Guide: OpenACC on AWS

Ignacio M. Llorente, David Sondak

v2.0 - February 22, 2020

Abstract

This is a guideline document to show the necessary actions to set up the system to use OpenACC in GPU-base accelerated computing instances on AWS.

Requirements

- **First you should have followed the Guide "First Access to AWS"**. It is assumed you already have an AWS account and a key pair, and you are familiar with the AWS EC2 environment.
- Take into account that **GPU-powered instances are expensive**.
- The files needed to do the exercises are available for download from **Canvas**.

Acknowledgments

The author is grateful for constructive comments and suggestions from David Sondak, Charles Liu, Matthew Holman, Keshavamurthy Indireshkumar, Kar Tong Tan, Zudi Lin and Nick Stern.

1. Configure the VM

- Launch an instance with "**Ubuntu Server 16.04**" as AMI and "g3.4xlarge" as instance type. This is an instance powered by one NVIDIA Tesla M60 GPU with 8 GiB of GPU memory and 2048 parallel processing cores. [Your default account may not allow you to use any GPUs (including g3.4xlarge). In that case, via "support" on AWS dashboard, request access to g3.4xlarge.]

- You should include the internal hostname and IP to /`etc/hosts`. You will find these under Description once the instance is up and running. In my specific case:

```
$ cat /etc/hosts
127.0.0.1 localhost
172.30.4.157 ip-172-30-4-157
```

- Check the availability of the GPU within the running instance

```
$ lspci | grep -i nvidia

00:1e.0  VGA  compatible  controller:  NVIDIA  Corporation
GM204GL [Tesla M60] (rev a1)
```

- By default the EBS volume is only 8GiB and we need 128GiB.

```
$ df -h

Filesystem      Size  Used Avail Use% Mounted on
udev             60G     0   60G   0% /dev
tmpfs            12G  8.7M   12G   1% /run
/dev/xvda1        8G    6G  4.0G  69% /
```

- Go to the AWS control panel and in the Volumes section of the EC2 dashboard find your EBS partition and resize its volume. Then within the running system you have to extend the Linux File System.

```
$ sudo growpart /dev/xvda 1

CHANGED:   disk=/dev/xvda   partition=1:   start=4096   old:
size=16773086,end=16777182 new: size=73396190,end=73400286
```

- A look at the `lsblk` output confirms that the partition `/dev/xvda1` now fills the available space on the volume `/dev/xvda`:

```
$ lsblk

NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
...
```

```
xvda    202:80   0  128G  0 disk
└─xvda1 202:81   0  128G  0 part
```

- Use a file system-specific command to resize each file system to the new volume capacity. For a Linux ext2, ext3, or ext4 file system, use the following command, substituting the device name to extend:

```
$ sudo resize2fs /dev/xvda1
```

- Make sure we have some basic packages installed on Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install build-essential
```

- The `gcc` version I'm using is 5.x

```
$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609
```

2. Install CUDA

- Use `wget` from the EC2 instance

```
$ wget
http://developer.download.nvidia.com/compute/cuda/repos/ubu
ntu1604/x86_64/cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
```

- We should now have a deb file called `cuda-repo-ubuntu1604_8.0.61-1_amd64.deb` in the home directory. Run the following commands to install CUDA:

```
$ sudo dpkg -i cuda-repo-ubuntu1604_8.0.61-1_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

- Now you can check the CUDA installation:

```
$ nvidia-smi

Tue Oct 17 14:47:53 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.81                 Driver Version: 384.81                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M60           Off  | 00000000:00:1E.0 Off |                    0 |
| N/A   28C    P0    40W / 150W |      0MiB /  7613MiB |     98%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- Check the time needed to run this command. This is not reasonable. It turns out this is due to the default configurations being suboptimal. You can follow the steps below to re-configure the GPU settings:
  - Configure the GPU settings to be persistent

```
$ sudo nvidia-smi -pm 1
```

  - Disable the autoboost feature for all GPUs on the instance

```
$ sudo nvidia-smi --auto-boost-default=0
```

  - Set all GPU clock speeds to their maximum frequency

```
$ sudo nvidia-smi -ac 2505,875
```

- Running `nvidia-smi` should now be faster!

3. Install PGI Community Edition

PGI Community Edition includes a no-cost license to a recent release of the PGI Fortran, C and C++ compilers and tools for multicore CPUs and NVIDIA Tesla GPUs, including all OpenACC, OpenMP and CUDA Fortran features.

- Download `pgilinux-2019-1910-x86-64.tar.gz` from `https://www.pgroup.com/products/community.htm` and perform the following installation steps
  1. Download the file from the link onto your local machine (it's a big file)
  2. Copy it to the AWS instance `$ scp -i *course_key* *local filepath* *remote filepath*`

Once the file is on the VM instance, proceed to unpack it using the following command.

```
$ tar -xzvf pgilinux-2019-1910-x86-64.tar.gz

$ sudo ./install
```

During install you will need to go through the following steps:

1. Sign the EULA license by hitting **q** where you should type “**accept**” to accept the license.
2. You will then be asked if you want to do a single system install or a network. Choose **1 for single system**
3. Then you will be asked which directory you would like to install in. Press **enter** to keep the default /opt/pgi
4. You will be asked if you want to update the links in the 2019 directory. Press **y** for yes and then hit **enter** to continue
5. You will be asked two questions about MPI and GPU support for OpenMPI. Go ahead and press **y** for both of these and hit enter to continue
6. Finally, you will be asked two last questions about using a professional license and setting read/write permissions. Answer **no** to both of these.

- Configure your shell environment.

```
$ export PGI=/opt/pgi;
$ export PATH=/opt/pgi/linux86-64/19.10/bin:$PATH;
$ export MANPATH=$MANPATH:/opt/pgi/linux86-64/19.10/man;
$ export LM_LICENSE_FILE=$LM_LICENSE_FILE:/opt/pgi/license.dat;
```

- Run `pgaccelinfo` to see that your GPU and drivers are properly installed and available. For NVIDIA, you should see output that looks something like the following:

```
$ pgaccelinfo

CUDA Driver Version:           9000
NVRM version:                    NVIDIA UNIX x86_64 Kernel Module  384.81
Sat Sep  2 02:43:11 PDT 2017
Device Number:                 0
Device Name:                   Tesla M60
Device Revision Number:        5.2
Global Memory Size:            7983005696
Number of Multiprocessors:     16
Concurrent Copy and Execution: Yes
Total Constant Memory:         65536
Total Shared Memory per Block: 49152
Registers per Block:           65536
Warp Size:                     32
Maximum Threads per Block:     1024
Maximum Block Dimensions:      1024, 1024, 64
Maximum Grid Dimensions:       2147483647 x 65535 x 65535
Maximum Memory Pitch:          2147483647B
Texture Alignment:             512B
Clock Rate:                    873 MHz
Execution Timeout:             No
```

```
Integrated Device:              No
Can Map Host Memory:            Yes
Compute Mode:                   default
Concurrent Kernels:             Yes
ECC Enabled:                    Yes
Memory Clock Rate:              2505 MHz
Memory Bus Width:               256 bits
L2 Cache Size:                  2097152 bytes
Max Threads Per SMP:            2048
Async Engines:                  2
Unified Addressing:             Yes
Managed Memory:                 Yes
PGI Compiler Option:            -ta=tesla:cc50
```

4. Our First OpenACC Program

- Upload to the VM the `acc_sc.c` code, compile it with `pgcc`, and run the code on the GPU. Use options `-acc` to support OpenACC and `–Minfo` to provide verbose info:

```
$ pgcc -acc -Minfo acc_sc.c -o acc_sc
```

```
vecaddgpu:
      4, Generating copyin(a[:n])
         Generating copyout(r[:n])
         Generating copyin(b[:n])
      5, Loop is parallelizable
         Accelerator kernel generated
         Generating Tesla code
       5, #pragma acc loop gang, vector(128)/* blockIdx.x threadIdx.x */
```

- Run the code

```
$ ./acc_sc
```

- You should see the output

```
0 errors found
```

- You can enable additional output by setting environment variables.

```
$ export PGI_ACC_NOTIFY=1
```

- After executing the code you should see the output

```
launch      CUDA      kernel           file=/home/ubuntu/acc_sc.c
   function=vecaddgpu  line=5  device=0  threadid=1  num_gangs=782
   num_workers=1 vector_length=128 grid=782 block=128
```

- The extra output tells you that the program launched a kernel for the loop at line 5, with a CUDA grid of size 782, and a thread block of size 128.

- if you set the environment variable `PGI_ACC_NOTIFY` to 3, the output will include information about the data transfers as well:

```
upload CUDA data  file=/home/ubuntu/acc_sc.c function=vecaddgpu
   line=4 device=0 threadid=1 variable=a bytes=400000

upload CUDA data  file=/home/ubuntu/acc_sc.c function=vecaddgpu
   line=4 device=0 threadid=1 variable=b bytes=400000

launch     CUDA     kernel          file=/home/ubuntu/acc_sc.c
   function=vecaddgpu  line=5  device=0  threadid=1  num_gangs=782
   num_workers=1 vector_length=128 grid=782 block=128

download     CUDA     data          file=/home/ubuntu/acc_sc.c
   function=vecaddgpu  line=6  device=0  threadid=1  variable=r
   bytes=400000
```

- If you set the environment variable `PGI_ACC_TIME` to 1, the runtime summarizes the time taken for data movement between the host and GPU, and computation on the GPU.

```
Accelerator Kernel Timing data

/home/ubuntu/acc_sc.c

  vecaddgpu  NVIDIA  devicenum=0

    time(us): 162

    4: compute region reached 1 time

        5: kernel launched 1 time

            grid: [782]  block: [128]

             device time(us): total=7 max=7 min=7 avg=7

            elapsed time(us): total=550 max=550 min=550 avg=550

    4: data region reached 2 times

        4: data copyin transfers: 2

             device time(us): total=105 max=59 min=46 avg=52

        6: data copyout transfers: 1
```

```
            device time(us): total=50 max=50 min=50 avg=50
```

- This tells you that the program entered one accelerator region and spent a total of about 162 microseconds in that region. It copied two arrays to the device, launched one kernel and brought one array back to the host.

Stop your instances when are done for the day to avoid incurring charges
Terminate them when you are sure you are done with your instance