Using the GWDG Scientific Compute Cluster - An Introduction

by Christian Boehme and Tim Ehlers

Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

Am Fassberg, 37077 Göttingen

Fon: 0551 201-1510 Fax: 0551 201-2150
gwdg@gwdg.de www.gwdg.de

Outline

Section 1

Connecting to the frontend via ssh (openssh, or PuTTY on Windows)

ssh to the frontends gwdu101.gwdg.de, gwdu102.gwdg.de or gwdu103.gwdg.de

- Linux or OS X: “ssh gwdu101.gwdg.de -l {GWDG-USERID}“
- Windows: Download putty.exe from http://www.chiark.greenend.org.uk/~sgtatham/putty/
  - ➥ and start it. Enter “gwdu101.gwdg.de“ for Hostname and press open
  - ➥ Press “Yes“ to trust the connection
  - ➥ login as: {GWDG-USERID}
  - ➥ Enter password

```
The authenticity of host 'gwdu101.gwdg.de (134.76.8.101)' can't be
established.
ECDSA key fingerprint is SHA256:sIJNEepmILeEq/7Zqq4HCtpTM8L98arWTny5EiAX+gI.
or
ECDSA key fingerprint is 7c:52:2b:17:f8:ba:29:bd:c5:45:d1:1a:9e:8d:d6:f0.
or
RSA key fingerprint is b9:f9:46:0f:23:c8:8d:76:b9:83:b9:1b:f6:5e:d5:6b.
or
ED25519 256 key fingerprint is e3:ef:39:f5:df:4f:c2:e2:c4:d0:28:95:46:6c:56:39.
Are you sure you want to continue connecting (yes/no)?
```

ssh to the frontends gwdu101.gwdg.de, gwdu102.gwdg.de or gwdu103.gwdg.de

- gwdu101 (and transfer-scc): Abu-Dhabi AMD Opteron 6220
  - ➥ processor features identical to gwdaXXX
  - ➥ older nodes in fat-queue
  - ➥ access to /scratch
- gwdu102: Sandy-Bridge Intel E5-2670 v1
  - ➥ processor features identical to gwddXXX
  - ➥ older nodes in mpi-queue
  - ➥ access to /scratch
- gwdu103: Broadwell Intel E5-2650 v4
  - ➥ processor features identical to dfaXXX, dmpXXX, dgeXXX, dteXXX
  - ➥ new nodes in fat and mpi queue (and gpu queue)
  - ➥ access to /scratch2

Section 2

The most important Linux commands

basic commands using Linux

- List the current directory you are in, “ls“
  - ➥ List the “hidden“ files (beginning with “.“) too, “ls -a“
  - ➥ All files in an extended manner, “ls -la“ or just type “l“
- Let’s look at three lines output

```
drwxrwxrwx   3 tehlers users    4096  4. Apr 17:29 test
-rw-r--r--   1 tehlers users     283 24. Sep 2003  Info.txt
lrwxrwxrwx   1 root root          23  Jul 22 12:10 passwd -> /etc/passwd
```

ten permission flags:

1 directory flag, “d“ for directory, “-“ for normal file, “l“ for symlink

2,3,4 read, write, execute permission for User (Owner of the file)

5,6,7 read, write, execute permission for Group

8,9,10 read, write, execute permission for Others

Changing the language, what if I don't undestand German

```
> echo $LANG
de_DE.UTF-8
> rm test
rm: reguläre leere Datei "test" entfernen?
```

```
> export LANG=en_US.UTF-8
> rm test
rm: remove regular empty file `test'?
```

For persistent English language, put it in your “.profile“:

```
echo 'export LANG=en_US.UTF-8' >> ~/.profile
```

File operations and processes

- cd (change directory)
- top (display Linux processes, sorted list)
- ps (display current processes), imp. opt. a [all sessions], u [owner], x [all], w [wide], ww [even wider]
- touch
- cp, rm, mv, mkdir, rmdir, ln
- df, df -h, df -hl

chmod

- These files attributes (mode bits) we can change with chmod
- chmod can be used in two ways:
  - u (user) g (group) o (others) $\rightarrow$ chmod a+r {file}, chmod a=rx {file}
  - tell the mode bits: e.g. chmod 744 {file}

chmod (2)

- 0-7 are 3 bits: 111 → 7
- same order, like in dir listing: r,w,x

000 0 → no read write or execute allowed

001 1 → x (last bit is set)

010 2 → w (middle bit is set)

011 3 → w,x (last 2 bits are set)

100 4 → r (first bit is set)

101 5 → r,x (first and last bits are set)

110 6 → r,w (first and second bits are set)

111 7 → r,w,x (all 3 bits are set)

chmod (3)

- In sum we have 9 bits now in 3 groups (owner, group, others)
- But there is a 4th group: SUID/SGID/sticky-bits
- SUID/SGID means that the called program will run with the UID or GID of the owner
  - ➥ e.g. if the program owns root and has SUID set, you run the program as root
  - ➥ chmod u+s {file}, or chmod g+s {file}, chmod a+s {file} would set both
  - ➥ Since we are normal users on the system, this is very seldom needed.
- sticky-bit is more relevant for you, if you open a directory for colleagues to write (chmod g=rwx {dir})
  - ➥ the sticky bit prevents others from deleting files, they are not owning. (chmod +t {dir})
  - ➥ e.g. if you create a file, other must not delete it, even though they have write permission to the dir.

umask

- standard umask is “022“ or “u=rwx,g=rx,o=rx“
- umask is the inversion of default file attributes, when creating a file
  - ➥ But you may use it like chmod with u=XXX, g=XXX or o=XXX, to display write “umask -S“
  - ➥ r and w counts for both, files and directories; x is only for directories

Editors, help and Bash

- vi, mcedit, joe, nano

For most commands you can read the manual pages, just type “man {COMMAND}“.

The prompt is a so called “Shell” with built in commands and functions. We are using the “bash”. Type “man bash“ to get an impression about the power and flexibility of that shell.

Editors, help and Bash

- vi, mcedit, joe, nano

For most commands you can read the manual pages, just type “man {COMMAND}“.
The prompt is a so called “Shell“ with built-in commands and functions. We are using the “bash“. Type “man bash“ to get an impression about the power and flexibility of that shell.

nice and renice

- You want to test your program (only short tests!)
- Please be nice on gwdu101, gwdu102 and gwdu103
  - ➥ nice -n 19 {command}
- you forgot to nice and don't want to restart
  - ➥ open a new terminal:
  - ➥ renice -n 19 {process id}

Environment variables

- Where does the system know all the commands we learned today from?
- The bash searches all pathes, the environment variable PATH contains.

```
tehlers@gwdu101:..users/tehlers> echo $PATH
/usr/lib64/qt-3.3/bin:/opt/lsf/9.1/linux2.6-glibc2.3-x86_64/etc:/opt/lsf/9
.1/linux2.6-glibc2.3-x86_64/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sb
in:/usr/sbin:/sbin:/sbin:/usr/sbin
```

The first Shell-Script

For our first Shellscript we need additional information

- grep handles the input and only outputs matching lines, a “grep tehlers“ would only output lines with that name in it.
- A Pipe “|“ puts the output stream (stdout) into the input stream (stdin) of another program:
  - ➥ “ls -la | grep tehlers“ only shows files owned by tehlers or if the filename has a “tehlers“ in it.
- “mktemp -d /scratch/${USER}/XXXXXXXX“ will create a unique directory, e.g. /scratch/tehlers/XymeK4nq and echo it to stdout
- To write an output of a program in a variable, we write “TEMPDIR=$(mktemp -d /scratch/${USER}/XXXXXXXX)“

Let’s write a little Shell script.

The first Shell-Script

For our first Shellscript we need additional information

- grep handles the input and only outputs matching lines, a “grep tehlers“ would only output lines with that name in it.
- A Pipe “|“ puts the output stream (stdout) into the input stream (stdin) of another program:
  - ➥ “ls -la | grep tehlers“ only shows files owned by tehlers or if the filename has a “tehlers“ in it.
- “mktemp -d /scratch/${USER}/XXXXXXXX“ will create a unique directory, e.g. /scratch/tehlers/XymeK4nq and echo it to stdout
- To write an output of a program in a variable, we write “TEMPDIR=$(mktemp -d /scratch/${USER}/XXXXXXXX)“

Let’s write a little Shell script.

Reformatting Data

```
~ > cat 1
Column1         column2         column3
1               2               3
4               5               6
```

We just want column number 2.

```
~ > cat 1 | (while read a b c; do echo $b; done)
column2
2
5
```

Reformatting Data II

```
~ > cat 2
Column1,column2,column3
1,2,3
4,5,6
```

We still want column 2, but the separator is “,“.

```
~ > cat 2 | sed "s/,/ /g" | (while read a b c; do echo $b; done)
column2
2
5
```

Reformatting Data III

We only need line number 2 and column number 2 from file “2“.

```
~ > cat 2
Column1,column2,column3
1,2,3
4,5,6
```

```
~ > cat 2 | sed "s/,/ /g" | (count=0; while read a b c; do let count=$count+1;
if [ "$count" = "2" ]; then echo $b; fi; done)
2
```

Reformatting Data IV

The comma separated list has empty values.

```
~ > cat 3
Column1,column2,column3
1,2,3
4,5,6
7,,9
```

With “ “ as a separator we get:

```
~ > cat 3 | sed "s/,/ /g" | (while read a b c; do echo $c; done)
colum3
3
6
```

We set the bash-variable “IFS“

```
~ > IFS=","
~ > cat 3 | (while read a b c; do echo $c; done)
colum3
3
6
9
```

Stageout from /scratch (not for /scratch2)

- We have a stageout mechanism from /scratch to your Home
- All data you want to have copied into your Home should be located under /scratch/${USER}/scc_backup
- it will be copied during the night to your Home (${HOME}/scc_backup)
- you will get a mail about this process to your GWDG-Account
- If you want to get the mail to an other mailaddress, put the address in ${HOME}/scc_backup/.mailaddress

GWDG

Section 3

Preparing the compilation environment with “modules“

The modules system

- “module avail“ find a list of installed modules
- “module list“ list of currently loaded modules
- “module load software/version“
- “module purge“ unload all modules
- “module unload software“ unload a single module

- Most of the modules just append or prepend a path to PATH and MANPATH variable.
- Or default variables to be found by compiler/configure scripts at compile time.

Table of Contents, Begin Part II

Section 4

Compiling Software

Why Compiling?

- GWDG cannot install all software required by users (see modules for what is available)
- Scientific software is often only available as source code
- Compiling means to create a executable – or a library – from the source code
- Compiling on the target system often yields better performance
- Prepackaged software typically requires administrator (root) privileges ...
  - ➥ (`sudo` or `su` won't work)

Getting and Unpacking the Source Code

- Source code is usually packaged as “tarball”
  - ➥ Look for file extensions “`tar.gz`”, “`tar.bz2`”, “`tgz`”
  - ➥ Naming convention is often name-version.tar.gz
- If the tarball is available on the web use “`wget`” to download
- Use “`tar`” to unpack the tarball
  - ➥ Use “`tar xvzf`” for ‘`tar.gz`”, “`tgz`”
  - ➥ Use “`tar xvjf`” for “`tar.bz2`”

Recipe: wget and tar

Using wget and tar to prepare the source code

```
> mkdir $HOME/build
> cd $HOME/build
> wget <tarball URL>
> tar xvzf <name-version>.tar.gz
> cd <name-version>
```

Compiling (or “Building”) the Software

- Standard method: “`./configure; make; [make check; make install]`”
- Without root privileges: “`--prefix`”
- For better performance: Use Intel compiler and MKL
- For MPI (distributed parallel) applications: Use Intel MPI

About “--prefix”

- “--prefix” is used to specify the base diretory for your software
- Use “./configure --prefix=$HOME” to install directly under $HOME.
- Use e.g. “./configure --prefix=$HOME/software/<name-version>” to install into a software specific directory.

Recipe: Basic Building and Installing

Building and installing software into a specific directory

```
> cd $HOME; mkdir software
> cd $HOME/build/<name-version>
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
> ln -s $HOME/software/<name-version>/bin/* $HOME/bin
> ln -s $HOME/software/<name-version>/lib/* $HOME/lib
> ln -s $HOME/software/<name-version>/include/* $HOME/include
```

Compilers

- The GNU compilers (`gcc`, `gfortran`) are the standard compilers in Linux
- Other compilers are often faster, especially for Fortran code
- Recommended for overall performance: Intel compilers (`icc`, `ifort`)
- Other compilers available at GWDG: PGI, Open64
  - ➥ For special cases and users willing to try several approaches for best performance

Recipe: Using Intel Compilers

Building and installing software with Intel compilers

```
> module load intel/compiler
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Intel Math Kernel Library (MKL)

- A (shared) library is a collection of thematically related subroutines ready to use in a program
- The process of connecting a library to the (compiled) program is called linking
- Intel's Math Kernel Library provides performance optimized linear algebra and Fourier transform functions

Recipe: Using the MKL

Linking programs to MKL

```
> module load intel/compiler
> CC=icc; CXX=icpc; FC=ifort; F77=ifort; F90=ifort
> export CC CXX FC F77 F90
> module load intel/mkl
> export CPPFLAGS="-I${MKLROOT}/include -I${MKLROOT}/include/fftw"
> export LDFLAGS="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64\
> -lmkl_sequential -lmkl_core -lpthread -lm"
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

MPI programs

- MPI programs are meant to run distributed across several computers
- They require to be linked to an MPI library
- The recommended MPI library at GWDG is Intel MPI
- Others available are OpenMPI (tested), MVAPICH, and MVAPICH2

Recipe: Building MPI programs

Building MPI programs with Intel MPI

```
> module load intel/compiler
> module load intel/mpi
> CC=mpiicc; CXX=mpiicpc; FC=mpiifort; F77=mpiifort; F90=mpiifort
> export CC CXX FC F77 F90
> module load intel/mkl
> export CPPFLAGS="-I${MKLROOT}/include -I${MKLROOT}/include/fftw"
> export LDFLAGS="-L${MKLROOT}/lib/intel64 -lmkl_intel_lp64\
> -lmkl_sequential -lmkl_core -lpthread -lm"
> ./configure --prefix=$HOME/software/<name-version>
> make -j 4; make check
> make install
```

Recipe: Building Rmpi for R

Preparation

```
> module load openmpi/gcc
> export OMPI_MCA_mtl=^psm
> echo $MPI_HOME
/cm/shared/apps/openmpi/gcc/64/1.10.1
> R
```

R command line

```
> install.packages("Rmpi", dependencies=TRUE,
    configure.args=c("--with-mpi=/cm/shared/apps/openmpi/gcc/64/1.10.1"
    ))
> install.packages(c("foreach", "doMPI"))
```

Section 5

Efficiently Submitting Jobs to the Cluster

Basic Concepts 1

Cluster A collection of networked computers intended to provide compute capabilities.

Node One of these computers, also called host or server.

frontend Special node provided to interact with the cluster. gwdu101, gwdu102, and gwdu103 in our case.

(Job-)Slot Compute capacity for one task (a process or thread) at a time, usually one processor core.

Job Program calls consisting of one or several parallel tasks.

Batch System Management system distributing job tasks across job slots. In our case Platform LSF, which is operated by shell commands.

Basic Concepts 2

Serial job Job consisting of one task using one job slot.

SMP job Job with shared memory parallelization (often realized with OpenMP), meaning that all tasks need access to the memory of the same node. Consequently uses several job slots **on the same node**.

MPI job Job with distributed memory parallelization, realized with MPI. Can use several job slots on several nodes and needs to be started with a helper program, e.g., `mpirun.lsf`.

Queue Label applied to a job to indicate its general requirements and intended execution nodes.

bsub: Submitting Job to the Cluser

- bsub submits information on your job to the batch system
  - ➥ What is to be done? (path to your program and required parameters)
  - ➥ What are its requirements? (for example queue, number of tasks, maximum runtime)
- LSF matches the job's requirements against the capabilities of available job slots
- When suitable job slots are found the job is started
- LSF considers jobs to be started in the order of their priority

Available Queues

- Base queues
  - mpi: General purpose queue, especially well suited for large MPI jobs. Up to 1024 tasks, up to 48 hours runtime.
  - fat: For SMP jobs. Up to 512 GB in one host. Otherwise as `mpi`.
- Queue suffixes
  - -long: Maximum runtime increased to 120 hours. Limited job slot availability.
  - -short: Maximum runtime decreased to two hours. Higher base priority, but also limited job slot availability.
- Special purpose queues
  - fat+: For extreme memory requirements. Up to 2048 GB in one host. Up to 120 hours, max 40 tasks.
  - gpu: For jobs using GPGPU acceleration.
  - int: For interactive jobs, i.e., jobs which require a shell or a GUI.

bsub: Usage 1

```
bsub <bsub options> [mpirun.lsf] <path to program> <program parameters>
```

bsub options for serial jobs

-q <queue> Submission queue.

-W <hh:mm> Maximum runtime. If this is exceeded the job is killed.

-o <file> Store job output in file (otherwise sent by email). %J in the filename stands for the jobid.

-N Get notification email (when using -o option).

Download examples


```
http://wwwuser.gwdg.de/~cboehme1/pkurs/pkurs.tar.bz2
```

bsub: Using Job Scripts

A job script is a shell script with a special comment section.

bsub: Basic job script example

```
#!/bin/sh
#BSUB -q mpi
#BSUB -W 00:10
#BSUB -o out.%J

/bin/hostname
```

Submit with:

```
bsub < <script name>
```

bsub: Usage 2

bsub: Interactive jobs

-ISs Starts an interactive job with ssh (`S`) and shell support (`s`)

-XF Adds X11 (GUI) forwarding. This requires that you connect to the frontend with `ssh -Y` and your local machine supports X-Windows.

-q int Use the interactive queue. In `int` the nodes have no slot limit. They will take jobs until their load crosses a specified threshold, so jobs start immediately.

Interactive Job Example

Running Matlab

```
> ssh -Y gwdu101.gwdg.de
> module load matlab/2015a
> bsub -ISs -XF -q int matlab
```

- The job will be dispatched and as soon as an available node is found and the Matlab interface will start.
- If you have your own license for Matlab then you need to place your `license.lic` file in `$HOME/.matlab/R2015a_licenses` directory (dependent on the version you are using).

bsub: Usage 3

bsub options for parallel (SMP or MPI) jobs.

-n <min>,<max> Minimum and maximum process count. You can also specify the exact number.

-a <wrapper> This option denotes a wrapper script required to run SMP or MPI jobs. The most important wrappers are `openmp`, `intelmpi`, and `openmpi`.

mpirun.lsf LSF's MPI helper program. Needs to be put in front of the program path in MPI jobs.

bsub: Usage 4

bsub: Specifying process distribution with -R

-R span[hosts=1] Put all processes on **one** host. You always want to use this with SMP jobs.

-R span[ptile=<x>] `x` denotes the exact number of job slots to be used on each host. If the total process number is not divisible by `x` the residual processes will be put on one host.

Recipe: Submitting an MPI job

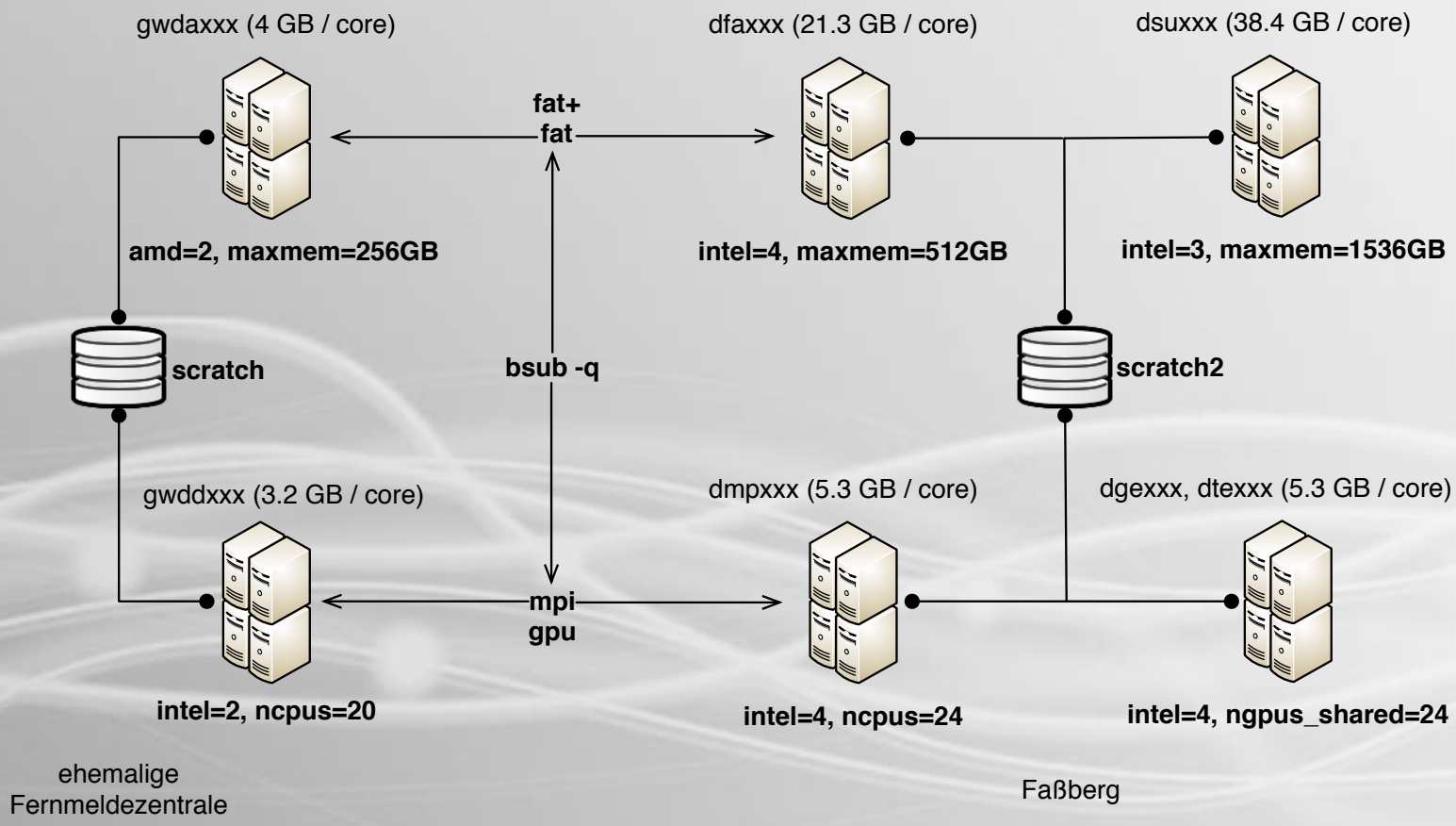
Using only complete nodes in the `mpi` queue

```
#BSUB -q mpi
#BSUB -n 240
#BSUB -R span[ptile='!']
#BSUB -R same[model]
#BSUB -a intelmpi
#BSUB -o out.%J

module purge
module load intel/compiler intel/mkl intel/mpi namd

mpirun.lsf namd2 +setcpuaffinity apoa1.namd
```

The GWDG Scientific Compute Cluster



Job Disk Space Usage Options

/local Local hard disk of the node. SSD based on almost all nodes, therefore a very fast option for storing temporary data. Automatic file deletion.

/scratch Shared scratch space, available on most nodes, but there are two instances (use `-R scratch` or `-R scratch2`). Very fast, no automatic file deletion, but also no backup! Files may have to be deleted manually when we run out of space.

$HOME Available everywhere, permanent, with backup. Personal disk space can be increased. Comparably slow.

bsub: Usage 5

bsub: Specifying node properties with -R

-R scratch Access to shared `/scratch`.

-R scratch2 Access to shared `/scratch2`.

-R ncpus=<x>
: Choose nodes by available cores (usually equals job slots). Useful with `span[ptile=<x>]`.

-R "maxmem><x>"
: Choose nodes with more than `x` MB equipped memory. Allow for rounding and unit issues.

-R "maxmem/ncpus><x>"
: Choose nodes with more than `x` MB memory per core (it's possible to use basic math in a `-R` statement).

Recipe: Using /scratch

Running Gaussian09 using /scratch for temporary files

```
#!/bin/sh
#BSUB -q fat
#BSUB -n 64
#BSUB -R "span[hosts=1]"
#BSUB -R scratch
#BSUB -W 24:00
#BSUB -C 0
#BSUB -a openmp

export g09root="/usr/product/gaussian/g09/d01"
. $g09root/g09/bsd/g09.profile

MYSCRATCH=`mktemp -d /scratch/${USER}/g09.XXXXXXXX`
if [ ${MYSCRATCH} -a -d ${MYSCRATCH} ]; then export GAUSS_SCRDIR=${MYSCRATCH}
else export GAUSS_SCRDIR=/local; fi

g09 myjob.com myjob.log

if [ ${MYSCRATCH} -a -d ${MYSCRATCH} ]; then rm -rf ${MYSCRATCH}; fi
```

General remark on resources

The difference between resource specification and reservation

- Generally compute host and their resources are *shared*.
- Resource requirements specify properties your execution hosts (or job slots) must have, but they **don't reserve them**.
- The only ways to *reserve* resources are `-n`, `-x`, and `ngpus_shared`.
- Usually one should try to use hosts non-shared.

Job Memory Usage

- LSF will try to fill up each node up to its job slot limit (normally equals `ncpus`).
- Therefore each job slot must not use more memory than available per core (i.e. `maxmem/ncpus`)!
- If you need more memory than `n * maxmem/ncpus`, you have to add more job slots (i.e. increase `n`).
- If your jobs memory usage increases with the number of job slots, you have to leave additional slots *empty*.
  - ➥ Trivial for serial and SMP jobs
  - ➥ You can also use exclusive jobs

Job slots on an dmp0xx node

dmp0xx
maxmem = 127.8 GB
ncpus = 24
slots = 24

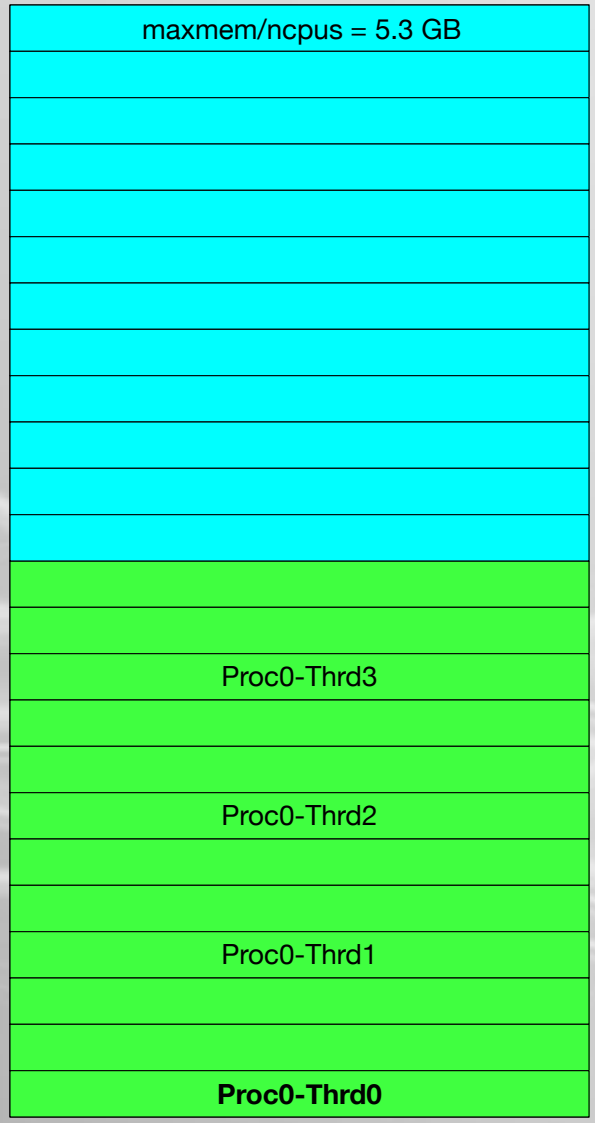
Recipe: Reserving Memory for OpenMP

Using empty job slots to reserve memory for OpenMP jobs

```
#!/bin/sh
#BSUB -q fat
#BSUB -W 00:10
#BSUB -o out.%J
#BSUB -n 12
#BSUB -R "maxmem/ncpus>5000"
#BSUB -R span[hosts=1]
#BSUB -a openmp

export OMP_NUM_THREADS=4
./myopenmpprog
```

Job slots with OpenMP job



Other users' job slots

Reserved Slots

dmp0xx
maxmem = 127.8 GB
ncpus = 24
slots = 24

Exclusive jobs

- `#BSUB -x` in a job script denotes an exclusive job.
- An exclusive job uses all job slots (cores) of all its nodes.
- This can be used to request all memory (`maxmem`) of all requested nodes without thinking about `n`.
- Using `-x` together with `-R span[hosts=1]` reserves one complete node, independent of `-n`.
- Disadvantage: Jobs with many nodes may wait longer, compared to those with exact `-n`.

Recipe: MPI jobs with -x

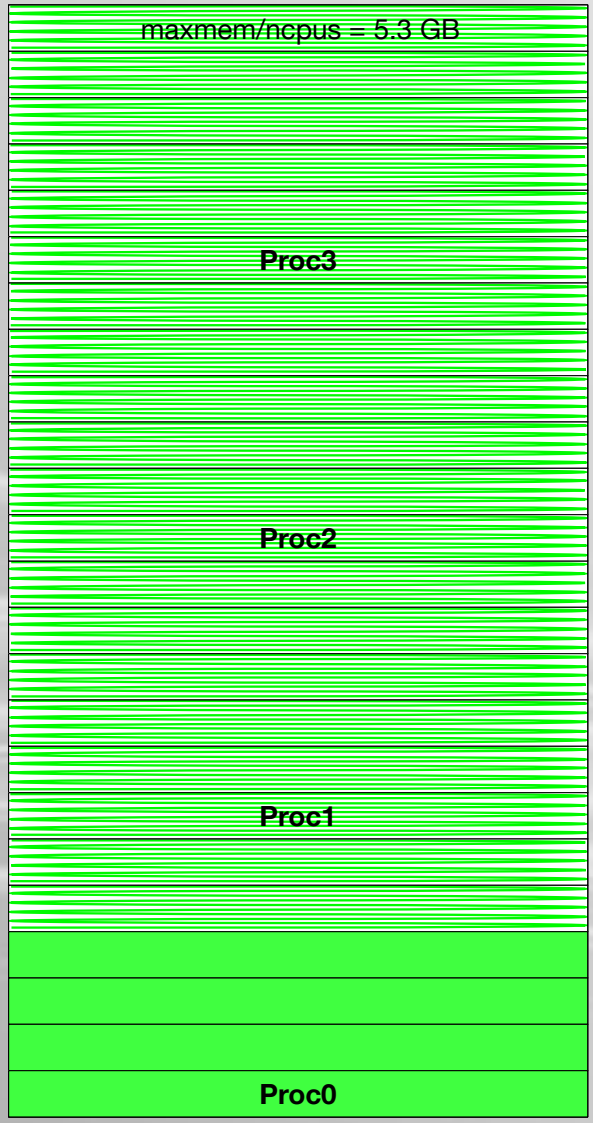
Using exclusive jobs to reserve memory for MPI jobs

```
#BSUB -q mpi
#BSUB -n 16
#BSUB -R span[ptile=4]
#BSUB -R "maxmem>120000"
#BSUB -R "ncpus>3"
#BSUB -a intelmpi
#BSUB -o out.%J
#BSUB -x

module purge
module load intel/compiler intel/mpi

mpirun.lsf big_mpi
```

Job slots with exclusive job



Blocked due to -x

Reserved Slots

dmp0xx
maxmem = 127.8 GB
ncpus = 24
slots = 24

x 4

Adding Job Slots to non-exclusive MPI Jobs

- Requires an MPI helper that is LSF aware but can choose an independent process distribution
- Currently not supported by mpirun.lsf!
- Supported by OpenMPI
- Requires some shell scripting with Intel-MPI
- Use cases:
  - ➥ Reserving slots for more memory per MPI process without increasing wait time
  - ➥ Running more than one thread per MPI process (hybrid jobs)

Recipe: Adding Slots to an OpenMPI Job

Running hybrid jobs

```
#BSUB -q mpi
#BSUB -n 360
#BSUB -R span[ptile=24]
#BSUB -R ncpus=24
#BSUB -a openmpi
#BSUB -o out.%J

module purge
module load openmpi/gcc

export OMP_NUM_THREADS=6

mpiexec -n 60 -npernode 4 hybrid_job
```

Job slots with hybrid job

Reserved Slots

dmp0xx
maxmem = 127.8 GB
ncpus = 24
slots = 24

x 15

Recipe: Adding Slots to an Intel-MPI Job

Running hybrid jobs

```
#BSUB -q mpi
#BSUB -n 360
#BSUB -R span[ptile=24]
#BSUB -R ncpus=24
#BSUB -a intelmpi
#BSUB -o out.%J

module purge
module load intel/compiler intel/mpi

export OMP_NUM_THREADS=6

HOSTS=$(echo "$LSB_HOSTS" | sed "s/ /\n/g" | sort | uniq | xargs)

mpiexec.hydra -n 60 -hosts ${HOSTS// /,} -perhost 4 hybrid_job
```

bsub: Usage 6

bsub: Specifying CPU architecture

-R x64inlevel=1
: Request CPU with AVX support.

-R x64inlevel=2
: Request CPU with AVX2 (and AVX) support.

-R amd=<x>
: Choose AMD CPU (1: Interlagos, 2: Abu Dhabi)

-R intel=<x>
: Choose Intel CPU (1: Sandy Bridge, 2: Ivy Bridge, 3: Haswell, 4: Broadwell)

Using the gpu queue

- All jobs in the `gpu` queue **must** use GPUs.
- The `ngpus_shared` property of a GPU node is equal to its `ncpus` value, and denotes virtual shares of the GPU.
- You must set `x` in `-R ngpus_shared=<x>` at least to 1, in order to get access to a GPU.
- GPU sharing is supported, but not recommended. Use `-x` to get a GPU node exclusively.
- You can choose to get a double precision GPU with `-R tesla`.
- You can choose a node equipped with two GPUs with `-R ngpus=2`, or with four GPUs with `-R ngpus=4`.

Using the `fat+` queue

- All jobs **must** use more than 250 GB memory.
- The queue provides nodes with 512 GB, 1.5 TB, and one node with 2 TB.
- All jobs must use at least a full 512 GB node or half a 1.5 TB or 2 TB node.
- For a full 512 GB node:

```
#BSUB -x
#BSUB -R "maxmem < 600000"
```

- For half a 1.5 TB node (your job needs more than 500 GB RAM):

```
#BSUB -n 20
#BSUB -R span[hosts=1]
#BSUB -R "maxmem < 1600000 && maxmem > 600000"
```

Recipe: Iterators in R

Using the foreach package

```
library(foreach)

ls<-foreach(i=1:1e3) %do% {
      norm=rnorm(1e6)
      summ=summary(norm)
      summ
      }
ls
```

Recipe: Parallelization in R with doMPI

Using doMPI as backend for foreach

```
library(doMPI)

cl <- startMPIcluster()
registerDoMPI(cl)

ls<-foreach(i=1:1e3) %dopar% {
      norm=rnorm(1e6)
      summ=summary(norm)
      summ
      }
ls

closeCluster(cl)
mpi.quit()
```

Recipe: MPI and R

Using R with doMPI in a batch job

```
#BSUB -q mpi
#BSUB -n 20
#BSUB -a openmpi
#BSUB -o out.%J

module load openmpi/gcc

mpirun.lsf Rscript "doMPI_script.R"
```

Task parallelization with GNU `parallel`

- GNU `parallel` distributes a set of tasks to a set of cores
- Requirement: No dependencies and side effects between tasks (*embarrassingly parallel*)

Using `parallel` to run a program with multiple input files

```
parallel 'cp {} .; g09 {/} {/.}.log' \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)

parallel 'cp {} .; if (eval "g09 {/} {/.}.log");
then echo {/} >> ok; else echo {/} >> failed; fi' \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)
```

Recipe: GNU parallel in a batch job

Multiple input files with parallel in a batch job

```
#!/bin/bash

#BSUB -q mpi-short
#BSUB -x
#BSUB -W 02:00
#BSUB -R "scratch||scratch2"

module load gaussian
mkdir /scratch/${USER}/g09_ptest
cd /scratch/${USER}/g09_ptest

parallel \
  'cp {} .;
  if (eval "g09 {/} {/.}.log");
    then echo {/} >> ok;
    else echo {/} >> failed;
  fi' \
::: $(find /usr/product/gaussian/g09/tests -name *.com -type f)
```

Miscallaneous LSF Commands

GWDG

bjobs	Lists current jobs. Useful options: `-p1`, `-l`, `-a`, `<jobid>`, `-u all`, `-q <queue>`, `-m <host>`.
bpeek	Watch output of running job.
bhist	Lists older jobs. Useful options: `-l`, `-n`, `<jobid>`.
lsload	Status of cluster nodes. Useful options: `-l`, `<hostname>`.
bqueues	Status of batch queues. Useful options: `-l`, `<queue>`.
bhpart	Why do I have to wait? Priorities. Useful options: `-r`, `<host partition>`.

`bkill`: The Final Command

- Two use modes:
  1. `bkill <jobid>`: Kill job with specific jobid.
  2. `bkill <select options> 0`: Kill all jobs fitting the selection.
     - Select option examples: `-q <queue>`, `-m <host>`.

Section 6

Getting Help

Information sources

- man pages
- LSF online help
  - ➥ For example: `bsub -help`
- GWDG scientific compute cluster documentation
  - ➥ `https://info.gwdg.de/docs/doku.php?id=en:services:application_services:high_performance_computing:start`
- GWDG scientific compute cluster user wiki
  - ➥ `https://info.gwdg.de/wiki/doku.php?id=wiki:hpc:start`
- Cluster status page
  - ➥ `http://lsf.gwdg.de/lsfinfo/`
- HPC announce mailing list
  - ➥ `https://listserv.gwdg.de/mailman/listinfo/hpc-announce`

Adding to the Wiki

- Everyone with a cluster account can add to the Wiki!
- Please inform us of all changes and new articles at parallel@gwdg.de.
- Please add the category "*Scientific Computing*" to all contributions regarding the cluster.

Using the GWDG Support Ticket System

- Write an email to *hpc@gwdg.de*
- State your user id (`$USER`)
- If you have a problem with your jobs **always send the complete standard output and error**!
- If you have a lot of failed jobs send at least two outputs. You may also list the jobid's of all failed jobs.
- If you don't mind us looking at your files, please state this in your request
  - ➥ You may limit your permission to specific directories or files

Digression: Directory Structure 1

- Convention: Executables are stored in "`bin`", shared libraries in "`lib`" directories
- Directories in "`$PATH`" are searched for binaries, directories in "`$LD_LIBRARY_PATH`" for libraries
- Two strategies:
  1. Put everything directly under `$HOME/bin`, `$HOME/lib`
     - Easy to setup search paths
     - Difficult to remove software packages
  2. Put each software into its own subdirectory
     - Easy to remove software (with "`rm -rf` <subdirectory>")
     - Difficult to setup search paths

Digression: Directory Structure 2

- Or combine both strategies:
  - Put each software in its own subdirectory
  - Use "`ln -s`" to link everything to $HOME/bin and $HOME/lib, respectively
  - Use "`export LD_LIBRARY_PATH=$HOME/lib:$LD_LIBRARY_PATH; export PATH=$HOME/bin:$PATH`" in your shell and scripts
  - Use "`find $HOME/bin $HOME/lib -xtype l -delete`" after removing software