SciNet Tutorial

March 19, 2010

Contents

1 Introduction **1**

2 Using SciNet's Systems **2**
2.1 Software modules . . . 2
2.2 The login/develop/compute setup . . . 2
2.3 Compilers . . . 3
2.4 Running jobs . . . 4
2.5 Limits and priorities . . . 5

3 Examples of Computing on the GPC **6**
3.1 OpenMP jobs on GPC . . . 6
3.2 MPI jobs on GPC . . . 7
3.3 Serial jobs on GPC . . . 8

4 Examples of Computing on the TCS **9**
4.1 OpenMP jobs on TCS . . . 9
4.2 MPI jobs on TCS . . . 10

1 Introduction

SciNet is a consortium for High-Performance Computing consisting of researchers at the University of Toronto and its associated hospitals. SciNet has two main clusters, the General Purpose Cluster (GPC) and the Tightly Coupled System (TCS)

- The GPC has 3,780 Nehalem nodes with 8 cpus (2.5GHz) and 16GB RAM per node. One quarter of the GPC is interconnected with Infiniband, the rest with gigabit ethernet.
- The TCS has 104 nodes. The nodes each have 32 Power 6 cores (4.7GHz) and 128GB RAM, and are connected by Infiniband.

The purpose of this document is to get you up and running on SciNet. It will be assumed that you have a SciNet account and a rudimentary knowledge of the UNIX prompt.

- The SciNet user wiki at support.scinet.utoronto.ca/wiki covers much more than this short tutorial. Throughout this tutorial, we will refer to the relevant wiki pages where you can find more information.
- How to get an account is explained on www.scinet.utoronto.ca/support/How_to_get_an_Account.htm .
- If you still have difficulties, contact us at support@scinet.utoronto.ca.
- Access to the TCS is not enabled by default. We ask that people justify the need for this highly specialized machine. Email us explaining the nature of your work if you want access to the TCS. In particular, applications should scale well to 64 processes/threads to run on this system.

2 Using SciNet's Systems

Because of the large variety of users and the size of SciNet's systems, computing on SciNet works slightly different than you may be used to. You should become familiar with these specifics of the SciNet systems:

1. The software module system
2. The login/develop/compute node setup
3. The compilers
4. How to run jobs
5. Job limits and priorities

2.1 Software modules

Much of the software is not accessible by default but has to be loaded using the 'module' command. The reason is that

- it allows us to easily keep multiple versions of software for different users on the system;
- it allows users to easily switch between versions.

The module system works similarly on the GPC and the TCS.

To use particular software just load the module:

```
module load <module-name>
```

For a full list of available software packages that may be accessed with the module system, use the command

```
module avail
```

For a list of the currently loaded modules in your shell, use

```
module list
```

For a description of a particular module, type

```
module help <module-name>
```

- Modules that load libraries define environment variables pointing to the location of library files and include files for use Makefiles. These environment variables follow the naming convention

  ```
  SCINET_[shortpkgname]_{LIB,INC,BASE}
  ```

- It is recommended to load frequently used modules in the file `.bashrc` in your home directory.
- Apart from the compilers, commercial packages are not available on SciNet because of licensing issues.
- Read more on the wiki page Software and Libraries

2.2 The login/develop/compute setup

On SciNet, you cannot just login, compile and start running. Rather, this involves a three-stage process:

1) Login: Access to the SciNet systems

Access to the SciNet systems is via `ssh` only. Ssh to `login.scinet` to use the GPC or TCS:

```
ssh -l <user-name> login.scinet.utoronto.ca
```

From here you can view your directories, see the GPC queue using `showq`, and log into development nodes.

- Users can transfer *small* files into or out of the data centre via the login nodes, using `scp`, or `rsync` over ssh. Large data transfers, however, should be done via the `datamover1` node. This node can initiate both

incoming and outgoing transfers, and since it is on a 10 Gbps link to the University of Toronto, it is the fastest - and recommended - way to transfer data. `datamover1` is not accessible from the outside, so you must login to `login.scinet.utoronto.ca` and then ssh to the data mover node.
- The SciNet firewall monitors for too many attempted connections, and will shut down all access (including previously working connections) from your IP address if more than four connection attempts (successful or not) are made within the space of a few minutes. In that case, you will be locked out of the system for an hour. Be patient in attempting new logins!
- Read more on the wiki pages Data Transfer , Storage Quickstart , Essentials

2) Develop: compilation and testing

The login machines are not the same architecture as either the GPC or TCS nodes, so you should not compile programs on the login machines but on the development nodes.

From `login.scinet.utoronto.ca`, you log into one of four GPC development nodes, `gpc01..gpc04`, e.g.

```
ssh gpc04
```

or either of the TCS development nodes `tcs01` or `tcs02`, e.g.

```
ssh tcs02
```

These nodes are to be used for compiling and test runs (on the TCS, use only tcs02 for test runs). However, because these machines are used by *everyone* who needs to use the SciNet systems, be considerate; only run scripts or programs that use a moderate amount of memory, only a few of the cores and do not take more than a few minutes.

- While this document focuses on getting you started, we encourage you to make the most of SciNet's resources. For available tools to analyze and improve your code's performance, see Introduction To Performance , Performance And Debugging Tools: GPC , and Performance And Debugging Tools: TCS

3) Run: use the queuing system

After copying your executable and any input files to `/scratch/<user-name>`, you can submit jobs to the queue from the development nodes using a script that specifies what executable to run, from which directory to run it, on how many nodes, with how many threads, and for how long. The queuing system used at SciNet is based around the Cluster Resources Moab Workload Manager , with Torque as the back-end resource manager on the GPC and IBM's LoadLeveler on the TCS. The queuing system will send the jobs to the compute nodes. Except in very rare cases, you do not need to access these compute nodes yourself.

- Queuing commands are explained in the 'Running jobs' section below.
- Your home directory is read-only from the compute nodes, so you have to run your job from the scratch.
- *The scratch directory is not backed up, so you need to copy essential results to you home directory!*
- Read more on the wiki pages Moab , GPC Quickstart , TCS Quickstart , Storage Quickstart .

2.3 Compilers

GPC compilers

It is recommended that you compile with the Intel® compilers, which are `icc/icpc/ifort` for C/C++/Fortran, and are available with the default module `intel`. The Intel® compilers are recommended over the GNU compilers. To ensure that these compilers are in your PATH and their libraries are in your `LD_LIBRARY_PATH`, use the command

```
module load intel
```

If you really need the GNU compilers, the latest version of the GNU compiler collection is available by loading the `gcc` module, with `gcc/g++/gfortran` for C/C++/Fortran (g77 is not supported).

- We suggest the use of at least the following compiler flags
  `-O3 -xHost`
  to optimize your code for the GPC machine (`-O3 -march=native` for GNU compilers).
- Add `-openmp` to both compile and link line for code that uses openmp (`-fopenmp` for GNU compilers).
- If you get the warning '`feupdatreenv is not implemented`', add `-limf` to the link line.
- When using the Intel® Math Kernel Library, software.intel.com/en-us/articles/intel-mkl-link-line-advisor can tell you what to append to the link command.
- See software.intel.com/en-us/articles/intel-software-technical-documentation for further documentation on the Intel® compilers.

TCS compilers

You should use the IBM compilers, which are `xlc/xlC/xlf` for C/C++/Fortran. For OpenMP or other threaded applications, use the 'reentrant-safe' versions of them, `xlc_r`, `xlC_r`, and `xlf_r`. For MPI applications, the scripts `mpcc/mpCC/mpxlf` are wrappers for the compilers which include the appropriate flags to use the IBM MPI libraries. Hybrid applications will need to use `mpcc_r/mpCC_r/mpxlf_r`.

- We strongly suggest the compiler flags
  `-q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6`
  supplemented by
  `-qsmp=omp`
  for OpenMP programs.
- On the link line we suggest using
  `-q64 -bdatapsize:64k -bstackpsize:64k`
  also supplemented by
  `-qsmp=omp`
  for OpenMP programs.
- For c++ code that uses the full C++ bindings in MPI (those in the MPI namespace), add
  `-cpp`
  to the compilation command.

2.4 Running jobs

GPC test runs

You can run short test runs on the development nodes of GPC as long as they only take a few minutes, a moderate amount of memory, and do not use all 8 cores.

To run a short serial test run on the GPC, simply type on a development node

```
./<executable-name> [arguments]
```

To run a short 4-thread OpenMP run on the GPC, type

```
OMP_NUM_THREADS=4 ./<executable-name> [arguments]
```

To run a short 4-process MPI run on GPC (single node test), type

```
mpirun -np 4 ./<executable-name> [arguments]
```

- A `debug` queue is available for longer, multinode test runs. See the wiki page section Moab .

TCS test runs

You can run short test runs on development node `tcs02` of the TCS as long as they only take a few minutes, a moderate amount of memory, and do not use all 32 cores.

To run a short 16-thread OpenMP test run on tcs02:

```
OMP_NUM_THREADS=16 ./<executable-name> [arguments]
```

To run a short 16-process MPI test run on tcs02:

```
mpiexec -n 16 ./<executable-name> [arguments] -hostfile <hostfile>
```

- `<hostfile>` should contain as many of the line `tcs-f11n06` as you want processes in the MPI run.
- Furthermore, the file `.rhosts` in your home directory has to contain a line with `tcs-f11n06`.

Production runs

To run a job on the compute nodes you have to submit it to a queue. You need to write a small script which details the requirement of the job and the commands to be executed. Below we will give examples of such scripts that you can use as a starting point.

Note that you have to run from the scratch directory, because your home directory is read-only on the compute nodes. Since the scratch directory is not backed up, you need to copy essential results to your home directory after.

For the GPC, you submit the script to the queue using

```
qsub <script-name>
```

where you will replace `<script-name>` with the file containing the submission script.

- It can take a minute for the job to be incorporated in the queue, after which you can use:
- `showq` to show the queue,
- and job-specific commands such as `showstart, checkjob, canceljob`
- Read more on the wiki pages GPC Quickstart , Moab

For the TCS, submitting is done with

```
llsubmit <script-name>
```

and `llq` shows the queue.

- The Power6 series of processors has a facility called Simultaneous Multi Threading which allows two tasks to be very efficiently bound to each core. Using this requires no changes to the code, only running 64 rather than 32 tasks on the node. For OpenMP application, see if setting `OMP_NUM_THREADS` and `THRDS_PER_TASK > 32` makes your job run faster. For MPI, increase `tasks_per_node > 32`.
- It can take a minute for the job to be incorporated in the queue.
- One your job is in the queue, you can use `llq` to show the queue, and job-specific commands such as `llcancel, llhold, ...`
- *Do not run serial jobs on the TCS!* The GPC can do that, of course, in bunches of 8.
- Read more on the wiki pages TCS Quickstart , Moab

2.5 Limits and priorities

If your group has a default account, up to 16 jobs are allowed in the queue, running on a total of 32 nodes at a time for 48 hours per job on the GPC cluster; and for those who have also applied to use the more specialized TCS resource, up to 8 jobs in the queue running on a total of 2 nodes at a time, again with a 48 hour wall-clock limit per job.

Users who need more than the default amount of resources must apply for it through the account allocation/LRAC/NRAC process. While their resources last, their jobs will run at a higher priority than others.

- Because of the group based allocation, it is conceivable that your jobs won't run if your colleagues have already exhausted your group's limits.
- See also Essentials#Usage Policy on the SciNet wiki page.
- Users with an NRAC/LRAC allocation, see Accounting on the Moab page about group/RAP priorities.

How to making your jobs start sooner

- Reduce the requested time (`walltime/wall_clock_limit`) to be close to the estimated required time (perhaps adding about 10 percent to be sure). Shorter jobs are scheduled sooner than longer ones.
- Do not request Infiniband nodes. Because there are a limited number of these nodes, your job will start running faster if you do not request them.

3 Examples of Computing on the GPC

3.1 OpenMP jobs on GPC

Load required modules:

```
module load intel
```

Compiling

For Fortran/C/C++:

```
ifort -openmp -O3 -xHost example.f   -c -o example.o
icc   -openmp -O3 -xHost example.c   -c -o example.o
icpc  -openmp -O3 -xHost example.cpp -c -o example.o
```

Linking

For Fortran/C/C++:

```
ifort -openmp example.o -o example
icc   -openmp example.o -o example
icpc  -openmp example.o -o example
```

Submitting

Copy the executable to the scratch disk (`/scratch/<user-name>`) and create a simple script, as follows

```
#!/bin/bash
#MOAB/Torque submission script for SciNet GPC (OpenMP)
#PBS -l nodes=1:ppn=8,walltime=1:00:00
#PBS -N test
#DIRECTORY TO RUN - $PBS_O_WORKDIR is directory job was submitted from
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=8
./example
```

Submit the job with

```
qsub <script-name>
```

where you will replace `<script-name>` with the file containing the submission script. This will return a job ID. Information about a queued job can be found using `checkjob JOB-ID`, and jobs can be canceled with the command `canceljob JOB-ID`.

- Read more on the wiki pages Moab and GPC Quickstart .

3.2 MPI jobs on GPC

Load required modules:

```
module load intel openmpi
```

- Read more on the wiki page GPC MPI Versions
- Warning: different MPI version require different arguments to mpirun!

Compiling

For Fortran 77/Fortran 90/C/C++:

```
mpif77 -O3 -xHost example.f   -c -o example.o
mpif90 -O3 -xHost example.f   -c -o example.o
mpicc  -O3 -xHost example.c   -c -o example.o
mpicxx -O3 -xHost example.cpp -c -o example.o
```

Linking

For Fortran 77/Fortran 90/C/C++:

```
mpif77 -limf example.o -o example
mpif90 -limf example.o -o example
mpicc  -limf example.o -o example
mpicxx -limf example.o -o example
```

Submitting

Copy the executable to the scratch disk (`/scratch/<user-name>`) and create a simple script, for example,

```
#!/bin/bash
#MOAB/Torque submission script for SciNet GPC (ethernet)
#PBS -l nodes=2:ppn=8,walltime=1:00:00
#PBS -N test
#
#DIRECTORY TO RUN - $PBS_O_WORKDIR is directory job was submitted from
cd $PBS_O_WORKDIR
#EXECUTION COMMAND; -np = nodes*ppn
mpirun -np 16 -hostfile $PBS_NODEFILE ./example
```

Submit the job with

```
qsub <script-name>
```

where you will replace `<script-name>` with the file containing the submission script.

- The different versions of MPI require different commands to launch the run, and thus different scripts. The above script is specific for the openmpi module. For the intelmpi module, the last line should read
  ```
  mpirun -r ssh -np 16 -env I_MPI_DEVICE ssm ./example
  ```

Ethernet vs. Infiniband

About 1/4 of the GPC is connected with a high bandwidth low-latency fabric called InfiniBand. Many jobs which require tight coupling to scale well greatly benefit from this interconnect; other types of jobs, which have relatively modest communications, do not require this and run fine on Gigabit ethernet. Jobs which require the InfiniBand for good performance can request the nodes that have the 'ib' feature in the line

```
#PBS -l nodes=2:ib:ppn=8,walltime=1:00:00
```

Your job will start running faster if you do not request them. InfiniBand nodes are to be used only for jobs that are known to scale well and will benefit from this type of interconnect. The MPI libraries automatically correctly use either the InfiniBand or ethernet interconnect depending on which nodes your job runs on.

- *Requesting Infiniband has no effect on OpenMP or serial jobs, except a longer wait in the queue!*
- Read more on the wiki page GPC Quickstart

3.3 Serial jobs on GPC

SciNet is a parallel computing resource, and our priority will always be parallel jobs. Having said that, if you can make efficient use of the resources using serial jobs and get good science done, that's good too. The GPC nodes each have 8 processing cores, and making efficient use of these nodes means using all eight cores. As a result, we'd like to have the users take up whole nodes (e.g., run multiples of 8 jobs) at a time. The easiest way to do this is to bunch the jobs in groups of 8 that will take roughly the same amount of time.

As before, start by loading required modules if this is not done in your `.bashrc`:

```
module load intel
```

Compiling

For Fortran/C/C++:

```
ifort -O3 -xHost dojobX.f   -c -o dojobX.o
icc   -O3 -xHost dojobX.c   -c -o dojobX.o
icpc  -O3 -xHost dojobX.cpp -c -o dojobX.o
```

Linking

For Fortran/C/C++:

```
ifort dojobX.o -o dojobX
icc   dojobX.o -o dojobX
icpc  dojobX.o -o dojobX
```

Submitting

Copy the executable to the scratch disk (`/scratch/<user-name>`) and create a script in the same directory which bunches 8 serial jobs together. You could do this by creating 8 sub-directories, copy the executable to each one. An example is given here:

```
#!/bin/bash
#MOAB/Torque submission script for multiple serial jobs on SciNet GPC
#PBS -l nodes=1:ppn=8,walltime=1:00:00
#PBS -N serialx8
#
#DIRECTORY TO RUN - $PBS_O_WORKDIR is directory job was submitted from
cd $PBS_O_WORKDIR
#EXECUTION COMMAND; ampersand off 8 jobs and wait
(cd jobdir1; ./dojob1) &
(cd jobdir2; ./dojob2) &
(cd jobdir3; ./dojob3) &
(cd jobdir4; ./dojob4) &
(cd jobdir5; ./dojob5) &
(cd jobdir6; ./dojob6) &
(cd jobdir7; ./dojob7) &
(cd jobdir8; ./dojob8) &
wait
```

- *The wait command at the end is crucial; without it the job will terminate immediately, killing the 8 programs you just started!*
- It is important to group the programs by how long they will take. If (say) dojob8 takes 2 hours and the rest only take 1, then for one hour 7 of the 8 cores on the GPC node are wasted; they are sitting idle but are unavailable for other users, and the utilization of this node is only 56 percent.
- You should have a good idea of how much memory the jobs require. The GPC compute nodes have about 14.5GB in total available to user jobs running on the 8 cores (less, roughly 13GB, on gpc01..04). So the jobs also have to be bunched in ways that will fit into 14.5GB. If that's not possible, one could in principle to just run fewer jobs so that they do fit; but then, the under-utilization problem remains.

Submit the job with

```
qsub <script-name>
```

where you will replace `<script-name>` with the file containing the submission script.

- Read more on the wiki page GPC Quickstart

4 Examples of Computing on the TCS

4.1 OpenMP jobs on TCS

Compiling

For Fortran/C/C++:

```
xlf_r -qsmp=omp -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.f   -c -o example.o
xlc_r -qsmp=omp -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.c   -c -o example.o
xlC_r -qsmp=omp -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.cpp -c -o example.o
```

Linking

```
xlf_r -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
xlc_r -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
xlC_r -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
```

Submitting

Copy the executable to the scratch and create a script along the following lines

```
#Specifies the name of the shell to use for the job
#@ shell = /usr/bin/ksh
#@ job_name = <some-descriptive-name>
#@ job_type = parallel
#@ class    = verylong
#@ environment = copy_all; memory_affinity=mcm; mp_sync_qp=yes; \
#           mp_rfifo_size=16777216; mp_shm_attach_thresh=500000; \
#           mp_euidevelop=min; mp_use_bulk_xfer=yes; \
#           mp_rdma_mtu=4k; mp_bulk_min_msg_size=64k; mp_rc_max_qp=8192; \
#           psalloc=early; nodisclaim=true
#@ node = 1
#@ tasks_per_node = 1
#@ node_usage = not_shared
#@ output = $(jobid).out
#@ error = $(jobid).err
#@ wall_clock_limit = 04:00:00
#@ queue
export target_cpu_range=-1
cd /scratch/<user-name>/<some-directory>
## To allocate as close to the cpu running the task as possible:
export MEMORY_AFFINITY=MCM
## next variable is for OpenMP
export OMP_NUM_THREADS=32
## next variable is for ccsm_launch
export THRDS_PER_TASK=32
## ccsm_launch is a "hybrid program launcher" for MPI-OpenMP programs
poe ccsm_launch ./example
```

Submit the job with

```
llsubmit <script-name>
```

where you will replace `<script-name>` with the file containing the submission script.

4.2 MPI jobs on TCS

Compiling

For Fortran/C/C++:

```
mpxlf     -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.f   -c -o example.o
mpcc      -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.c   -c -o example.o
mpCC -cpp -q64 -O3 -qhot -qarch=pwr6 -qtune=pwr6 example.cpp -c -o example.o
```

Linking

```
mpxlf -q64 -O3 -bdatapsize:64k -bstackpsize:64k example.o -o example
mpcc  -q64 -O3 -bdatapsize:64k -bstackpsize:64k example.o -o example
mpCC  -q64 -O3 -bdatapsize:64k -bstackpsize:64k example.o -o example
```

Submitting

Copy the executable to the scratch and create a script along the follows lines

```
#LoadLeveler submission script for SciNet TCS: MPI job
#@ job_name         = <some-descriptive-name>
#@ initialdir       = /scratch/<user-name>/<some-directory>
#@ executable       = example
#@ arguments        =
#@ tasks_per_node   = 64
#@ node             = 2
#@ wall_clock_limit= 12:00:00
#@ output           = $(job_name).$(jobid).out
#@ error            = $(job_name).$(jobid).err
#@ notification     = complete
#@ notify_user      = <user@example.com>
#Don't change anything below here unless you know exactly
#why you are changing it.
#@ job_type         = parallel
#@ class            = verylong
#@ node_usage       = not_shared
#@ rset = rset_mcm_affinity
#@ mcm_affinity_options = mcm_distribute mcm_mem_req mcm_sni_none
#@ cpus_per_core=2
#@ task_affinity=cpu(1)
#@ environment = COPY_ALL; MEMORY_AFFINITY=MCM; MP_SYNC_QP=YES; \
#            MP_RFIFO_SIZE=16777216; MP_SHM_ATTACH_THRESH=500000; \
#            MP_EUIDEVELOP=min; MP_USE_BULK_XFER=yes; \
#            MP_RDMA_MTU=4K; MP_BULK_MIN_MSG_SIZE=64k; MP_RC_MAX_QP=8192; \
#            PSALLOC=early; NODISCLAIM=true
# Submit the job
#@ queue
```

Submit the job with

```
llsubmit <script-name>
```

where you will replace `<script-name>` with the file containing the submission script.

- Read more on the wiki page TCS Quickstart