New User Seminar: Part 2 (best practices)

General Interest Seminar
January 2015

Hugh Merz
merz@sharcnet.ca

compute • calcul
CANADA

SHARCNET™

Session Outline

- Submitting Jobs
  - Minimizing queue waits
  - Investigating jobs
  - Checkpointing
- Efficiency and high performance
- Software / shell environment configuration
- Data management

What is a job?

- invocation of a program or script on a particular system
  - may call other scripts/programs, possibly multiple times
  - may be serial or parallel and require GPUs, access to significant storage and/or system memory
  - the wallclock run time may not be known in advance
- the command to be executed, plus it's resource requests, is submitted to the system as a job via *sqsub*
  - The underlying system uses a script interface, see `*man qsub*`
- allows for sharing of large systems

What happens to jobs?

- Jobs are submitted to a system program called a scheduler
  - decides which jobs run when, and where, based on administrative policies (eg. accrued usage; fairshare)
- As jobs proceed from being queued, started and finally completed, the job scheduler (and / or resource manager) records important information (eg. exit status)
  - SHARCNET collects this data stores it for each job in our database

Submitting jobs

- Automate as much as possible
  - Learn the shell (bash) or another language (Eg. Python) and how to write scripts!
- Use the best system for the job at hand (or where it will run asap!)
  - Which system should I use?
  - Available resources and queue statistics
- Use efficient submission methods
  - Submitting lots of jobs
  - Ability to chain jobs (*sqsub -w jobid1 jobid2 ...*)

Submitting jobs: sqsub tips

- pick appropriate resource estimates:
  - *--mpp* =~1.2x maximum virtual memory
  - *-r* =~1.3x maximum wallclock run duration
- different process/thread geometries
  - *-N, --pack, –-ppn* and *–-tpp*
  - *–nompirun* for unusual requirements
- may need extended shell resource limits
  - *-f dostack* and *-f permitcoredump*
- --gpp is GPUs per process (/node for MPI)
  - GPUs are *compute exclusive* by default

Investigating jobs

- Running and completed jobs will show up in My Account -> Usage -> Activity
- Use *qstat* / *sqjobs* to get more info about presently running jobs
  - `sqjobs $USER ; sqjobs [-l|-L] $JOB_ID`
  - can log in to compute nodes to check on jobs with *ps*/*uptime*/*top*
    - use *sqjobs -l* to get nodelist, *pdsh* to run a command on multiple hosts
    - attach a debugger to see a running process' stack (*man gdb ; search for "attach"*)
- historical resource usage: Ganglia

TMM exit codes

- If the exit code is < 128, it is an internal scheduler error, report it to help@sharcnet.ca
- If it's between 129 and 255 then the program exit code is (the reported code – 128)
  - eg. *137 – 128 = 9 , SIGKILL*
  - to see most signals, run `*kill -l*`
  - You'll usually have to debug your program to solve the problem
- Job output gives advice concerning common errors

Job Output File: Epilogue

```
--- SharcNET Job Epilogue ---
               job id: 5519935
          exit status: 0
             cpu time: 11s / 1200s (0 %)
         elapsed time: 15s / 300s (5 %)
       virtual memory: 156.1M / 1.0G (15 %)

Job completed successfully
WARNING: Job only used 5 % of its requested walltime.
WARNING: Job only used 0 % of its requested cpu time.
WARNING: Job only used 18 % of allocated cpu time.
WARNING: Job only used 15% of its requested memory.
```

Submitting Jobs: checkpointing

- Checkpointing refers to the practice of storing the state of your program such that you may resume it at a later date
- Why checkpoint?
  - Failure recovery
  - To complete calculations that take longer than the queue runtime limit
- Methods for checkpointing
  - coding yourself, *BLCR*
- How often should I checkpoint?
  - Balancing cost vs. Potential gain

BLCR

```
$ ssh orca
$ cc ~merz/best_practices/blcr/sleeper.c
$ sqsub -q serial -n 1 -r 60m -o ofile.%J $snrun ./a.out
submitted as jobid 61756
$ snchkpt 61756 a.out
$ sqkill 61756
$ tail ofile.61756
$ sqsub -q serial -n 1 -r 60m -o ofile.%J $snrestart 61756
$ tail -f  ofile.61759
```

- Have to run *snchkpt* periodically
- Generally not portable to other clusters
- Only works with dynamically linked executables
- Output file specified in second *sqsub* command must differ from the output file in the first

Why use HPC?

- Two main justifications for going beyond your basic desktop computer (both may apply!):
  - Solve a problem more quickly
  - Solve a larger problem
- The particular HPC platform to use depends on the computational problem
  - Have to consider and weigh different factors, including computer utilization/efficiency, time to results, programming effort, ...

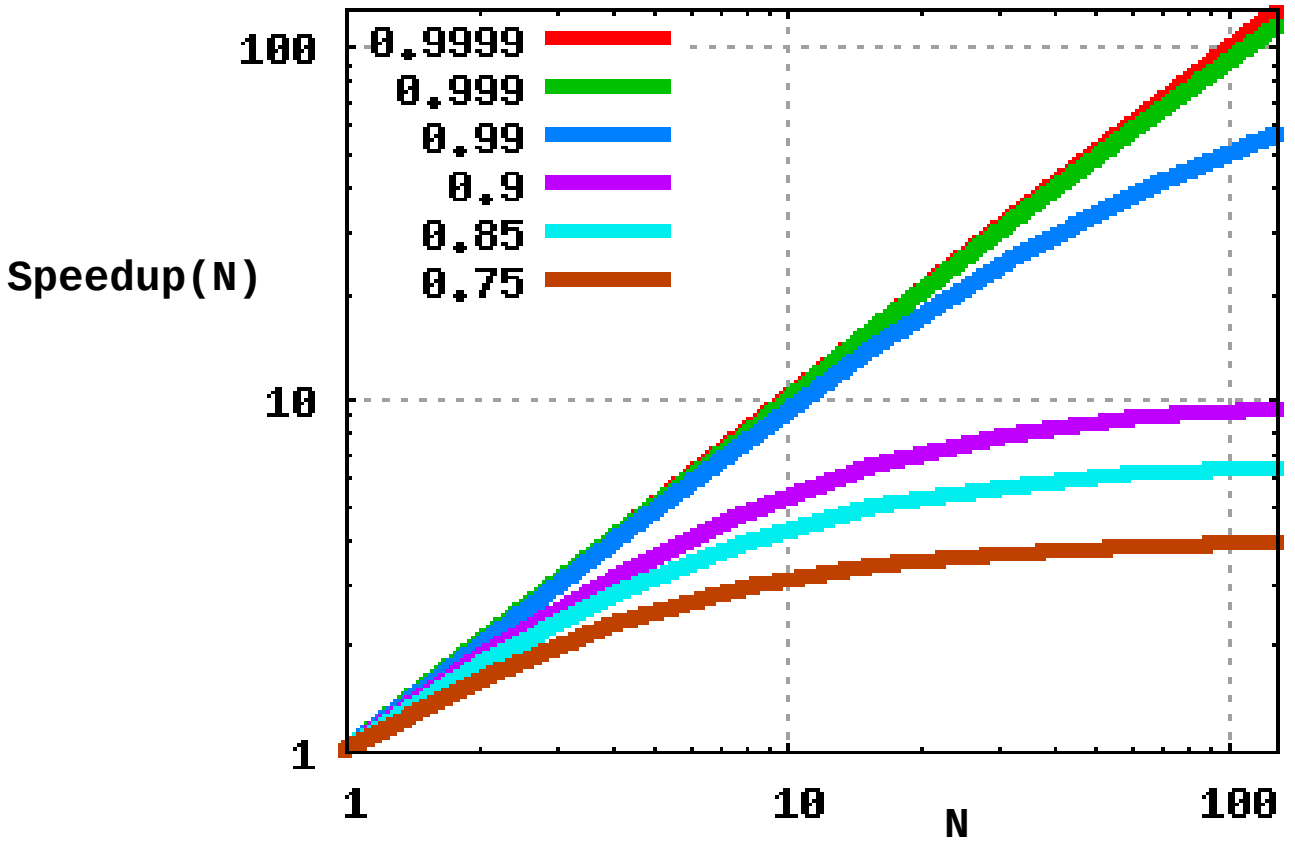
Efficiency and High Performance

- Compiling: make sure you are using optimization flags (*-show* , check man page)
  - can even compare performance across compilers, flags
- Profiling: if you're not familiar with your program, or it isn't widely used, profile it and try to understand why it takes so long to run
  - MAP profiling tool
- Scaling: make sure you've adequately measured the scaling performance of your parallel application using representative, production-scale tests

Limit to Parallelism: Amdahl's Law

Attainable speedup depends on how much of the calculation can be done in parallel

$$\text{Speedup}(N) = [\ (1 - \text{serial\_execution}) + (\text{parallel\_execution} / N)\ ]^{-1}$$



SHARCNET™

Modules

- Basically *module* runs a script (*modulefile*) to configure your shell environment at SHARCNET
- Main (necessary) module commands:
  - *module list*
  - *module avail*
  - *module show $PACKAGE*
  - *module load $PACKAGE*
  - *module unload $PACKAGE*
- Every module has a default version

Module / shell tips

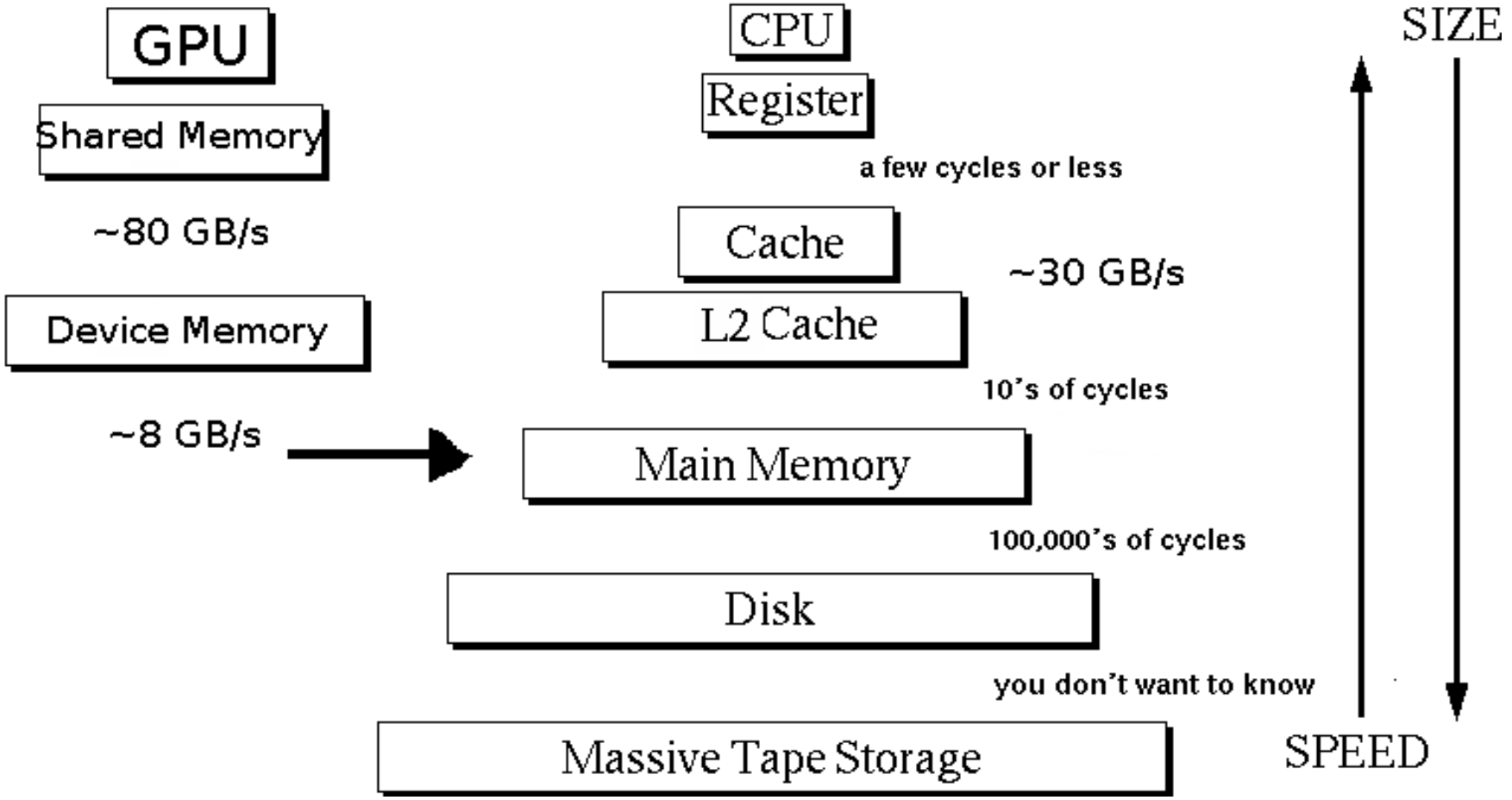
- Presently, *module* does not resolve all dependencies / conflicting transactions!
- SHARCNET compiler modules load libraries via LD_RUN_PATH – the linking environment is preserved for the application (don't use LD_LIBRARY_PATH !)
- For other module settings to persist between login sessions you must add them to your *~/.bashrc* shell configuration file
  - load different modules on different systems
  - set up command aliases, functions, etc.

Computer Architecture Basics

- A computer is essentially a calculator
  - Input data -> processing -> output data

- Main components in a modern computer:
  - CPU (Central Processing Unit)
    - the calculator, may include coprocessors (eg. GPU)
  - RAM (Random Access Memory)
    - working memory for the calculations in progress
  - I/O (input / output)
    - networking (inter-computer I/O)
    - storage (hard drives, persistent memory)

Data Locality: Size vs. Speed



Intel Sandy Bridge:

- 32 kB data + 32 kB instruction L1 cache (3 clocks) per core
- 256 kB L2 cache (8 clocks) per core
- Shared 8MB L3 cache (~24 clocks)

HPC Storage

- HPC systems typically employ one or more large storage pools that are network-attached
  - These are typically parallel in nature
    - in disks per storage server (eg. RAID) and in storage targets (servers) per filesystem (eg. Lustre)
- Clusters may also have node-local storage (eg. /tmp) for temporary files and explicit caching

Data Storage Recommendations

- Use storage that is as close to the CPU as possible (memory hierarchy / less contention)
- Use backed-up storage for important files
  - Or back it up yourself, eg. Incrementals
  - Git and SVN are handy for revision control and multi-developer environments
- Archive historical data
- Compress your data (if possible)!
  - Gzip, tar, DAR, zlib, etc.
- Aggregate files; directories become unwieldy beyond 10,000 files

Storage: Performance

- In principle, /work and /scratch at SHARCNET are of roughly equivalent performance (10G backbone)
  - /work more prone to contention?
- Obtaining optimal performance may require the use of specialized parallel filesystem-aware techniques (eg. MPI-IO, file striping) and/or software libraries (eg. HDF, pNetCDF)
- Excessive file operations can dramatically impact global performance (metadata bottleneck)

Transferring Data

- GUI programs are convenient, eg. Globus
- For single files *scp* works fine;
  - **-C** (compression; less bandwidth, more CPU)
  - **-c** (cipher, eg. '*blowfish*' or '*none*' - less CPU)
  - **-r** (recursive; all files and directories in path)
- For multiple files use *rsync*
  - Ability to resume interrupted downloads; only copy over changed files
  - **`rsync -avz foo:src/bar/ /data/tmp`**

Transferring Data: fine points

- Only the cluster login nodes can see the outside world; all transfers typically through them or **dtn.sharcnet.ca**
  - Limited process runtime; long running rsyncs get killed
    - aggregate your files to avoid slow rsync processing, or use dtn
- Different networks have different routing paths and bandwidth into SHARCNET
- Massive (more than a few TB) datasets may need special arrangements – contact help@sharcnet.ca if you're encountering problems

Summary

Take your time to learn about the computers and the nature of your computation so you can contribute to effective use of the shared resources and get your results as fast as possible

Ask questions - email help@sharcnet.ca