Quick Start

Policies

Before accessing MEL, users should become familiar with the policies governing its use.

- Users of MEL are expected to follow the University policies regarding information technology usage. The Biological Sciences Division has additional policies due to HIPAA compliance. If a computer or other resource is not explicitly described as HIPAA-certified, do not store PHI on it. In particular, MEL and its associated local and network storage are NOT HIPAA-certified.
- In addition, there are usage policies local to MEL, which are intended to allow multiple users to co-exist peacefully while maintaining an equitable level of service to each. These policies will change on occasion with accompanying changes in hardware and software. Changes will be announced via the MEL users mailing list. Users are assumed to have read the University and local policies before accessing MEL.
- Policy violations will result in the temporary or permanent revocation of facility access rights, depending on their nature and severity.

As of 23 January 2020, MEL is still in a gray area between beta testing and full production status. The one remaining system problem is an incompatibility between the way the Superdome Flex and the nVidia GPUs handle hardware timeouts due to memory access contention on the Flex Fabric interconnect. Only a handful of programs have so far triggered this problem; however, the result is a full system reboot. Since MEL is a shared multi-user system a reboot would affect everyone, and thus must be avoided if possible. HPE is waiting for a firmware update from nVidia to reconcile the two different ways of setting hardware timeouts, but until then, users should treat the procedures in the Running Non-GPU and GPU Accelerated Programs sections below as mandatory rather than advisory.

Accessing MEL

After obtaining an account, you can use the NoMachine remote graphical desktop client to access MEL. If the NoMachine client is not installed on your device, you can download a version for your platform at https://www.nomachine.com/download-enterprise#NoMachine-Enterprise-Client.

Clients for iOS and Android are available at their respective app stores.

Under Linux, if you cannot find the NoMachine client in your desktop's application menu, you can start it from the command line via the path /usr/NX /bin/nxplayer.

MEL’s Internet host name is mel.bsd.uchicago.edu. The NoMachine client must be configured to use Secure Shell (SSH) as the communication protocol, and the non-standard port 443 instead of the usual Secure Shell port 22. The BSD network firewall blocks from off-campus both the standard SSH port 22 as well as NoMachine's proprietary protocol, which communicates over port 4000.

You can also use Secure Shell to transfer files to your account using scp or sftp on Linux and MacOS, and the WinSCP client for Windows. There are also sftp/scp clients available for iOS and Android.

The NoMachine client will start the XFCE4 desktop environment. XFCE4 was chosen as a lightweight desktop which resembles a Windows desktop and which does not rely on 3D hardware acceleration.

User Environment

If you previously had access to the retired SIRAF cluster, you will see the same files in your home directory. The former SIRAF file server has been repurposed and upgraded with a 10Gbps network connection to MEL.

In addition to your home directory, you also have a private scratch directory under /scratch/<your login name>. The /scratch filesystem is built on top of high speed NVMe SSDs and is capable of >1 GBps read/write in both streaming (large files) and latency-bound (small files) I/O. The /scratch filesystem should be your default destination for writing results. Copying small files to /scratch for reading may also be advantageous, instead of reading directly from your home directory or another network share.

Note that your home directory is backed up weekly to tape storage. /scratch however is not on the backup schedule, so any results which you want to store permanently should be moved to your home directory.

Running Non-GPU Programs

Programs which do not use GPU acceleration can be started from the desktop environment's application launcher or via a command line terminal without any special steps. There is no need to use a cluster job scheduler like SLURM, Grid Engine, or PBS. For computationally intensive tasks which use multiple CPU cores, use the numactl command to pin programs to a fixed set of CPU sockets/nodes (for the Superdome Flex, a CPU socket is equivalent to a computation node). numactl will keep processes and memory together on the same nodes, and improve performance by reducing traffic across the Flex Fabric network.

The numactl utility is executed by prepending it to the rest of your command, e.g.

```
numactl -N <set of nodes/sockets> <command>
```

e.g. numactl -N 8,9 /opt/anaconda3/bin/ipython would start ipython on CPU sockets 8 and 9. The threads spawned by ipython would run only on CPU sockets 8 and 9, and access only the memory directly attached to those sockets. Each socket is directly connected to 192GB of memory, so the maximum amount of memory the program could access would be 384GB.
Because of the ongoing issues with the nVidia GPU timeouts, users should restrict their jobs to adjacent socket pairs: 0,1 - 2,3 - 4,5 - .... - 14,15. Adjacent sockets are connected directly by the Intel Ultra-Path Interconnect (UPI), so traffic between the sockets avoids the Flex Fabric network and will not trigger a GPU hardware timeout error.

For a detailed discussion of how and why to use numacl as well as other process placement utilities see this page:

https://www.glennklockwood.com/hpc-howtos/process-affinity.html

Note that these optimization techniques also apply to running jobs on distributed memory clusters such as the RCC’s Midway2 or any platform with multiple CPU sockets.

Running GPU Accelerated Programs

The 16 CPUs in MEL are capable of 9.6 TFLOPS in total of double precision floating point (FP64) performance. The 16 GPUs in MEL are capable of 112 TFLOPS of FP64, 224 TFLOPS of single precision (FP32), and 448 TFLOPS of half-precision (FP16) in total, using only their general purpose graphical processing cores. Each nVidia Tesla V100 also has 640 tensor cores for mixed FP16/FP32 add/multiply/accumulate operations, which increases performance to 114 TFLOPS per GPU, or >1800 TFLOPS in total using the tensor cores. If performance is important to you, please use applications with GPU support.

Without properly associating a GPU with its controlling CPU, the time expended in CPUGPU memory transfers can outweigh the GPU’s acceleration. Unlike CPU, memory, and I/O, GPUs are not managed by the operating system. That means that the OS does not load balance or schedule GPU jobs. On a multi-GPU system with multiple users, by default all the jobs will run on the first GPU (GPU 0) until it crashes from memory exhaustion or overheating, while all the other GPUs sit idle. To prevent this, users are assigned a random GPU at login by setting CUDA_VISIBLE_DEVICES to a number between 0-15.

We are currently investigating automated GPU management applications, but for now users of MEL will have to manually schedule their GPU jobs.

The general procedure to choose a GPU for your program is to

1. Use the command sort-gpus.sh to output a list of GPUs sorted by activity. The output will look like

<table>
<thead>
<tr>
<th>gpu #</th>
<th>pid #</th>
<th>type</th>
<th>sm %</th>
<th>mem %</th>
<th>enc %</th>
<th>dec %</th>
<th>command</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2093</td>
<td>C</td>
<td>32</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>gpu_programX</td>
</tr>
<tr>
<td>9</td>
<td>2143</td>
<td>C</td>
<td>55</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>gpu_programY</td>
</tr>
</tbody>
</table>

- denotes that a resource is not being accessed.
gpu # is the GPU index number. On MEL, it will range from 0 to 15 since there are 16 GPUs.
pid # is the process ID number of the program thread running its calculations on the GPU.
type is the type of job: Compute (C) or Graphical (G). Almost all the jobs will be of type C, but in the future visualization jobs will be type G.
sm % is the percentage of Streaming Multiprocessors (SM) being used by the application. This is analogous to the total CPU % usage and shows how much computation is being performed. The nVidia Tesla V100 has 5120 SM units, so an sm % of 55 would indicate 2816 of them are active and devoted to the application.
mem % is the total percentage of GPU memory in use. The V100 has 32GB of memory, so mem % of 6 would be 1.92GB.
en % and dec % record the % usage of the onboard video encoder and decoder. Unless you are transcoding video formats, these should be 0 or unused.
command is the name of the application from the operating system process table.

2. Choose a GPU which has a "-" in the pid # column. If all GPUs have actively running processes on them, choose one with the smallest sm % which has enough free memory \((1.0 - 0.01*mem \%) \times 32GB\) to accommodate your job. If you don’t know how much memory your job uses, choose the GPU which has the smallest mem %.

3. Set the environment variable CUDA_VISIBLE_DEVICES to select the GPU chosen in step 2:

export CUDA_VISIBLE_DEVICES=\$N, N=0, ... , 15

Note: some applications do not honor the CUDA_VISIBLE_DEVICES environment variable and have their own way of selecting a GPU. However, both Tensorflow and Matlab do honor CUDA_VISIBLE_DEVICES.

Use numacl to bind the program to the CPU socket which is on the same PCIe segment as the GPU:

numacl -N \$<same CPU socket # as GPU index #> \$<your GPU accelerated program>,
e.g.

export CUDA_VISIBLE_DEVICES=8; numacl -N 8 /opt/anaconda3/bin/ipython

Binding the CPU socket on the same PCIe bus as the GPU ensures faster communication between GPU and CPU memory and will prevent saturating the Flex Fabric interconnect.

Multi-GPU applications are an advanced topic which will be covered in a separate section.