminal*
ssh -fNL 8888:10.22.33.44:8888  user_name@yak.hpc.umanitoba.ca

8. *In a local browser (Firefox, Chrome, etc.) open localhost URL with the same port fom 6*; http://localhost:8888 ; *use the access token as printed by Jupyter server. Navigate in the browser to the point-e and shap-e examples (*.ipynb *files) and run the models.*


University
of Manitoba