User Tutorial

SciNet HPC Consortium
Compute/Calcul Canada

April 21, 2011

DON'T PANIC

1 Introduction **2**
1.1 General Purpose Cluster (GPC) . . . 2
1.2 Tightly Coupled System (TCS) . . . 2
1.3 Disk space . . . 3

2 Usage **3**
2.1 Login . . . 3
2.2 Software modules . . . 3
2.3 Compiling . . . 4
2.4 Testing/debugging . . . 6
2.5 Running your jobs . . . 7
2.6 Data Management . . . 8
2.7 Acknowledging SciNet . . . 10

3 GPC examples **10**
3.1 Shared memory parallel jobs (OpenMP) . . . 11
3.2 Distributed memory parallel jobs (MPI) . . . 11
3.3 Serial jobs . . . 12
3.4 Hybrid MPI/OpenMP jobs . . . 13

4 TCS examples **14**
4.1 Shared memory parallel jobs (OpenMP) . . . 14
4.2 Distributed memory parallel jobs (MPI) . . . 15
4.3 Hybrid MPI/OpenMP jobs . . . 16

1 Introduction

SciNet is a consortium for High-Performance Computing made up of researchers at the University of Toronto and its associated hospitals. It is part of Compute/Calcul Canada, as one of seven consortia in Canada providing HPC resources to their own academic researchers, other users in Canada and international collaborators.

Any qualified researcher at a Canadian university can get a SciNet account through this two-step process:

- Register for a Compute Canada Database (CCDB) account at ccdb.computecanada.org/
- Non-faculty need a sponsor (supervisors CCRI number), who has to have a SciNet account already.
- Login and apply for a SciNet account (click *Apply* beside SciNet on the *Consortium Accounts* page)

Usage reports are available on the SciNet portal portal.scinet.utoronto.ca .

Users who need more than the default amount of resources must have their PI apply for it through the competitively awarded account allocation process once a year, in the fall. Without such an allocation, a user may still use up to 32 nodes (256 cores) of the General Purpose Cluster at a time at low priority.

The SciNet wiki, support.scinet.utoronto.ca/wiki , contains a wealth of information about using SciNet's systems and announcements for users, and also shows the current system status. Other forms of support:

- Users are kept up-to-date with development and changes on the SciNet systems through a monthly email.
- Monthly SciNet User Group meetings including one or more TechTalks.
- Classes, such as the Intro to SciNet, 1-day courses on parallel I/O, (parallel) programming, etc., and a 5-day intensive parallel programming course. The courses web site is support.scinet.utoronto.ca/courses
- Past lecture slide and some videos can be found on the Tutorials and Manuals page.
- The SciNet team can help you with wide range problems such as how to set up your runs most efficiently, how to parallelize or optimize your code, and how to use and install libraries. If you have problems, questions, or requests, and you couldn't find the answer in this wiki, send an e-mail with all relevant information to support@scinet.utoronto.ca.

SciNet has currently two main clusters available for users, which can access the same file system.

1.1 General Purpose Cluster (GPC)

- 3780 nodes with 8 cores each (two 2.53GHz quad-core Intel Xeon 5500 *Nehalem* x86-64 processors)
- HyperThreading lets you run 16 threads per node efficiently.
- 16GB RAM per node
- Running CentOS 5.3 linux.
- Gigabit ethernet network on all nodes (4:1 blocking switch): for management, disk I/O, boot, etc.
- InfiniBand network on 1/4 of the nodes (1:1 non-blocking): only used for job communication
- 306 TFlops $\longrightarrow$ #16 on the June 2009 *TOP500* list of supercomputer sites (#1 in Canada)
- Moab/Torque schedules by node with a maximum wall clock time of 48 hours.

1.2 Tightly Coupled System (TCS)

- 104 nodes with 32 cores (16 dual-core 4.7GHz POWER6 processors).
- Simultaneous MultiThreading allows two tasks to be very efficiently bound to each core.
- 128GB RAM per node
- Running AIX 5.3L operating system.
- Interconnected by full non-blocking InfiniBand
- 62 TFlops $\longrightarrow$ #80 on the June 2009 *TOP500* list of supercomputer sites (#3 in Canada)
- Moab/LoadLeveler schedules by node. The maximum wall clock time is 48 hours.
- Access to this highly specialized machine is not enabled by default. For access, email us explaining the nature of your work. Your application should scale well to 64 processes/threads to run on this system.

1.3 Disk space

- 1790 1TB SATA disk drives, for a total of 1.4 PB of storage
- Two DCS9900 couplets, each delivering 4-5GB/s read/write access to the drives
- Single *GPFS* file system on both the TCS and the GPC
- I/O goes over the ethernet network on the GPC, and over the InfiniBand network on the TCS
- The file system is a shared resource. Creating many small files or opening and closing files with small reads, and similar inefficient I/O practices hurt your job's performance, and are felt by other users too.
- See 'Data Management' below for more details.

location	quota	block-size	time-limit	backup	devel	comp
/home/USER	10GB	256kB	perpetual	yes	read/write	read-only
/scratch/USER	20TB/one million files	4MB	3 months	no	read/write	read/write

2 Usage

Using SciNet's resources is significantly different from using a desktop machine. The rest of this document is meant to guide you through the process.

Computing on SciNet's clusters is done through a batch system. In its simplest form, it is a four stage process:

1. Login with ssh to the login nodes and transfer files.
   These login nodes are gateways, you do not run or compile on them!
2. Ssh to a devel node where you load modules, compile your code and write a script for the batch job.
   The scheduler works per node, you cannot request jobs for less than 8 (GPC) or 32 (TCS) cores.
3. Move the script, input data, etc. to the scratch disk, *as you cannot write to your home directory from the compute nodes*. Submit the job to a queue.
4. After the scheduler has run the job on the compute nodes (this can take some time), and the job is completed, deal with the output of the run.

2.1 Login

Access to the SciNet systems is via secure shell (`ssh`) only. Ssh to the gateway `login.scinet.utoronto.ca`:

```
ssh -l <username> login.scinet.utoronto.ca
```

The login nodes are a front end to the data centre, and are part of neither of the two compute clusters. For anything but small file transfer and viewing your files, you should next login to the GPC or the TCS through their development nodes (gpc01,...,gpc04 for GPC, and tcs01 or tcs02 for TCS, respectively).

- More about ssh and logging in from Windows at the wiki page Ssh .
- The SciNet firewall monitors for too many connections, and will shut down access (including previously connections) from your IP address if more than four connection attempts are made within 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 at the wiki page Essentials .

2.2 Software modules

Most software and libraries have to be loaded using the `module` command. This allows us to easily keep multiple versions of software for different users on the system, and it allows users to easily switch between versions. The module system simply sets up environment variables (PATH, MANPATH, LD_LIBRARY_PATH, etc.) and works similarly on the GPC and the TCS.

Basic usage of the `module` command:

`module load <module-name>`	to use particular software
`module unload <module-name>`	to stop using particular software
`module switch <module1> <module2>`	to unload module1 and load module2
`module purge`	to remove all currently loaded modules
`module avail`	to list available software packages (+ all versions)
`module list`	to list currently loaded modules in your shell

You should load frequently used modules in the file `.bashrc` in your home directory.

Many modules are available in several versions (e.g. `intel/intel-v11.1.056` and `intel/intel-v11.1.072`). When you load a module with its short name (the part before the slash '/',e.g., `intel`), you get the most recent and recommended version of that library or piece of software. In general, you should use the short module name, especially since we may upgrade to a new version and deprecate the old one. By using the short module name, you ensure that your existing `module load` commands still work. (However, for reproducibility of your runs, you should record the full names of loaded modules.)

Modules that load libraries, define the following environment variables pointing to the location of library files, include files and the base directory for use Makefiles:

```
SCINET_[shortmodulename]_LIB
SCINET_[shortmodulename]_INC
SCINET_[shortmodulename]_BASE
```

That means that to compile code that uses that package you should add

```
-I${SCINET_[shortmodulename]_INC}
```

to the compile command, and to the link command, you should add

```
-L${SCINET_[shortmodulename]_LIB}
```

- On April 21, 2011, the module list for the GPC contained:
  `intel, gcc, intelmpi, openmpi, nano, emacs, xemacs, autoconf, cmake, git, scons, svn, ddd, gdb, mpe, openspeedshop, scalasca, valgrind, padb, grace, gnuplot, vmd, ferret, ncl, ROOT, paraview, pgplot, ImageMagick,netcdf, parallel-netcdf, ncview, nco, udunits, hdf4, hdf5, encfs, gamess, nwchem, gromacs, cpmd, blast, amber, gdal, meep, mpb, R, petsc, boost, gsl, fftw, intel, extras, clog, gnu-parallel, guile, java, python, ruby, octave, gnu-parallel, gotoblas`
- and for the TCS:
  `upc, xlf/13.1, vacpp, mpe, scalasca, hdf4, extras, netcdf, parallel-netcdf, nco, gsl, antlr, ncl`
- A current list of available software is maintained on the wiki page Software and Libraries .
- The IBM compilers are standard available on the TCS and do not require a module to be loaded, although newer versions may be installed as modules.
- Math software supporting things like BLAS and FFT is either standard available, or part of a module: on the GPC, there is the Intel's Math Kernel Library (MKL) which is part of the intel module and the gotoblas modules, while on the TCS, IBM's ESSL high performance math library is standard available.
- Other commercial packages (MatLab, Gaussian, IDL,...) are **not** available on SciNet for licensing reasons. But Octave, a highly MatLab-compatible open source alternative, is available as a module.

2.3 Compiling

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. Instead, you should compile on the specialized devel nodes, aptly named gpc01, gpc02, gpc03, gpc04 for the GPC, and tcs01 and tcs02 for the TCS. These nodes may also be used for short, small scale test runs (although on the GPC there's a specialized queue for that). Please test your job's requirements and scaling behaviour before submitting a large scale computation to the queue. For available

tools to analyze and improve your code's performance, see at the wiki pages Introduction To Performance , Performance And Debugging Tools: GPC , and Performance And Debugging Tools: TCS .

Because the devel nodes 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.

GPC compilation

To compile code for runs on the GPC, you log in from `login.scinet.utoronto.ca` to one of the four GPC devel nodes gpc01, gpc02, gpc03, or gpc04, e.g.

```
ssh gpc04
```

The GPC has compilers for C, C++, Fortran (up to 2003 with some 2008 features), Co-array Fortran, and Java. We will focus here on the most commonly used languages: C, C++, and Fortran.

It is recommended that you compile with the Intel compilers, which are `icc`, `icpc`, and `ifort` for C, C++, and Fortran. These compilers are available with the module `intel` (i.e., put `module load intel` in your .bashrc). If you really need the GNU compilers, recent versions of the GNU compiler collection are available as modules, with `gcc,g++,gfortran` for C, C++, and Fortran. The ol' g77 is not supported, but both `ifort` and `gfortran` are able to compile Fortran 77 code.

- Optimize your code for the GPC machine using of at least the following compiler flags
  `-O3 -xhost`
  (`-O3 -march=native` for GNU compilers).
- Add `-openmp` to the command line for OpenMP and hybrid OpenMP/MPI code (`-fopenmp` for GNU).
- The intel module includes the Intel MKL, which has BLAS and FFT support, among other things. The web page software.intel.com/en-us/articles/intel-mkl-link-line-advisor can tell you what to append to the link command when using the MKL.

MPI code can be compiled with `mpif77/mpif90/mpicc/mpicxx`. These commands are wrapper (bash) scripts around the compilers which include the appropriate flags to use MPI libraries. Hybrid MPI/OpenMP applications are compiled with same commands. Currently, the GPC has following MPI implementations installed:

1. Open MPI, in module `openmpi` (v1.4.1)
2. Intel MPI, in module `intelmpi` (v4.0.0)

You can choose which one to use with the module system, but you are recommended to stick to Open MPI unless you have a good reason not to. Switching between the different MPI implementations is not always obvious.

- For hybrid OpenMP/MPI code using Intel MPI, add the compilation flag `-mt_mpi` for full thread-safety.
- If you get the warning '`feupdatreenv is not implemented`', add `-limf` to the link line.
- Other versions of these MPI implementations are installed only to support legacy code and for testing.

TCS compilers

The TCS has compilers for C, C++, Fortran (up to 2003), UPC, and Java. We will focus here on the most commonly used languages: C, C++, and Fortran.

Compilation for the TCS should be done with the IBM compilers on the TCS devel nodes, so from login, do

```
ssh tcs01  or  ssh tcs02
```

The compilers are `xlc,xlC,xlf` for C, C++, and Fortran compilations. For OpenMP or other threaded applications, one has to use 're-entrant-safe' versions `xlc_r,xlC_r,xlf_r`. For MPI applications, `mpcc,mpCC,mpxlf` are the appropriate wrappers. Hybrid MPI/OpenMP applications require `mpcc_r,mpCC_r,mpxlf_r`.

- We strongly suggest the compiler flags
  `-O3 -q64 -qhot -qarch=pwr6 -qtune=pwr6`
  For OpenMP programs, we suggest
  `-qsmp=omp -O4 -q64 -qhot -qarch=pwr6 -qtune=pwr6`
  (the increased optimization level is needed for proper inlining of openmp loops).
- On the link line we suggest using
  `-q64 -bdatapsize:64k -bstackpsize:64k`
  supplemented by
  `-qsmp=omp`
  for OpenMP programs.
- For production runs (i.e., not for runs on tcs01 or tcs02), change `-qarch=pwr6` to `-qarch=pwr6e`.
- To use the full C++ bindings of MPI (those in the MPI namespace) with the IBM `c++` compilers, add `-cpp` to the compilation line. If you're linking several `c++` object files, add `-bh:5` to the link line.

2.4 Testing/debugging

GPC

You can run short test runs on the devel 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, simply type from a devel node

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

Serial production jobs must be bunched together to use all 8 cores. See 3.3 and wiki page User Serial .

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

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

To run a short 4-process MPI run on a single node, type

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

- Use the `debug` queue for longer, multinode test runs.
- mpirun may complain about not being able to find a network (Open MPI) or the list of hosts not being provided (Intel MPI). These warnings are mostly harmless.

For debugging, the GNU (`gdb`) and intel debugger (`idbc`) are available on the GPC.

TCS

Short test runs are allowed on devel nodes if they only don't use much memory and only use a few cores.

To run a short 8-thread OpenMP test run on tcs02:

```
OMP_NUM_THREADS=8 ./<executable> [arguments]
```

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

```
mpiexec -n 16 ./<executable> [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`.

The standard debugger on the TCS is called `dbx`.

2.5 Running your jobs

To run a job on the compute nodes you must submit it to a queue. You can submit jobs from the devel nodes in the form of 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 Moab Workload Manager, with Torque (PBS) 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. It schedules by nodes, so you cannot request e.g. a two-core job. It is the user's responsibility to make sure that the node is used efficiently, i.e., all cores on a node are kept busy.

The best way to learn how to write the job scripts is to look at some examples, which are given in sections 3 and 4 below. You can use these example scripts as starting points for your own.

Note that it is best 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, copy essential results to your home directory after your runs have finished.

- 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.
- Scheduling big jobs greatly affects the queue and other users, so you have to talk to us first to run massively parallel jobs (over 2048 cores). We will help make sure that your jobs start and run efficiently.
- See Essentials#Usage Policy on the SciNet wiki page.
- Users needing 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.
- Users with an NRAC/LRAC allocation, see the wiki page Accounting on the Scheduler page about group/RAP priorities.

GPC

There are three queues available on the GPC:

queue	time(hrs)	max jobs	max cores
batch	48	32/1000	256/8000 (512/16000 threads)
debug	2/0.5	1	16/64 (32/128 threads)
largemem	48	1	16 (32 threads)

You submit to these queues with

```
qsub [options] <script>
```

where you will replace `<script>` with the file name of the submission script. Common options are:

`-l`: specifies requested nodes and time, e.g.

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

The "ppn=8" part is mandatory, since scheduling goes by node, and each node has 8 cores!

`-q`: specifies the queue, e.g.

```
-q largemem
-q debug
```

`-I` specifies that you want an interactive session; a script is not needed in that case.

The number of nodes option is **mandatory**, but can be specified in the job script as well.

- The GPC nodes have HyperThreading enabled, which allows efficient switching between tasks, and makes it seem like there are 16 processors rather than 8 on each node. Using this requires no changes to the code, only running 16 rather than 8 tasks on the node. For OpenMP application, setting `OMP_NUM_THREADS=16` may make your job run faster. For MPI, try `-np 16`.

- Once the job is incorporated into the queue, you can use: `showq` to show the queue, and job-specific commands such as `showstart, checkjob, canceljob`
- There is no separate queue for infiniband nodes. You request these through the option `:ib`.
- You cannot request less than 8 processors per node, i.e., ppn=8 always in the qsub line.
- Even when you use HyperThreading, you should still request ppn=8.
- The largemem queue is exceptional, in that it provides access to two nodes (only) that have 16 processors and 128GB of ram. (for these you can have ppn=16, but ppn=8 will be excepted silently).
- There is no queue for serial jobs, so if you have serial jobs, you will have to bunch together 8 of them to use the full power of a node (Moab schedules by node). See wiki page User Serial .
- *To make your jobs start faster:*
  - Reduce the requested time (`walltime`) to be closer to the estimated run 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 InfiniBand.
- Read more on the wiki pages GPC Quickstart , Scheduler

TCS

For the TCS, there is only one queue:

queue	time(hrs)	max jobs	max cores
verylong	48	2/25	64/800 (128/1600 threads)

Submitting is done with

```
llsubmit <script>
```

and `llq` shows the queue.

- The POWER6 series of processors has a facility called Simultaneous MultiThreading 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` to a number larger than 32 makes your job run faster. For MPI, increase `tasks_per_node > 32`.
- Once 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.
- To make your jobs start sooner, reduce the `wall_clock_limit`)to be closer to the estimated run time (perhaps adding about 10 % to be sure). Shorter jobs are scheduled sooner than longer ones.
- Read more on the wiki pages TCS Quickstart , Scheduler

2.6 Data Management

Storage Space

The storage at SciNet is divided over different file systems. The two most important ones are /home and /scratch. Every SciNet user gets a 10GB directory on /home (called /home/$USER) which is regularly backed-up. On the compute nodes of the GPC clusters, /home is mounted read-only; thus GPC jobs can read files in /home but cannot write to files there. /home is a good place to put code, input files for runs, and anything else that needs to be kept to reproduce runs. In addition, every SciNet user gets a directory in /scratch, in which up to 20TB could be stored (although there is not enough room for each user to do this!). Scratch is always mounted as read-write. Thus jobs would normally write their output somewhere in /scratch. *There are NO backups of /scratch.* Furthermore, /scratch is purged routinely (i.e., files on it have

a time-limit), so that all users running jobs and generating large outputs will have room to store their data temporarily. Computational results which you want to save for longer than this must be copied off of SciNet entirely.

location	quota	block-size	time-limit	backup	devel	comp
/home/USER/	10GB	256kB	perpetual	yes	rw	ro
/scratch/USER/	20TB/one million files	4MB	3 months	no	rw	rw

Do not keep many small files on the system. They waste quite a bit of space, especially on /scratch, as the block size for the file system is 4MB, but even on home, with a block size of 256kB, you can at most have 40960 files no matter how small they are, so you would run out of disk quota quite rapidly.

- Read more on the wiki page Data Management .

I/O

The compute nodes do not contain hard drives, so there is no local disk available to use during your computation. The available disk space, i.e., the home and scratch directories, are all part of the GPFS file system which runs over the network. GPFS is a high-performance file system which provides rapid reads and writes to large data sets in parallel from many nodes. As a consequence of this design, however, ***it performs quite poorly at accessing data sets which consist of many, small files.***

Because of this file system setup, you may well find that you have to reconsider the I/O strategy of your program. The following points are very important to bear in mind when designing your I/O strategy

- Do not read and write lots of small amounts of data to disk. Reading data in from one 4MB file can be enormously faster than from 100 40KB files.
- Unless you have very little output, make sure to write your data in binary.
- Having each process write to a file of its own is not a scalable I/O solution. A directory gets locked by the first process accessing it, so the other processes have to wait for it. Not only has the code just become considerably less parallel, chances are the file system will have a time-out while waiting for your other processes, leading your program to crash mysteriously.
- Consider using MPI-IO (part of the MPI-2 standard), NetCDF or HDF5, which allow files to be opened simultaneously by different processes. You could also use dedicated process for I/O to which all other processes send their data, and which subsequently writes this data to a single file.

If you must read and write a lot to disk, consider using the ramdisk. On the GPC, this is setup such that you can use part of a compute node's ram like a local disk. This *will* reduce how much memory is available for your program. The ramdisk can be accessed using /dev/shm/ and is currently set to 8GB. Anything written to this location that you want to preserve must be copied back to the /scratch file system as /dev/shm is wiped after each job and since it is in memory will not survive through a reboot of the node.

- See wiki pages Data Management and User Ramdisk .

Transfers

All traffic to and from the data centre goes via SSH, or secure shell. This is a protocol which sets up a secure connection between two sites. In all cases, incoming connections to SciNet go through relatively low-speed connections to the login.scinet gateways, but there are many ways to copy files on top of the ssh protocol. What node to use for data transfer to and from SciNet depends mostly on the amount of data to transfer:

Moving less than 10GB through the login nodes

The login nodes are visible from outside SciNet, which means that you can transfer data to and from your own machine to SciNet using scp or rsync starting from SciNet or from your own machine. The login node has a cpu time out of 5 minutes, which means that even if you tried to transfer more than 10GB, you would

probably not succeed. While the login nodes can be used for transfers of less than 10GB, using the data mover node would still be faster.

Moving more than 10GB through the datamover nodes

Serious moves of data (more than 10GB) to or from SciNet should be done from the datamover1 or datamover2 node. From any of the interactive SciNet nodes, one should be able to ssh to datamover1 or datamover2. These machines have the fastest network connection to the outside world (by a factor of 10; a 10Gb/s link as vs 1Gb/s). Consider that datamover2 is sometimes under heavy load for sysadmin purposes, but datamover1 is intended for user traffic only.

Transfers must be originated from the datamovers; that is, one can not copy files from the outside world directly to or from a datamover node; one has to log in to that datamover node and copy the data to or from the outside network. Your local machine must be reachable from the outside, either by its name or its IP address. If you are behind a firewall or a (wireless) router, this may not be possible. You may need to ask your system administrator to allow datamover to ssh to your machine.

- Transfers through login time-out after 5 minutes, so if you have a slow connection, use datamover1.
- Read more on the wiki page on Data Management .

2.7 Acknowledging SciNet

In publications based on results from SciNet computations, please use the following acknowledgment:

> *Computations were performed on the <systemname> supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto.*

where you replace <systemname> by GPC or TCS.

Also please cite the SciNet datacentre paper:

> Chris Loken *et al.*, *SciNet: Lessons Learned from Building a Power-efficient Top-20 System and Data Centre*, J. Phys.: Conf. Ser. **256**, 012026 (2010).

We are very interested in keeping track of such SciNet-powered publications! We track these for our own interest, but such publications are also useful evidence of scientific merit for future resource allocations as well. Please email details of any such publications, along with PDF preprints, to support@scinet.utoronto.ca.

In any talks you give, please feel free to use the SciNet logo, and images of GPC, TCS, and the data centre. These can be found on the wiki page Acknowledging SciNet .

3 GPC examples

All example presume that the necessary modules are loaded in `.bashrc` (i.e., `module load intel openmpi`). Submission of these examples can be done using `qsub <script>` where you will replace `<script>` with the file containing the submission script. There are no options given to qsub in this case, because the scripts contain all the necessary requests. The qsub command 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`.

3.1 Shared memory parallel jobs (OpenMP)

Compiling

```
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

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

Submitting

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 openmp-test
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=8
./example
```

and submit the job with qsub.

3.2 Distributed memory parallel jobs (MPI)

Compiling

```
mpif77 -O3 -xhost example.f   -c -o example.o
mpif90 -O3 -xhost example.f90 -c -o example.o
mpicc  -O3 -xhost example.c   -c -o example.o
mpicxx -O3 -xhost example.cpp -c -o example.o
```

Linking

```
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 - ethernet

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 mpi-test-eth
cd $PBS_O_WORKDIR
mpirun -np 16 ./example
```

and submit the job with qsub.

Submitting - infiniband

```
#!/bin/bash
#MOAB/Torque submission script for SciNet GPC (infiniband)
#PBS -l nodes=2:ib:ppn=8,walltime=1:00:00
#PBS -N mpi-test-ib
cd $PBS_O_WORKDIR
mpirun -np 16 ./example
```

and submit the job with qsub.

- The MPI libraries automatically use infiniband or ethernet depending on the nodes your job runs on.
- As a result, when using ethernet, the MPI libraries print out (library-dependent) mostly harmless warning messages that they cannot find/use infiniband.
- To suppress these messages for OpenMPI, add a flag `--mca btl self,sm,tcp` to the mpirun command.
- To suppress these messages for IntelMPI, add `-env I_MPI_FABRICS shm:tcp` after `-np 16`.
- Remember to remove ethernet-specific options if you switch to infiniband, or you'll still get ethernet!
- Read more on the wiki: GPC MPI Versions

3.3 Serial jobs

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 acceptable 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.

Compiling

```
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

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

Submitting

Create a script in the same directory which bunches 8 serial jobs together. You could do this by creating 8 sub-directories, copying 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-test
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
```

and submit the job with qsub.

- *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 reasonable idea of how much memory the jobs require. The GPC compute nodes have about 14GB in total available to user jobs running on the 8 cores (less, roughly 13GB, on gpc01..04). So the jobs have to be bunched in ways that will fit into 14GB. 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.
- More advanced techniquues for running serial jobs while keeping 8 processors busy, can be found on the wiki page User Serial .

3.4 Hybrid MPI/OpenMP jobs

Compiling

```
mpif77 -openmp -O3 -xhost example.f   -c -o example.o
mpif90 -openmp -O3 -xhost example.f90 -c -o example.o
mpicc  -openmp -O3 -xhost example.c   -c -o example.o
mpicxx -openmp -O3 -xhost example.cpp -c -o example.o
```

Note: you have to specify the `-mt_mpi` flag as well if you are using Intel MPI instead of Open MPI.

Linking

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

Submitting

To run on 3 nodes, each node having 2 MPI processes, each with 4 threads, use a script such as

```
#!/bin/bash
#MOAB/Torque submission script for SciNet GPC (ethernet)
#PBS -l nodes=3:ppn=8,walltime=1:00:00
#PBS -N hybrid-test-eth
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=4
mpirun --bynode -np 6 ./example
```

and submit the job with qsub.

- The `--bynode` option is essential; without it, MPI processes bunch together in eights on each node.
- For Intel MPI, that option should be replaced by `-ppn 2`.
- For infiniband, add `:ib` to the `-l` option.
- Note the remarks above about using ethernet and warning messages given by OpenMPI and IntelMPI.

4 TCS examples

4.1 Shared memory parallel jobs (OpenMP)

Compiling

```
xlf_r -qsmp=omp -q64 -O4 -qhot -qarch=pwr6 -qtune=pwr6 example.f   -c -o example.o
xlc_r -qsmp=omp -q64 -O4 -qhot -qarch=pwr6 -qtune=pwr6 example.c   -c -o example.o
xlC_r -qsmp=omp -q64 -O4 -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

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          = $(job_name).$(jobid).out
#@ error           = $(job_name).$(jobid).err
#@ wall_clock_limit= 04:00:00
#@ queue
export target_cpu_range=-1
cd /scratch/<username>/<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 (replacing `<script>` with the file containing the submission script)

```
llsubmit <script>
```

4.2 Distributed memory parallel jobs (MPI)

Compiling

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

Submitting

Create a script along the following lines

```
#LoadLeveler submission script for SciNet TCS: MPI job
#@ job_name        = <some-descriptive-name>
#@ initialdir      = /scratch/<username>/<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 (replacing `<script>` with the file containing the submission script)

```
llsubmit <script>
```

4.3 Hybrid MPI/OpenMP jobs

Compiling

```
mpxlf_r     -qsmp=omp -q64 -O4 -qhot -qarch=pwr6 -qtune=pwr6 example.f   -c -o example.o
mpcc_r      -qsmp=omp -q64 -O4 -qhot -qarch=pwr6 -qtune=pwr6 example.c   -c -o example.o
mpCC_r -cpp -qsmp=omp -q64 -O4 -qhot -qarch=pwr6 -qtune=pwr6 example.cpp -c -o example.o
```

Linking

```
mpxlf_r -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
mpcc_r  -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
mpCC_r  -qsmp=omp -q64 -bdatapsize:64k -bstackpsize:64k example.o -o example
```

Submitting

To run on 3 nodes, each with 2 MPI processes that have 32 threads, create a file `poe.cmdfile` containing

```
ccsm_launch ./example
ccsm_launch ./example
ccsm_launch ./example
ccsm_launch ./example
ccsm_launch ./example
ccsm_launch ./example
```

Create a script along the following lines

```
#@ 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
#@ task_geometry   = {(0,1)(2,3)(4,5)}
#@ node_usage      = not_shared
#@ output          = $(job_name).$(jobid).out
#@ error           = $(job_name).$(jobid).err
#@ wall_clock_limit= 04:00:00
#@ core_limit      = 0
#@ queue
export target_cpu_range=-1
cd /scratch/<username>/<some-directory>
export MEMORY_AFFINITY=MCM
export THRDS_PER_TASK=32:32:32:32:32:32
export OMP_NUM_THREADS=32
poe -cmdfile poe.cmdfile
wait
```

and submit with `llsubmit <script>`.