Argon Cluster

General Description

The University of Iowa's Argon HPC system was deployed in February, 2017. It consists of 612 compute nodes running CentOS-7.4 Linux. There are several compute node configurations,

1. 24-core 512GB
2. 32-core 64GB
3. 32-core 256GB
4. 32-core 512GB
5. 40-core 96GB
6. 40-core 192GB
7. 56-core 128GB
8. 56-core 256GB
9. 56-core 512GB
10. 64-core 192GB
11. 64-core 384GB
12. 64-core 768GB
13. 80-core 96GB
14. 80-core 192GB
15. 80-core 384GB
16. 80-core 768GB
17. 80-core 1.5TB

The Argon cluster is split between two data centers,

- ITF Information Technology Facility
- LC Lindquist Center

Most of the nodes in the LC datacenter are connected with the OmniPath high speed interconnect fabric, while most of those in the ITF data center are connected with the InfiniPath fabric, with the latest nodes having a Mellanox Infiniband EDR fabric. There are imany machines with varying types of GPU accelerators:

1. 21 machines with Nvidia P100 accelerators
2. 2 machines with Nvidia K80 accelerators
3. 11 machines with NVidia K20 accelerators
4. 2 machines with Nvidia P40 accelerators
5. 17 machines with 1080Ti accelerators
6. 19 machines with Titan V accelerators
7. 4 machines with V100 accelerators
8. 28 machines with 2080Ti accelerators

The Titan V is now considered as a supported configuration in Argon phase 1 GPU-capable compute nodes but is restricted to a single card per node. Staff have completed the qualification process for the 1080 Ti and concluded that it is **not** a viable solution to add to phase 1 Argon compute nodes.

Heterogeneity

While previous HPC cluster systems at UI have been very homogenous, the Argon HPC system has a heterogeneous mix of compute node types. In addition to the variability in the GPU accelerator types listed above, there are also differences in CPU architecture. We generally follow Intel marketing names, with the most important distinction being the AVX (Advanced Vector Extensions) unit on the processor. The following table lists the processors in increasing generational order.

Architecture	AVX level	Floating Point Operations per cycle
Sandybridge Ivybridge	AVX	8
Haswell Broadwell	AVX2	16
Skylake Silver	AVX512	16 (1) AVX unit per processor core

Skylake Gold	AVX512	32 (2) AVX units per processor core
Cascade Lake Gold	AVX512	32

Note that code must be optimized during compilation to take advantage of AVX instructions. The CPU architecture is important to keep in mind both in terms of potential performance and compatibility. For instance, code optimized for AVX2 instructions will not run on the Sandybridge/Ivybridge architecture because it only supports AVX, not AVX2. However, each successive generation is backward compatible so code optimized with AVX instructions will run on Haswell/Broadwell systems.

Hyper Threaded Cores (HT)

One important difference between Argon and previous systems is that Argon has Hyper Threaded processor cores turned on. Hyper Threaded 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 Hyper Threaded cores can potentially run the application more efficiently. For non-threaded applications you can think of any pair of Hyper Threaded cores to be roughly equivalent to two cores at half the speed if both cores of the pair are in use. Again, that is an over simplification, but the main point is that CPU bound processes perform better when not sharing a CPU core. Hyper Threaded cores can help ensure that the physical processor is kept busy for processes that do not always use the full capacity of a core. The reason for enabling HT for Argon is 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. If your job script is written in `bash` syntax then you can use the `$NSLOTS` SGE variable as follows, using mpirun as an example:

   ```
   mpirun -np $(($NSLOTS/2)) ...
   ```

After the merger of Argon and Neon, there are a few of the older nodes that are not HT capable. These are the High Memory nodes with cpu_arch=sandybridge/ivybridge.

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 write up 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.

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 but the values below can be used for planning purposes.

Node memory (GB)	Job slots	Memory (GB) per slot
64	32	2
96	40	2
96	80	1
128	56	2
192	40	4
192	64	3
192	80	2

256	32	8
256	56	4
384	64	6
384	80	4
512	24 (no HT)	20
512	32 (no HT)	16
512	56	8
768	64	12
768	80	9
1536	80	18

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 (Xcpn, where X is the number of cores per node) for requesting entire nodes. This is especially useful for MPI jobs to ensure the best performance. Note that there are some additional parallel environments that are akin to the smp and Xcpn ones but are specialized for certain software packages. These are

- gaussian-sm and gaussian_lindaX, where X is the number of cores per node
- turbomole_mpiX, where X is the number of cores per node
- wien2k-sm and wien2k_mpiX, where X is the number of cores per node

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 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.