

Workflow Changes

"The Angry Penguin", used under creative commons licence from Swantje Hess and Jannis Pohlmann.



“Everything is a file”


Unix Philosophy

- One of the key ideas of *nix type operating systems (including Linux) is "everything is a file"
- If you open the "file" /proc/cpuinfo then you will get information about the CPUs installed in that computer
 - This **isn't** a file that is automatically created and you are just reading - the information is generated when you access it. Look at the CPU frequency - you may see it changing every time you look
- Even though it looks like a file and you can read it like a file it is completely virtual, in fact everything in /proc is not real files
 - https://en.wikipedia.org/wiki/Everything_is_a_file
- Several of the things that we are going to talk about here rely on this!

/proc/cpuinfo

```
processor : 127
vendor_id : AuthenticAMD
cpu family : 23
model : 49
model name : AMD EPYC 7742 64-Core Processor
stepping : 0
microcode : 0x830107b
cpu MHz : 3227.776
cache size : 512 KB
physical id : 1
siblings : 64
core id : 63
cpu cores : 64
apicid : 127
initial apicid: 127
fpu : yes
fpu_exception : yes
cpuid level : 16
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx
mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf pni pclmulqdq monitor ssse3
fma cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a
misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3
hw_pstate sme ssbd mba sev ibrs ibpb stibp vmmcall sev_es fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb
sha_ni xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr wbnoinvd
amd_ppin arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic
v_vmsave_vmload vgif umip rdpid overflow_recov succor smca
bugs : sysret_ss_attrs spectre_v1 spectre_v2 spec_store_bypass
bogomips : 4472.83
TLB size : 3072 4K pages
clflush size : 64
cache_alignment : 64
address sizes : 43 bits physical, 48 bits virtual
power management: ts ttp tm hwpstate cpb eff_freq_ro [13] [14]
```

Using non-shared filesystems



/tmp

- One way to avoid stressing the shared filesystem is to not use it until you have to
- On each node of Sulis there is a /tmp directory that is local to the node that you are running on
 - Writing to **it** is just like writing to a local disk on any computer
- Stuff in /tmp is only accessible while your job is running
- You can generate your files there and then combine them with something like **tar** or **zip** and then copy them to the shared filesystem as a single file

/tmp

- There is about 600GB of space on /tmp on each node of Sulis
- When a job starts, /tmp is cleaned up you **must** get data off before your job finishes
- This may mean that you have to be careful with requesting walltime
- It is also polite to delete your files from /tmp rather than relying on the automatic systems

/dev/shm

- If you need really fast access, low latency access then you can use /dev/shm
- This is one of those virtual filesystems that I mentioned, but it represents a filesystem in the computer's memory (note, memory as opposed to storage)
- It isn't quite as fast as actually accessing data in memory like normal, but it is much faster than any disk
 - You can read and write data to it like you could normal files, and it is an easy way of sharing data between programs quickly
- You have up to 256GB on Sulis nodes (out of 512GB of total memory), but be careful that you don't run out of memory for your actual program - it is the same memory that your program is using to run!
- You have to copy data out of /dev/shm before your job finishes and it is cleaned up before your job starts

Example Script

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=3850
#SBATCH --time=08:00:00
#SBATCH --account=suxxx-somebudget

module purge
module load GCC/10.2.0

#Make a temporary directory for the output
mkdir /tmp/$SLURM_JOB_ID
#Note this assumes that your program takes an output directory as an
argument
srun ./a.out /tmp/$SLURM_JOB_ID
#Compress the output using tar-gzip
#Compress directly to the shared filesystem
tar cvzf $SLURM_JOB_ID.tgz /tmp/$SLURM_JOB_ID
rm -rf /tmp/$SLURM_JOB_ID
```

- So long as your job finishes in under 8 hours this will tar-gzip your data and copy it to wherever you launched the script from

slurm sbcast

- Sometimes what you want is not to write data but to read it
- You can read from the shared filesystem, but same problem
- SLURM, the scheduler on SULIS has a built in tool to do it **sbcast**
- Simply put **sbcast {src} {dest}** and the file will be copied out to the nodes automatically and efficiently
- Obviously, it has to go to a local filesystem like /tmp or /dev/shm or there's no point!

Example Script

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=3850
#SBATCH --time=08:00:00
#SBATCH --account=su950

mkdir /tmp/$SLURM_JOB_ID
sbcast ~/FortArray.h5 /tmp/$SLURM_JOB_ID/FortArray.h5
ls /tmp/$SLURM_JOB_ID
```

- This is a trivial example
 - Copy a file and then list the directory to show that it is there
- **You have to specify the destination filename for sbcast**

Shared files

Shared Directories

- There are some datasets that are needed by multiple researchers
- These datasets can be large and contain many files
- It is better to have just one copy of these datasets that all researchers using them access
- Ask the support team about shared file space

Using virtual filesystems



Using HDF5 as a filesystem

- You might have spotted that the diagram of an HDF5 file looks rather like directories and files in a filesystem
- This is because this kind of hierarchical structure is a natural way of structuring data
- It does mean that we can use an HDF5 file as a file system
 - Not particularly good as a read/write filesystem, but works very well for just reading

FUSE

- Normally filesystems are mounted and unmounted by the system administrator
- FUSE (filesystem in userspace) is designed to allow normal users to mount certain types of filesystem
- We have written a FUSE mount for HDF5 that allows you to view the contents of HDF5 files as if the groups are files and datasets are directories
 - Deployed as a module on Sulis
- This is a local thing that we have written, although the source code is available

toHDF5

```
[phsgbd@node010(sulis) epoch]$ ls epoch3d  
Data Makefile Start.pro example_decks src tests unpack_source_from_restart
```

- This is a directory for a code that we have worked on called EPOCH
- It is just a directory, nothing special about it
- We want to pack it up so that we can access the data in it from an HDF5 file

toHDF5

```
[phsgbd@node010(sulis) epoch]$ toHDF5 epoch3d/  
  
toHDF5 version 0.1.0  
=====
```

Path = /gpfs/home/p/phsgbd/epoch/epoch3d
Creating new file epoch3d.h5
--Creating group epoch3d
----Creating group tests
-----Creating group custom_stencils
-----Creating dataset makefile
-----Creating group optimized_xaxis_soft
-----Creating dataset makefile
-----Creating dataset input.deck
...

- We use the toHDF5 program from our virtual filesystem module to pack the data into an HDF5 file. It tells you what it is doing as it runs

toHDF5

```
[phsgbd@node010(sulis) epoch]$ h5ls -r epoch3d.h5
/                               Group
/epoch3d                       Group
/epoch3d/.gitignore            Dataset {203}
/epoch3d/Data                  Group
/epoch3d/Data/.gitignore       Dataset {42}
/epoch3d/Makefile              Dataset {19650}
/epoch3d/Start.pro             Dataset {95}
/epoch3d/example_decks         Group
/epoch3d/example_decks/bremsstrahlung.deck Dataset {4966}
/epoch3d/example_decks/cone.deck Dataset {2698}
/epoch3d/example_decks/filter.deck Dataset {1837}
/epoch3d/example_decks/injectors.deck Dataset {2028}
/epoch3d/example_decks/power_law.deck Dataset {1646}
/epoch3d/example_decks/qed_rese.deck Dataset {6455}
/epoch3d/example_decks/window.deck Dataset {1576}
/epoch3d/src                   Group
/epoch3d/src/boundary.F90      Dataset {93686}
/epoch3d/src/constants.F90     Dataset {27247}
...
```

- h5ls is a part of the HDF5 library - it lets you see what is in an HDF5 file

toHDF5

```
[phsgbd@node010(sulis) epoch]$ mkdir /tmp/mnt
[phsgbd@node010(sulis) epoch]$ h5vfs epoch3d.h5 /tmp/mnt/
[phsgbd@node010(sulis) epoch]$ ls /tmp/mnt/
epoch3d
[phsgbd@node010(sulis) epoch]$ ls /tmp/mnt/epoch3d/
Data Makefile Start.pro example_decks src tests unpack_source_from_restart
[phsgbd@node010(sulis) epoch]$ ls /tmp/mnt/epoch3d/example_decks/
bremsstrahlung.deck cone.deck filter.deck injectors.deck power_law.deck
qed_rese.deck window.deck
```

- Now I use the h5vfs program to mount the HDF5 file as if it was a directory.
- I specify the HDF5 file to mount and the mount point (i.e. the directory that will have the new filesystem mounted into it). **THIS MUST BE ON /tmp!**
- Now I can use normal tools like "ls" to see the groups in the HDF5 file

toHDF5

```
[phsgbd@node010(sulis) epoch]$ cat /tmp/mnt/epoch3d/example_decks/cone.deck
begin:control
  nx = 250
  ny = 250
  nz = 250
  nparticles = nx * ny * nz * 1.123

  # Final time of simulation
  t_end = 50 * femto

  # Size of domain
  x_min = -10 * micron
  x_max = -x_min
  y_min = x_min
  y_max = x_max
  z_min = x_min
  z_max = x_max
end:control
...
```

- I can use any normal tools to interact with files, browse the directories as normal etc. Only limitation is that it is read only

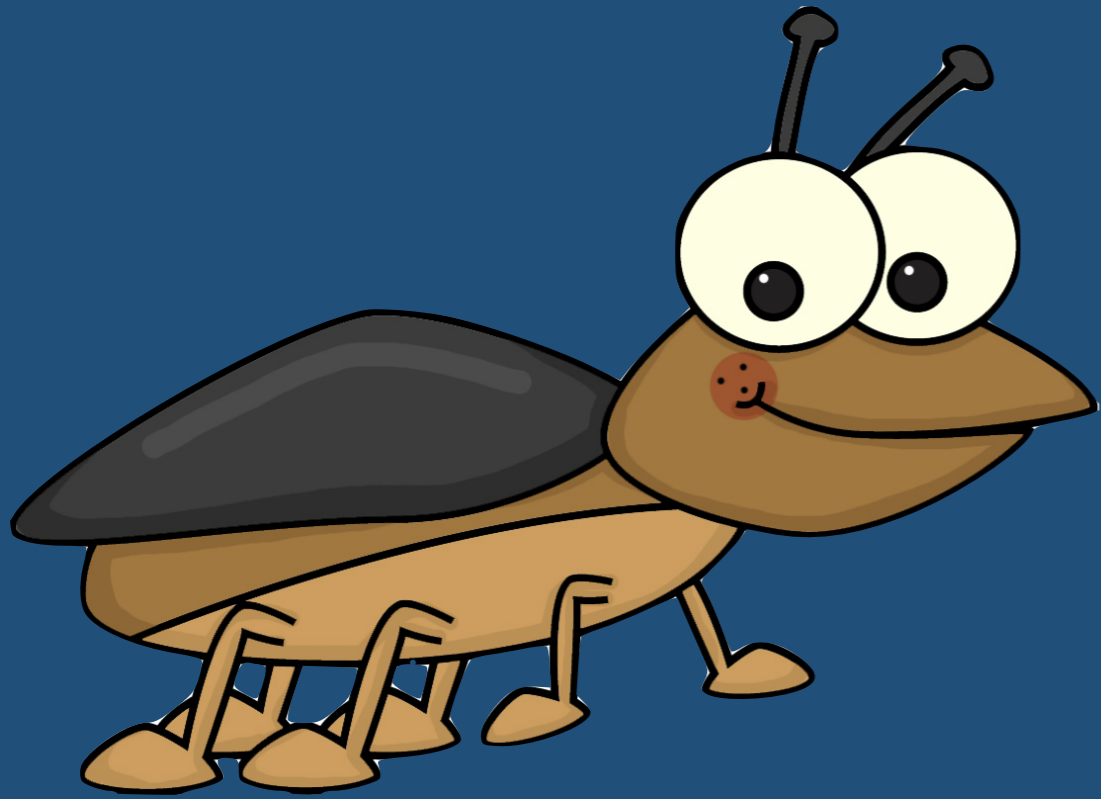
toHDF5

```
[phsgbd@node010(sulis) epoch]$ fusermount -u /tmp/mnt/  
[phsgbd@node010(sulis) epoch]$ ls /tmp/mnt/  
[phsgbd@node010(sulis) epoch]$
```

- When I am done, I can unmount the filesystem using
 - **fusermount -u** {mountpoint}
- Remember that even though you have mounted the file on /tmp, your actual data is still wherever you put the HDF5 file!

Using HDF5 as a filesystem

- This means that you can use HDF5 files without having to modify your code at all **for input**
- This is the quickest way of converting certain types of problem such as machine learning from multiple files to avoid having to use large numbers of files
- For read only access, you can combine this with a shared filespace - one HDF5 file mounted by different researchers



The End