Advanced Research Computing at Virginia Tech (https://...

HOKIESPEED (CPU/GPU)

HokieSpeed Index

Overview / Technical Specifications

HokieSpeed is a GPU-accelerated supercomputing cluster consisting of 204 compute nodes linked together using a quad data rate (QDR) InfiniBand interconnect. Each HokieSpeed node is outfitted with 24 GB of memory, two six-core Xeon E5645 CPUs and two NVIDIA M2050 / C2050 GPU. As of November 2012, HokieSpeed was ranked No. 221 on the Top500 (http://www.top500.org/)supercomputer list and No. 43 on the Green500 (http://www.green500.org/) list of energy efficient supercomputers.

Policies

Hokiespeed has two queues, each of which has different policies:

QUEUE	NORMAL_Q	DEV_Q
Intended Use	Production jobs	Development and testing
Available Nodes	hs004-hs201	All
Max Jobs/User	16	1
Max Nodes/User	64	2
Max Cores/User	768	24
Max Run Time	144 hours	2 hours
Max Core-Hours/User*	27,648	48

* A user cannot, at any one time, have more than this many core-hours allocated across all of their running jobs. (Core-hours is defined as the number of cores used multiplied by the remaining requested walltime.) For example, the setting means that a user wishing to run a job with the maximum of 768 cores will be limited to a 36 hour walltime (27,648/768 = 36). This also implies that a job using the maximum runtime of 144 hours cannot use more than 192 cores (27,648/144 = 192).

Software and Compilers

In general, users should utilize the `module avail` command to get an up-to-date list of the software available on a given system. However, the lists below should provide some feel for the software available on Hokiespeed. Note that a user will have to load the appropriate module(s) (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=992) in order to use a given software package.

Compilers

The following compilers are available on Hokiespeed. Compilers form the lowest level of Hokiespeed's module hierarchy (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=992#structure).

- Intel 13.1 (the default)
- GCC 4.9.2, 4.5.3, 4.4.7
- CUDA 6.5.14, 5.0.35, 4.1.28
- JDK 1.7, 1.6

MPI Stacks

The following MPI Stacks are available on BlueRidge. Compilers form the second level of BlueRidge's module hierarchy (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=992#structure).

- mvapich2 1.9b, 2.1
- OpenMPI 1.6.4 (the default), 1.8.4
- Intel MPI 4.1

Software

The following specialty software packages are available on HokieSpeed. Since HokieSpeed is a GPU accelerated resource, only those software packages which are able to utilize GPU efficiently will be available on this resource. For a comparison of software available on all ARC systems, click here (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=114).

- Gromacs 4.5.5

- LAMMPS 27Aug12
- NAMD 2.8
- Amber 2015
- OpenFOAM 2.3, 2.2
- R 3.0.3, 2.14.1
- Python 2.7.10
- VMD 1.9.1
- ParaView 4.0.1, 3.14.1
- VisIt 2.4.2

Usage

The cluster is accessed via ssh to one of the login nodes `hokiespeed1.arc.vt.edu` or `hokiespeed2.arc.vt.edu`. Log in using your username (usually Virginia Tech PID) and password. You will need an SSH Client to log in; see here (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=98#sshClients) for information on how to obtain and use an SSH Client. You must be on a campus network to access the login node, so off-campus use requires connecting to the CNS VPN (http://www.google.com/search?q=vpn+site:answers.vt.edu) first.

Jobs are submitted to ARC resources through a job queuing system, or "scheduler". Submission of jobs through a queueing system means that jobs may not run immediately, but will wait until the resources that it requires are available. The queuing system thus keeps the compute servers from being overloaded and allocates dedicated resources across running jobs. This will allow each job to run optimally once it leaves the queue.

Job management (submission, checking) is described in the Scheduler Interaction tutorial. Please take note of HokieSpeed's policies and queues when creating your submission script. A step-by-step example is provided below.

Examples

Submission Templates

This shell script (https://secure.hosting.vt.edu/www.arc.vt.edu/wp-content/uploads/2015/08/hs_example.qsub) provides a template for submission of jobs on HokieSpeed. The comments in the script include notes about how to add modules, submit MPI jobs, etc.

To utilize this script template, create your own copy and edit as described here (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=1003#script).

Step-by-Step Examples

To compile and run the example CUDA vector addition program (http://www.arc.vt.edu/resources/software/cuda/docs/vectorAdd.cu) on Hokiespeed:

1. SSH into Hokiespeed (See Submitting Jobs (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=100#submit), above).
2. Download the source code file (link above) and put it in a folder in your Home directory (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=112). See here (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=464) for information on how to transfer files to and from ARC's systems.
3. Use the Unix `cd` command (http://www.arc.vt.edu/resources/software/unix/directry.php#3) to navigate to that folder.
4. Compile the source code file into an executable:
    1. Load the required modules:
        1. Check which modules are loaded using the `module list` command.
        2. If they are not already loaded, add the GCC and CUDA modules using the `module load` command: `module load gcc cuda`. (Note that if the Intel compiler is loaded in place of GCC, you can replace Intel with GCC using the `module swap` command: `module swap intel gcc`.)
    2. Compile the source code: `nvcc -lcuda -lcudart -o vecadd vectorAdd.cu` (Note that here we use the `nvcc` command since it is an CUDA program.) (Note also that compiling could be done with a makefile by putting this file (http://www.arc.vt.edu/resources/hpc/docs/hs_vecadd_makefile) in the directory, renaming it to simply `makefile`, and typing `make`at the command line.)
    3. Your executable is now in the file `vecadd`. To execute the program, you would use the command: `./vecadd`. To do so, however, would run the job on a head node, thereby slowing down the system for other users. Therefore, Hokiespeed jobs should be submitted to the scheduler using a `qsub`command (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=100#submit)(see the next step).
5. To submit your program to the scheduler, download and open the sample Hokiespeed submission script (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=100#submit).
6. Edit the script to run your program:
    1. The walltime is set with the command `#PBS -l walltime`. This is the time that you expect your job to run; so if you submit your job at 5:00pm on Wednesday and you expect it to finish at 5:00pm on Thursday, the walltime would be 24:00:00. Note that if your job exceeds the walltime estimated during submission, the scheduler will kill it. So it is important to be conservative (i.e., to err on the high side) with the walltime that you include in your submission script. The walltime in the sample script is set to one hour; the quadrature code will run quickly so we'll change the walltime to 10 minutes using the command `#PBS -l walltime=00:10:00`.
    2. Edit the line `#PBS -lnodes=1:ppn=6` to set the number of nodes (lnodes) and processors per node (ppn) that you want to utilize. On Hokiespeed ppnshould generally be set to 6. The number of nodes available in each queue is described in the Policies (https://secure.hosting.vt.edu

/www.arc.vt.edu/?page_id=100#policies) section. For this example, we'll use 12 cores across 2 nodes using the command `#PBS -l nodes=2:ppn=6`.

3. The CUDA vector addition program requires the same modules to run that we needed to compile it (GCC and CUDA). So the `module load` line should be changed to `module load gcc cuda`. For more on Hokiespeed's module structure, click here (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=100#modules).
4. Replace everything between the `echo "Hello world!"` line and (but not including) the `exit` lines in the sample script with the command to run your job: `./vecadd`.

7. Your script should look something like this (http://www.arc.vt.edu/resources/hpc/docs/hs_vecadd_qsub.sh). Save the script file.
8. Copy the compiled file and script to your Work directory (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=112#work) (example command: `cp vecadd $WORK`). Running your program from Work (or Scratch) will ensure that it gets the fastest possible read/write performance.
9. Navigate to your Work directory: `cd $WORK`
10. To submit the script, use the `qsub` command. For example, if you saved your script file as "hs_vecadd_qsub.sh", the command would be `qsub ./hs_vecadd_qsub.sh`.
11. The system will return your job name of the form xxxx.Hokiespeed.arc.vt.edu (e.g., 53318.Hokiespeed.arc.vt.edu, where 53318 is the job number). Follow the instructions above (https://secure.hosting.vt.edu/www.arc.vt.edu/?page_id=100#submit) to use `qstat` to track the progress of your job, `qdel` to delete your job, etc.
12. When complete, the program output will be held in the file with the extension .o followed by your job number (e.g. "hs_vecadd_qsub.sh.o53318"). Any errors will be held in the analogous .e file (e.g. "hs_vecadd_qsub.sh.e53318").
13. Work and Scratch are wiped periodically, so to ensure that you have long-term access to the results, copy them back to your Home directory: `cp hs_vecadd_qsub.sh.o53318 $HOME`

Notes for compiling and running other examples:

- CUDA MPI program (http://www.arc.vt.edu/resources/software/cuda/docs/cuda-mpi.cu):
  1. Requires modules for GCC, CUDA, and an MPI implementation (e.g. OpenMPI): `module load gcc openmpi cuda`. Note that you may need to purge the modules before this (`module purge`); alternatively, if the Intel compiler is loaded in place of GCC, you can replace Intel with GCC using the`module swap` command (`module swap intel gcc`).
  2. To compile use this makefile (https://secure.hosting.vt.edu/www.arc.vt.edu/wp-content/uploads/2015/04/hs_cudampi_makefile.txt) or this command line: `nvcc -arch sm_13 -I$VT_MPI_INC -L$VT_MPI_LIB -lmpi -lcuda -lcudart -o run-cuda-mpi cuda-mpi.cu`. ($VT_MPI_INC and $VT_MPI_LIB are environment variables that point to directories associated with the MPI module loaded.)
  3. In the qsub script, run with the following command: `mpiexec -npernode 1 ./run-cuda-mpi` (This would run with 1 process per node, which is ideal for CUDA code so that you don't have

more than one process trying to access the same GPU.)

- CUDA OpenMP program (http://www.arc.vt.edu/resources/software/cuda/docs/cuda-omp.cu):
  1. Requires modules for GCC and CUDA: `module load gcc cuda`. Note that you may need to purge the modules before this (`module purge`); alternatively, if the Intel compiler is loaded in place of GCC, you can replace Intel with GCC using the `module swap` command (`module swap intel gcc`).
  2. To compile use this makefile (https://secure.hosting.vt.edu/www.arc.vt.edu/wp-content/uploads/2015/04/hs_cudaomp_makefile.txt) or this command line: `nvcc -Xcompiler -fopenmp -lcuda -lcudart -lgomp -o run-cuda-omp cuda-omp.cu`
  3. In the qsub script, run with the following command: `./run-cuda-omp`
- CUDA Matrix Multiplication program (http://www.arc.vt.edu/resources/software/cuda/docs/MatMul.cu):
  1. Requires modules for GCC and CUDA: `module load gcc cuda`. Note that you may need to purge the modules before this (`module purge`); alternatively, if the Intel compiler is loaded in place of GCC, you can replace Intel with GCC using the `module swap` command (`module swap intel gcc`).
  2. To compile use this command line: `nvcc -lcuda -lcudart -o run-cuda-matmul MatMul.cu`
  3. In the qsub script, run with the following command: `./run-cuda-matmul`