Argon Cluster

- Hyperthreaded Cores (HT)
- Job Scheduler/Resource Manager
  - new SGE utilities
  - SSH to compute nodes
- Software Packages
- Environment Modules
- Setting default shell
- Queues and Policies
  - The University of Iowa (UI) queue
  - Resource requests

The Argon HPC system is the latest HPC system of the University of Iowa. It consists of 346 compute nodes, each of which contain 28 2.4GHz Intel Broadwell processor cores, running CentOS-7.4 Linux. There are several compute node configurations,

1. standard memory 128GB
2. mid-memory 256GB
3. high-memory 512GB

There are 24 machines with Nvidia P100 accelerators and 2 machines with an Nvidia K80 accelerator. The $R_{\text{peak}}$ (theoretical Flops) is 285.60 TFlops, not including the accelerators, with 67.25 TB of memory. In addition, there are 2 login nodes of the same system architecture. The login nodes have 256GB of memory.

While on the backend Argon is a completely new architecture, the frontend should be very familiar to those who have used previous generation HPC systems at the University of Iowa. There are, however, a few key differences that will be discussed in this page.

Hyperthreaded Cores (HT)

One important difference between Argon and previous systems is that Argon has Hyperthreaded processor cores turned on. Hyperthreaded cores can be thought of as splitting a single processor into two virtual cores, much as a Linux process can be split into threads. That oversimplifies it but if your application is multithreaded then hyperthreaded cores can potentially run the application more efficiently. For non-threaded applications you can think of any pair of hyperthreaded cores to be roughly equivalent to two cores at half the speed if both cores of the pair are in use. This can help ensure that the physical processor is kept busy for processes that do not always use the full capacity of a core. The reasons for enabling HT for Argon are to try to increase system efficiency on the workloads that we have observed. There are some thing to keep in mind as you are developing your workflows.

1. For high throughput jobs the use of HT can increase overall throughput by keeping cores active as jobs come and go. These jobs can treat each HT core as a processor.
2. For multithreaded applications, HT will provide more efficient handling of threads. You must make sure to request the appropriate number of job slots. Generally, the number of job slots requested should equal the number of cores that will be running.
3. For non-threaded CPU bound processes that can keep a core busy all of the time, you probably want to only run one process per core, and not run processes on HT cores. This can be accomplished by taking advantage of the Linux kernel's ability to bind processes to cores. In order to minimize processes running on the HT cores of a machine make sure that only half of the total number of cores are used. See below for more details but requesting twice the number of job slots as the number of cores that will be used will accomplish this. A good example of this type of job is non-threaded MPI jobs, but really any non-threaded job.

Job Scheduler/Resource Manager

Like previous UI HPC systems, Argon uses SGE, although this version is based off of a slightly different code-base. If anyone is interested in the history of SGE there is an interesting writeup at History of Grid Engine Development. The version of SGE that Argon uses is from the Son of Grid Engine project. For the most part this will be very familiar to people who have used previous generations of UI HPC systems. One thing that will look a little different is the output of the qhost command. This will show the CPU topology.

```
qhost -h argon-compute-1-01
HOSTNAME    ARCH     NCPU NSOC NCOR NTHR  LOAD  MEMTOT
MEMUSE  SWAPTO  SWAPUS
--------------------------------------------
-------------
global
-     -     -     -     -     -     -     -
-     -     -
argon-compute-1-01  lx-amd64  56  2  28  56  0.03  125.5G
1.1G  2.0G  0.0
```
As you can see that shows the number of cpus (NCPU), the number of CPU sockets (NSOC), the number of cores (NCOR) and the number of threads (NTHR). This information could be important as you plan jobs but it essentially reflects what was said in regard to HT cores. Note that all argon nodes have the same processor topology. SGE uses the concept of job slots which serve as a proxy for the number of cores as well as the amount of memory on a machine. Job slots are one of the resources that is requested when submitting a job to the system. As a general rule, the number of job slots requested should be equal to or greater than the number of processes/threads that will actually consume resources. The parallel environment to request an entire node on Argon is called 56cpn. For one node you would request

```
qsub -pe 56cpn 56
```

More nodes would be requested by specifying a slot count that is a multiple of 56. So for 2 nodes

```
qsub -pe 56cpn 112
```

and so on.

You will need to be aware of the approximate amount of memory per job slot when setting up jobs if your job uses a significant amount of memory. The actual amount will vary due to OS overhead, and will be slightly lower than the values given below.

Using the Basic Job Submission and Advanced Job Submission pages as a reference, how would one submit jobs taking HT into account? For single process high throughput type jobs it probably does not matter, just request one slot per job. For multithreaded or MPI jobs, request one job slot per thread or process. So if your application runs best with 4 threads then request something like the following.

```
qsub -pe smp 4
```

That will run on two physical cores and two HT cores. For non-threaded processes that are also CPU bound you can avoid running on HT cores by requesting 2x the number of slots as cores that will be used. So, if your process is a non-threaded MPI process, and you want to run 4 MPI ranks, your job submission would be something like the following.

```
qsub -pe smp 8
```

and your job script would contain an mpirun command similar to

```
mpirun -np 4 ...
```

That would run the 4 MPI ranks on physical cores and not HT cores. Note that this will work for non-MPI jobs as well. If you have a non-threaded process that you want to ensure runs on an actual core, you could use the same 2x slot request.

```
qsub -pe smp 2
```

Note that if you do not use the above strategy then it is possible that your job process will share cores with other job processes. That may be okay, and preferred for high throughput jobs, but is something to keep in mind. It is especially important to keep this in mind when using the orte parallel environment. There is more discussion on the orte parallel environment on the Advanced Job Submission page. In short, that parallel environment is used in node sharing scenarios, which implies potential core sharing as well. For MPI jobs, that is probably not what you want. As on previous systems, there is a parallel environment (56cpn) for requesting entire nodes. This is especially useful for MPI jobs to ensure the best performance.

For MPI jobs, the system provided openmpi will not bind processes to cores by default, as would be the normal default for openmpi. This is set this way to avoid inadvertently oversubscribing processes on cores. In addition, the system openmpi settings will map processes by socket. This should give a good process distribution in all cases. However, if you wish to use less than 28 processes per node in an MPI job then you may want to map by node to get the most even distribution of processes across nodes. You can do that with the --map-by node option flag to mpirun.
mpirun --map-by node ...

If you wish to control mapping and binding in a more fine-grained manner, the mapping and binding parameters can be overridden with parameters to mpirun. Openmpi provides many options for fine grained control of process layout. The options that are set by default should be good in most cases but can be overridden with the openmpi options for

- mapping controls how processes are distributed across processing units
- binding binds processes to processing units
- ranking assigns MPI rank values to processes

See the mpirun manual page,

man mpirun

for more detailed information. The defaults should be fine for most cases but if you override them keep the topology in mind.

- each node has 2 processor sockets
- each processor socket has 14 processor cores
- each processor core has 2 hardware threads (HT)

If you set your own binding, for instance --bind-to core, be aware that the number of cores is half of the number of total HT processors. Note that core binding in and of itself may not really boost performance much. Generally speaking, if you want to minimize contention with hardware threads then simply request twice the number of slots than cores your job will use. Even if the processes are not bound to cores, the OS scheduler will do a good job of minimizing contention.

If your job does not use the system openmpi, or does not use MPI, then any desired core binding will need to be set up with whatever mechanism the software uses. Otherwise, there will be no core binding. Again, that may not be a major issue. If your job does not work well with HT then run on a number of cores equal to half of the number of slots requested and the OS scheduler will minimize contention.

new SGE utilities

While SoGE is very similar to previous versions of SGE there are some new utilities that people may find of interest. There are manual pages for each of these.

- qstatus: Reformats output of qstat and can calculate job statistics.
- dead-nodes: This will tell you what nodes are not physically participating in the cluster.
- idle-nodes: This will tell you what nodes do not have any activity on them.
- busy-nodes: This will tell you what nodes are running jobs.
- nodes-in-job: This is probably the most useful. Given a job ID it will list the nodes that are in use for that particular job.

SSH to compute nodes

On previous UI HPC systems it was possible to briefly ssh to any compute node, before getting booted from that node if a registered job was not found. This was sufficient to run an ssh command, for instance, on any node. This is not the case for Argon. SSH connections to compute nodes will only be allowed if you have a registered job on that host. Of course, qlogin sessions will allow you to login to a node directly as well. Again, if you have a job running on a node you can ssh to that node in order to check status, etc. You can find the nodes of a job with the nodes-in-job command mentioned above. We ask that you not do more than observe things while logged into the node as it may have shared jobs on it.

Software Packages

While there are many software applications installed from RPM packages, many commonly used packages, and their dependencies, are built from source. See the Argon Software List to view the packages and versions installed. Note that this list does not include all of the dependencies that are installed, which will consist of newer versions than those installed via RPM. Use of these packages is facilitated through the use of environment modules, which will set up the appropriate environment for the application, including loading required dependencies. Some packages like Perl, Ruby, R and Python, are extendable. We build a set of extensions based on commonly used and requested extensions so loading modules for those will load all of the extensions, and dependencies needed for the core package as well as the extensions. The number of extensions installed, particularly for Python and R is too large to list here. You can use the standard tools of those packages to determine what extensions are installed.

Environment Modules

Like previous generation UI HPC systems, Argon uses environment modules for managing the shell environment needed by software packages. Argon uses LMod rather than the TCL modules used in previous generation UI HPC systems. More information about Lmod can be found in the Lmod: A New Environment Module System — Lmod 6.0 documentation. Briefly, Lmod provides improvements over TCL modules in some key
ways. One is that Lmod will automatically load and/or swap dependent environment modules when higher level modules are changed in the environment. It can also temporarily deactivate modules if a suitable alternative is not found, and can reactivate those modules when the environment changes back. We are not using all of the features that Lmod is capable of so the modules behavior should be very close to previous systems but with a more robust way of handling dependencies.

Lmod provides a mechanism to save a set of modules that can then be restored. For those who wish to load modules at shell startup this provides a better mechanism than calling individual module files. The reasons are that

1. Only one command is needed
2. The same command can be used at any time
3. Restoring a module set runs a module purge which will ensure that the environment, at least the part controlled by modules, is predictable.

To use this, simply load the modules that you want to have loaded as a set. Then run the following command.

```
module save
```

That will save the loaded modules as the default set. To restore that run

```
module restore
```

That command could then be put in your shell initialization file. In addition to saving/restoring a default set you can also assign a name to the collection.

```
module save mymodules
module restore mymodules
```

There is also a technical reason to use the module save/restore feature as opposed to individual modules that involves how the LD_LIBRARY_PATH environment variable is handled at shell initialization.

One of the things that environment modules sets up is the \$LD_LIBRARY_PATH. However, when a setuid/setgid program runs it unsets \$LD_LIBRARY_PATH for security reasons. One such setgid program is the duo login program that runs as part of an ssh session. This will leave you with a partially broken environment as a module is loaded, sets \$LD_LIBRARY_PATH but then has it get unset before shell initialization is complete. This is worked around on previous systems by always forcing a reload of the environment module but this is not very efficient. This scenario should not be an issue on Argon as all software is built with RPATH support, meaning the library paths are embedded in the binaries. In theory, \$LD_LIBRARY_PATH would not be needed but this is something to keep in mind if you are loading modules from your ~/bashrc or similar.

Other than the above items, and some other additional features, the environment modules controlled by Lmod should behave very similarly to the TCL modules on previous UI HPC systems.

### Setting default shell

Unix attributes are now available in the campus wide Active Directory Service and Argon makes use of those. One of those attributes is the default Unix shell. This can be set via the following tool: Set Login Shell - Conch. Most people will want the shell set to /bin/bash so that would be a good choice if you are not sure. For reference, previous generation UI HPC systems set the shell to /bin/bash for everyone, unless requested otherwise. We recommend that you check your shell setting via the Set Login Shell - Conch tool and set it as desired before logging in the first time. Note that changes to the shell setting may take up to 24 hours to become effective on Argon.

### Queues and Policies

A significant portion of the HPC cluster systems at UI were funded centrally. These nodes are put into queues named UI or prefixed with UI-.
- UI Default queue
- UI-HM High memory nodes; request only for jobs that need more memory than can be met with the standard nodes.
- UI-MPI MPI jobs; request only for jobs that can take advantage of multiple nodes.
- UI-GPU Contains nodes with GPU accelerators; request only if job can use a GPU accelerator.
- UI-DEVELOP Meant for small, short running job prototypes and debugging.

These queues are available to everyone who has an account on an HPC system. Since that is a fairly large user base there are limits placed on these shared queues. Also note that there is a limit of 10000 active (running and pending) jobs per user on the system.

<table>
<thead>
<tr>
<th>Centrally funded queues</th>
<th>Node Description</th>
<th>Wall clock limit</th>
<th>Running jobs per user</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>(20) mid memory</td>
<td>None</td>
<td>2</td>
</tr>
<tr>
<td>UI-HM</td>
<td>(5) high memory</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>UI-MPI</td>
<td>(56 slot minimum)</td>
<td>48 hours</td>
<td>1</td>
</tr>
<tr>
<td>UI-GPU</td>
<td>(4) mid memory with P100 accelerator</td>
<td>None</td>
<td>1</td>
</tr>
<tr>
<td>UI-DEVELOP</td>
<td>(1) mid memory</td>
<td>24 hours</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that the number of slots available in the UI queue can vary depending on whether anyone has purchased a reservation of nodes. The UI queue is the default queue and will be used if no queue is specified. This queue is available to everyone who has an account on a UI HPC cluster system.

Please use the UI-DEVELOP queue for testing new jobs at a smaller scale before committing many nodes to your job.

In addition to the above, the HPC systems have some nodes that are not part of any investor queue. These are in the all.q queue and are used for node rentals and future purchases. The number of nodes for this purpose varies.

Resource requests

There are many resources that SGE keeps track of and most of them can be used in job submissions. However, the resource designations for machines based on memory and GPU are more likely to be used in practice. For the most part, machines with different amounts of memory and GPU capability are segregated by queues. However, the all.q queue contains all machines and when running jobs in that queue it may be desirable to request specific machine types. The following table lists these out. They would be selected with the `--l resource` flag to qsub. These are all Booleans.

<table>
<thead>
<tr>
<th>Full Resource Name</th>
<th>Shortcut Resource Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>std_mem</td>
<td>sm</td>
</tr>
<tr>
<td>mid_mem</td>
<td>mm</td>
</tr>
<tr>
<td>high_mem</td>
<td>hm</td>
</tr>
<tr>
<td>gpu</td>
<td>gpu</td>
</tr>
<tr>
<td>gpu_k80</td>
<td>k80</td>
</tr>
<tr>
<td>gpu_p100</td>
<td>p100</td>
</tr>
</tbody>
</table>

For example, if you run a job in the all.q queue and want to use a node with a GPU, but do not care which type,

qsub -l gpu=true

If you specifically wanted to use a node with a P100 GPU,

qsub -l gpu_p100=true

or use the shortcut,

qsub -l p100=true