The OSU College of Engineering DGX System
for Advanced GPU Computing

Mike Bailey
mjb@cs.oregonstate.edu

OSU’s College of Engineering bought six Nvidia DGX-2 systems

Each DGX server:
- Has 16 Nvidia Tesla V100 GPUs
- Has 28TB of disk, all SSD
- Has two 24-core Intel Xeon 8168 Platinum 2.7GHz CPUs
- Has 1.5TB of DDR4-2666 System Memory
- Runs the CentOS 7 Linux operating system

Overall compute power:
- Each V100 Nvidia Tesla card has 5,120 CUDA Cores and 640 Tensor Cores
- This gives each 16-V100 DGX server a total of 81,920 CUDA cores and 10,240 Tensor cores
- This gives the entire 6-DGX package a total of 491,520 CUDA Cores and 61,440 Tensor Cores

Performance Comparison with one of our previous Systems

How to SSH to the DGX Systems

 Bord with one of our previous Systems

BTW, you can also use the rabbit machine:

ssh rabbit.engr.oregonstate.edu
It is a good place to write your code and get it to compile. It is not a good place to run your code.

BTW, you can also use the rabbit machine:

ssh rabbit.engr.oregonstate.edu
It is a good place to write your code and get it to compile. It is not a good place to run your code.

ssh over to a DGX submission machine --
submit-a and submit-b will also work

submit-c 142% module load slurm
Type this right away to set your path correctly.
How to Check on the DGX Systems

Check on the queues:

```
squeue
```

Your partitions:

```
sinfo
```

System Information:

```
PARTITION AVAIL TIMELIMIT NODES STATE NODELIST
share* up 7-00:00:00 2 drain compute-4-
share* up 7-00:00:00 1 mix compute-3-
sharegpu up 7-00:00:00 3 idle compute-dgps-[2-3],compute-gpu
dgps up 7-00:00:00 1 drain compute-dgps-2
dgps up 7-00:00:00 5 mix compute-dgps-[1,3-4]
gpu up 7-00:00:00 1 idle compute-gpu
GPU up 7-00:00:00 1 down compute-gpu
DGX up 7-00:00:00 3 mix compute-dgps-[4-6]
dgps up 7-00:00:00 1 mix compute-dgps-1
dgps up 7-00:00:00 2 idle compute-dgps-[2-3]
dbgxclass* up 1:00:00 2 idle compute-dgps-[2-3]
dbgxclass* up 7-00:00:00 1 mix compute-2-
```

Submitting a CUDA job to the DGX Systems using Slurm

Create a shell file:

```
#!/bin/bash

#SBATCH -J MatrixMul
#SBATCH -A cs475-575
#SBATCH -p class
#SBATCH --gres=gpu:1
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=joeparallel@oregonstate.edu

/usr/local/apps/cuda/cuda-10.1/bin/nvcc -o matrixMul matrixMul.cu
```

Submit the job described in your shell file:

```
sbatch submit.bash
```

Check the output:

```
cat matrixmul.err
```

Submitting a Loop

Create a shell file:

```
#!/bin/bash

#SBATCH -J MatrixMul
#SBATCH -A cs475-575
#SBATCH -p class
#SBATCH --gres=gpu:1
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=joeparallel@oregonstate.edu

for t in 1 2 4 8 16 32
    do
        /usr/local/apps/cuda/cuda-10.1/bin/nvcc -DNUMT=$t -o matrixMul matrixMul.cu
        ./matrixMul
    done
```

Submit the job described in your shell file:

```
sbatch submitloop.bash
```

Displays the latest output added to matrixmul.err. Keeps doing it forever.

Control-c to get out of it.
Results for Multiplying two 1024x1024 Matrices

![Graph showing GigaFlops during Matrix Multiplication](image)

Use slurm's `scancel` if your Job Needs to Be Killed

```bash
submit-c 163% sbatch submitloop.bash
Submitted batch job: 475

submit-c 164% scancel 475
```

Submitting an OpenCL job to the DGX Systems using Slurm

```bash
#!/bin/bash
#SBATCH -J MatrixMult
#SBATCH -A cs475-575
#SBATCH -p class
#SBATCH --gres=gpu:1
#SBATCH -o printinfo.out
#SBATCH -e printinfo.err
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=joeparallel@oregonstate.edu
g++ -o printinfo printinfo.cpp /usr/local/apps/cuda/cuda-10.1/lib64/libOpenCL.so.1.1 -lm -fopenmp
./printinfo
```

Here's What `printinfo` Got on the DGX System

```plaintext
Number of Platforms = 1
Platform #0:
  Name = 'NVIDIA CUDA'
  Vendor = 'NVIDIA Corporation'
  Version = 'OpenCL 1.2 CUDA 10.1.351'
  Profile = 'FULL_PROFILE'
  Number of Devices = 1

Device #0:
  Type = 0x0004 = CL_DEVICE_TYPE_GPU
  Device Vendor ID = 0x10de (NVIDIA)
  Device Maximum Compute Units = 80
  Device Maximum Work Item Dimensions = 3
  Device Maximum Work Item Sizes = 1024 x 1024 x 64
  Device Maximum Work Group Size = 1024
  Device Maximum Clock Frequency = 1530 MHz
  Device Extensions:
    cl_khr_global_int32_base_atomics
    cl_khr_global_int32_extended_atomics
    cl_khr_local_int32_base_atomics
    cl_khr_local_int32_extended_atomics
    cl_khr_fp64
    cl_khr_byte_addressable_store
    cl_khr_int8_pointer
    cl_khr_gl_sharing
    cl_khr_global_mem_opencl_sharing
    cl_khr_1d_image
    cl_khr_1d_image_win_handle
    cl_khr_3d_image
    cl_khr_3d_image_win_handle
    cl_khr_memfence
    cl_khr_copy_buffer
    cl_khr_copy_buffer_subregion
    cl_khr_mem_lock
    cl_khr_phone
    cl_khr_phone_single_threaded
    cl_khr_phone_thread_group
    cl_khr_opencl_profiling
    cl_khr_fp16
    cl_khrclk
    cl_khr_copy_image
    cl_khr_copy_image_cl_buffer
    cl_khr_copy_image_cl_mem
    cl_khr_subdivide_image
    cl_khr_image2dMS
```

(A CUDA block was actually NUMT \times NUMT threads)