Argon Cluster

General Description

The University of Iowa's Argon HPC system was deployed in February, 2017. There are several compute node configurations:

1. 40-core 96GB
2. 40-core 192GB
3. 56-core 128GB
4. 56-core 256GB
5. 56-core 512GB
6. 64-core 192GB
7. 64-core 384GB
8. 64-core 768GB
9. 80-core 96GB
10. 80-core 192GB
11. 80-core 384GB
12. 80-core 768GB
13. 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 many machines with varying types of GPU accelerators:

1. 21 machines with Nvidia P100 accelerators
2. 2 machines with Nvidia K80 accelerators
3. 2 machines with Nvidia P40 accelerators
4. 17 machines with 1080Ti accelerators
5. 19 machines with Titan V accelerators
6. 14 machines with V100 accelerators
7. 38 machines with 2080Ti accelerators
8. 1 machine with RTX8000 accelerators

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:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>AVX level</th>
<th>Floating Point Operations per cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haswell Broadwell</td>
<td>AVX2</td>
<td>16</td>
</tr>
<tr>
<td>Skylake Silver</td>
<td>AVX512</td>
<td>16 (1) AVX unit per processor core</td>
</tr>
<tr>
<td>Skylake Gold</td>
<td>AVX512</td>
<td>32 (2) AVX units per processor core</td>
</tr>
<tr>
<td>Cascadelake Gold</td>
<td>AVX512</td>
<td>32</td>
</tr>
</tbody>
</table>

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 AVX512 instructions will not run on the Haswell/Broadwell architecture because it only supports AVX2, not AVX512. However, each successive generation is backward compatible so code optimized with AVX2 instructions will run on Skylake/Cascadelake 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 things 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 SNSLOTS SGE variable as follows, using mpirun as an example:

```
mpirun -np \$((SNSLOTS/2)) ...
```

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](http://www.sched.ac.uk/). The version of SGE that Argon uses is from the [Son of Grid Engine](http://www.sched.ac.uk/son_of_grid_engine.php) 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.

<table>
<thead>
<tr>
<th>Node memory (GB)</th>
<th>Job slots</th>
<th>Memory (GB) per slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>96</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>56</td>
<td>2</td>
</tr>
<tr>
<td>192</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>192</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>192</td>
<td>80</td>
<td>2</td>
</tr>
<tr>
<td>256</td>
<td>56</td>
<td>4</td>
</tr>
<tr>
<td>384</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>384</td>
<td>80</td>
<td>5</td>
</tr>
<tr>
<td>512</td>
<td>56</td>
<td>9</td>
</tr>
<tr>
<td>768</td>
<td>64</td>
<td>12</td>
</tr>
<tr>
<td>768</td>
<td>80</td>
<td>9</td>
</tr>
<tr>
<td>1536</td>
<td>80</td>
<td>19</td>
</tr>
</tbody>
</table>
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.

```bash
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.

```bash
qsub -pe smp 8
```

and your job script would contain an `mpirun` command similar to

```bash
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.

```bash
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 a number of processes less than half of the slots 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`.

```bash
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,

```bash
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.