Parallel Processing and Load Balancing

Robert Bukowski
Institute of Biotechnology
Bioinformatics Facility
(aka Computational Biology Service Unit - **CBSU**)

Workshop website: https://biohpc.cornell.edu/ww/1/Default.aspx?wid=136

Contact: brc_bioinformatics@cornell.edu

Motivation

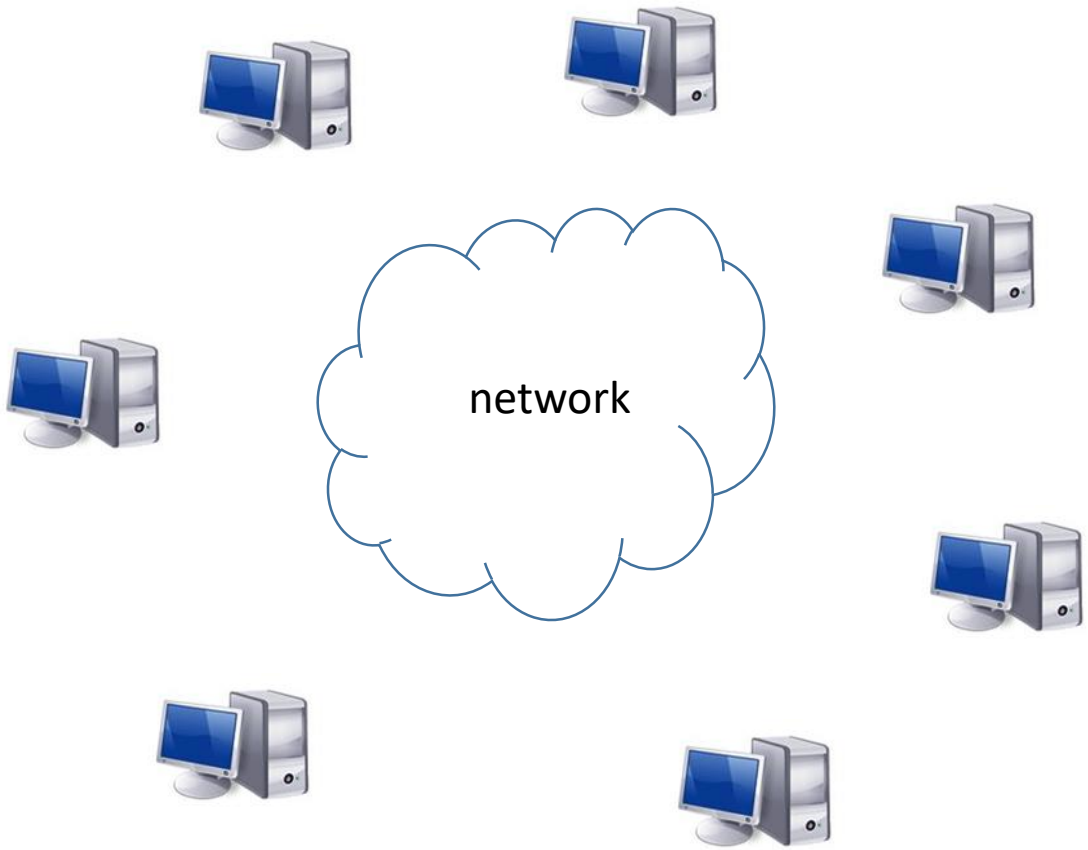
- Can you solve a ‘big task’ on your laptop? Not really…
  - too small: not enough memory, not enough disk to store big data
  - too slow: analysis would take forever
- You need a more powerful resource
  - bigger: more memory, more disk
  - FASTER!!!
- What does FASTER mean?
  - faster processor (and other hardware) – yes, but first of all….
  - MORE processors !!!
  - knowledge how to use it all

BioHPC renatal resources

Server type	#servers	#cores	RAM [GB]
interactive	4	4	24
general	32	8	16
medium gen1	1	16	64
	16	24	128
medium gen2	12	40	256
large gen1	8	64	512
large gen2	2	96	512
	4	112	512
	2	80	512
	3	88	512
extra-large	1	64	1,024
	1	112	1,024
	1	88	1,024
GPU gen2	2	32	256
Total	89	3,056	19,104

← your workshop machines

Big picture



Given:

- ❑ ‘big task’ at hand
- ❑ multiple CPUs, RAM, and disk storage, possibly scattered across multiple networked computers

Objective:

- ❑ Parallelize: solve the ‘big task’ in time shorter than it would take using a single CPU on a single computer
- ❑ Balance load: keep resources busy, but not overloaded

Synopsis

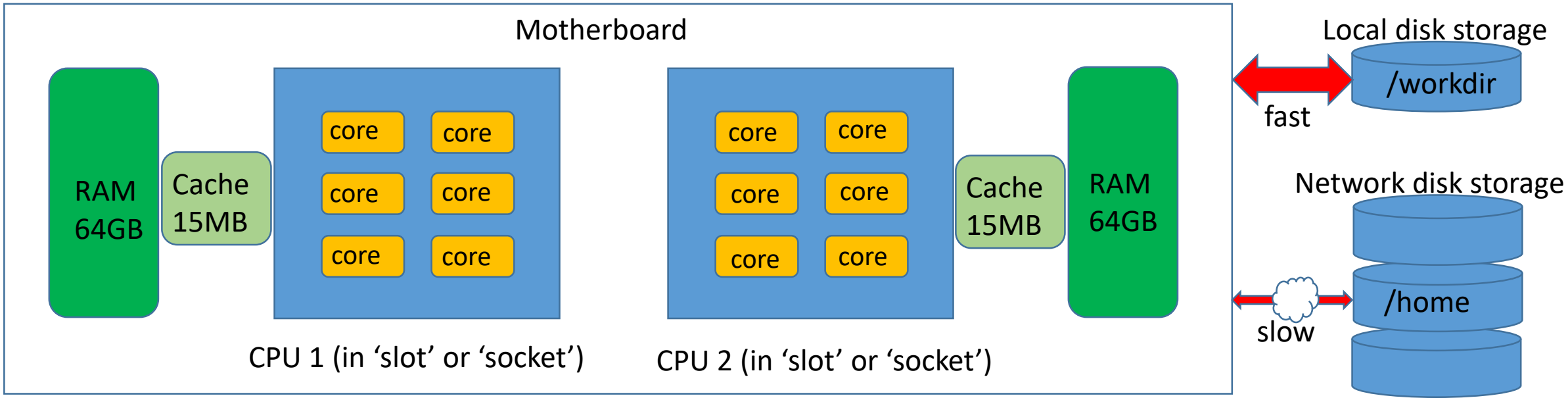
- ❑ Some basic hardware facts
- ❑ Some basic software facts
- ❑ Parallelization: problems and tools
- ❑ Monitoring and timing Linux processes
- ❑ Multiple independent tasks
  - Load balancing

Next week:

- ❑ Advanced load balancing using job scheduler (SLURM = Simple Linux Utility for Resource Management)

Hands-on exercises: will introduce some tools and techniques (although quantitative conclusions doubtful in shared environment…)

Resources on a single machine (here: cbsumm12)



CPU: an integrated circuit (a "chip") containing computational hardware. May be more than one per server, typically 2-4.

Core: a subunit of CPU capable of executing an independent sequence of instructions (a **thread**). Shares communication infrastructure and internal memory with other cores on the CPU.

threads possible to <u>run</u> at the same time = # cores

Hyperthreading (HT): technology to simultaneously run several (typically – two) independent sequences of instructions (**threads**) on each core, sharing the core's hardware; may be <u>disabled</u> or <u>enabled</u>.

If HT enabled, **core** is understood as **hyperthreaded core**

In this example, with HT enabled, **# cores =24**

RAM: memory. All accessible to all cores (but easier to access CPU's 'own' portion); **Cache**: fast-access (but small) memory 'close' to CPU

Check on CPU configuration with `lscpu`

```
[root@cbsumm12 ~]# lscpu

Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 45
Model name:            Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
Stepping:              7
CPU MHz:               2500.122
CPU max MHz:           2500.0000
CPU min MHz:           1200.0000
BogoMIPS:              4000.35
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              15360K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
Flags:                 fpu vme de pse tsc …
```

Hyper-threaded cores

Hyperthreading ON

CPUs

Check memory using `free`

show in Mbytes
(other options: -k -g)

```
[bukowski@cbsumm12 ~]$ free -m
              total        used        free      shared  buff/cache   available
Mem:         128738        1772      123220        1550        3746      124575
Swap:          4095         836        3259
```

Not always correct,
unfortunately…

Check disk storage using df

```
[root@cbsumm12 ~]# df -h
Filesystem                                          Size  Used Avail Use% Mounted on
/dev/mapper/rhel_cbsumm12-root                      300G   32G  269G  11% /
devtmpfs                                             63G     0   63G   0% /dev
tmpfs                                                63G     0   63G   0% /dev/shm
tmpfs                                                63G  2.1G   61G   4% /run
tmpfs                                                63G     0   63G   0%
/sys/fs/cgroup
/dev/md126                                          871G   72M  827G   1% /SSD
/dev/mapper/rhel_cbsumm12-home                      3.4T  130G  3.3T   4% /local
/dev/sda2                                           494M  144M  351M  30% /boot
tmpfs                                                13G   16K   13G   1% /run/user/42
tmpfs                                                13G     0   13G   0% /run/user/0
128.84.180.177@tcp1:128.84.180.176@tcp1:/lustre1    1.3P 1003T  300T  78% /home
cbsugfs1:/home                                      233T  135T   99T  58%
/glusterfs/home
tmpfs                                                13G     0   13G   0%
/run/user/4857
```

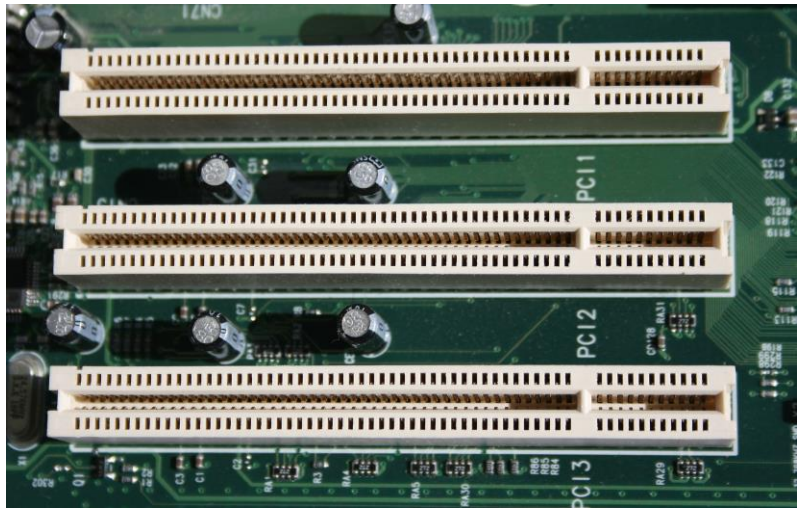
Local scratch space (fast, temporary)

Network-mounted (slow, permanent).
NO I/O-intensive computations there!

Check other hardware using `lspci`

PCI = **Peripheral Component Interconnect**

Most devices are attached this way



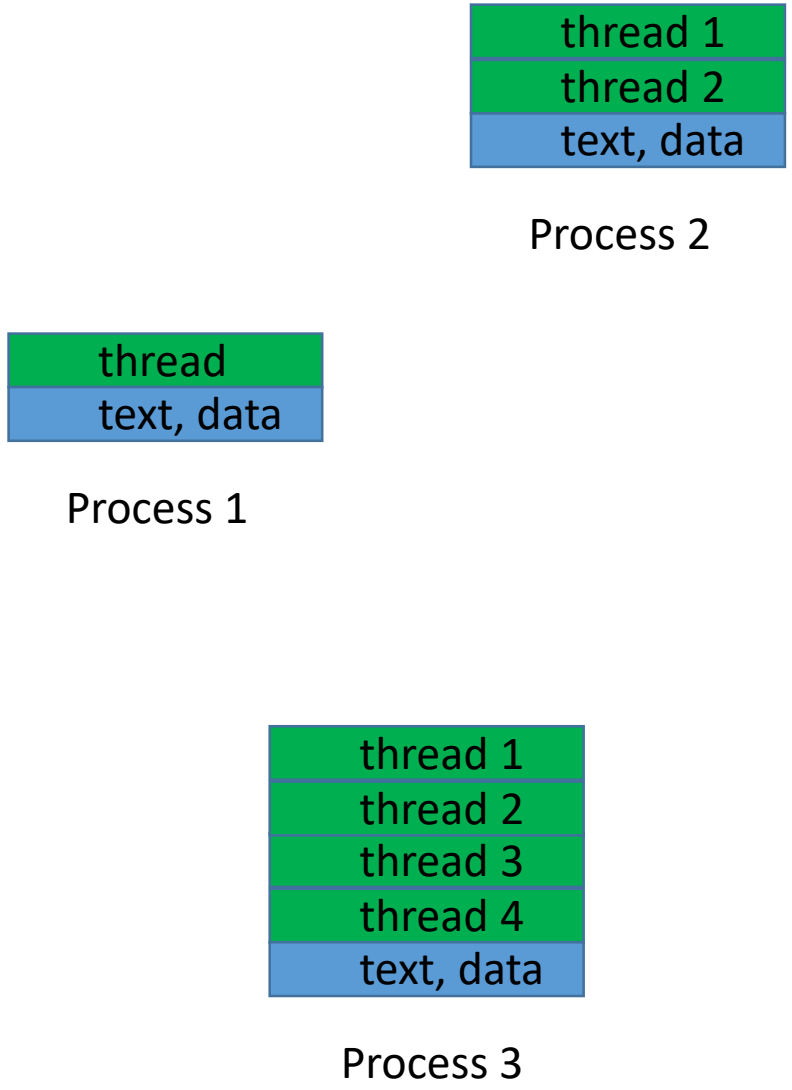
`lspci` produces long output, better paginate or filter, e.g.,

```
[root@cbsumm12 ~]# lspci | grep -i raid
00:1f.2 RAID bus controller: Intel Corporation C600/X79 series chipset SATA RAID
Controller (rev 06)

[cryosparc_user@cbsugpu03 ~]$ lspci | grep -i nvidia
02:00.0 3D controller: NVIDIA Corporation GP100GL [Tesla P100 PCIe 16GB] (rev a1)
83:00.0 3D controller: NVIDIA Corporation GP100GL [Tesla P100 PCIe 16GB] (rev a1)
```

What's running on a machine

- Everything 'running on a machine' (apps run by users, OS tasks) does this by means of **processes**
- **Process**: instruction sequence loaded into memory, to be executed by CPU cores, using some memory to store code (text) and data, communicating with peripherals (disk storage, network, …)
- Templates for processes stored on disk as **executable files**
- Process may contain **one or more threads** (multithreading), all with **access to the same data** (but not to data of other processes)
- Each process has a unique **process ID** (and so do individual threads)
- Each process is created by another process - its **parent** process (thus, there is a <u>process tree</u>)
- Each process (with all its threads) runs on a **single machine**



Cores and processes: mixing it all together

- At any given time, a **core** can be
  - executing one thread
  - idle

- At any given time, a **thread** can be
  - running on one of the cores
  - waiting off-core (for input or data from memory or disk, or for an available core)
  - stopped on purpose

- **Load**: number of threads running + waiting for a core to run on (<u>should not exceed number of cores</u>!)

- **Context switches**
  - a core executes one thread for a while, then switches to another (state of the previous one is saved to be resumed later)
  - some threads have higher priority (like quick house-keeping tasks by OS)
  - threads are only allowed to run for some time without being switched out
  - frequent context switches **not good for performance** (occur at high load)

- **Scheduler** (part of **Linux kernel**) takes care of distributing **threads** over **cores**
  - (not to be confused with <u>SLURM job scheduler</u> discussed in Part 2)

Software structure

User mode

User applications (bash, python, bwa, FireFox, ssh, VNC, blast+…)

Low-level system components (init, services, logind, networkd, X11,…)

Processes

C standard library (processes communicating with kernel)

Kernel mode

Linux kernel

system call interface (SCI) – used by processes
process scheduling
inter-process communication tools (IPC)
memory management
interface to hardware (drivers)

Hardware: CPUs, memory, disk storage, other peripherals

Cores and processes: mixing it all together

- Typically, there are many more threads than cores:

Example: <u>empty</u> (i.e., no users) machine **`cbsumm12`** (24 cores), some time last Saturday:

```
ps -ef | wc -l   :   596  (all processes)
ps -efL | wc -l  :   919  (all threads)
```

these are processes that keep the OS running
mostly waiting for stuff to help with, clean up, running only when needed
consume very few CPU cycles and little RAM

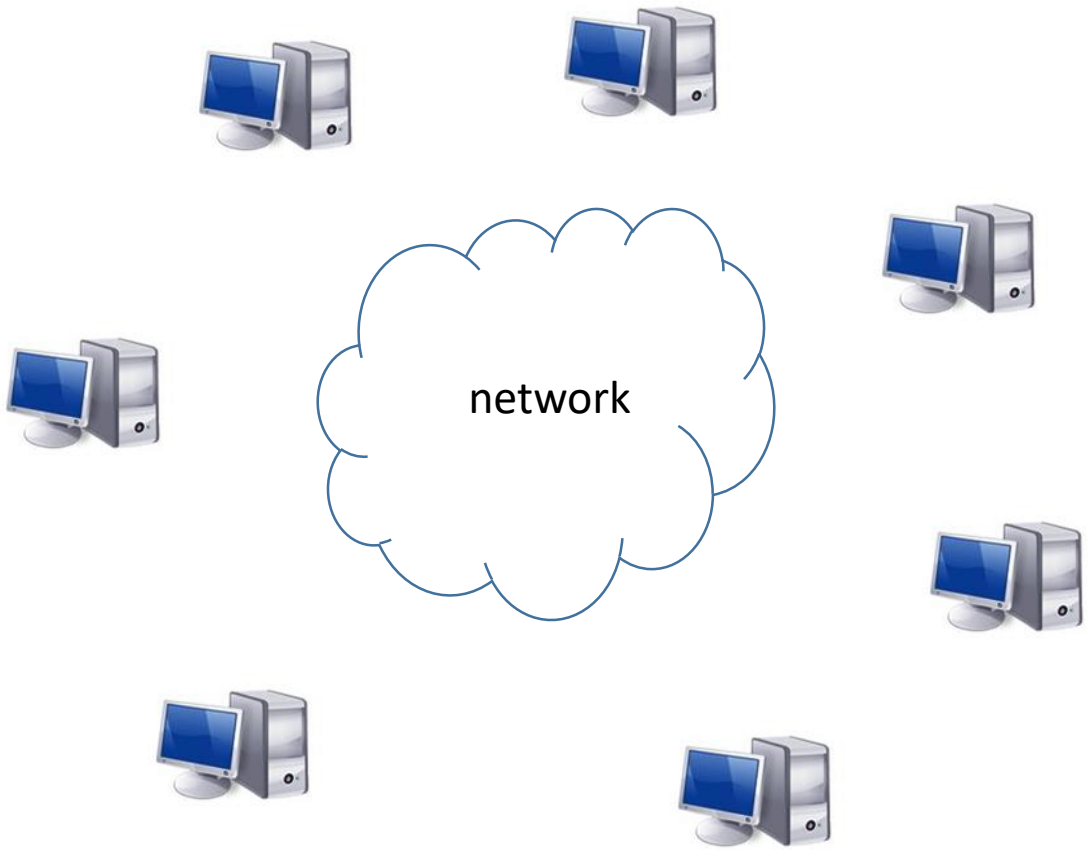
- Despite large number of threads, the **load** on the machine was very low, and most memory was available:

```
uptime
 10:37:32 up 265 days, 14:57,  3 users,  load average: 0.08, 0.21, 1.11


free -m
              total        used        free      shared  buff/cache   available
Mem:         128738        1772      123220        1550        3746      124575
```

Almost all CPU and memory resources up for grabs by users' programs

Big picture

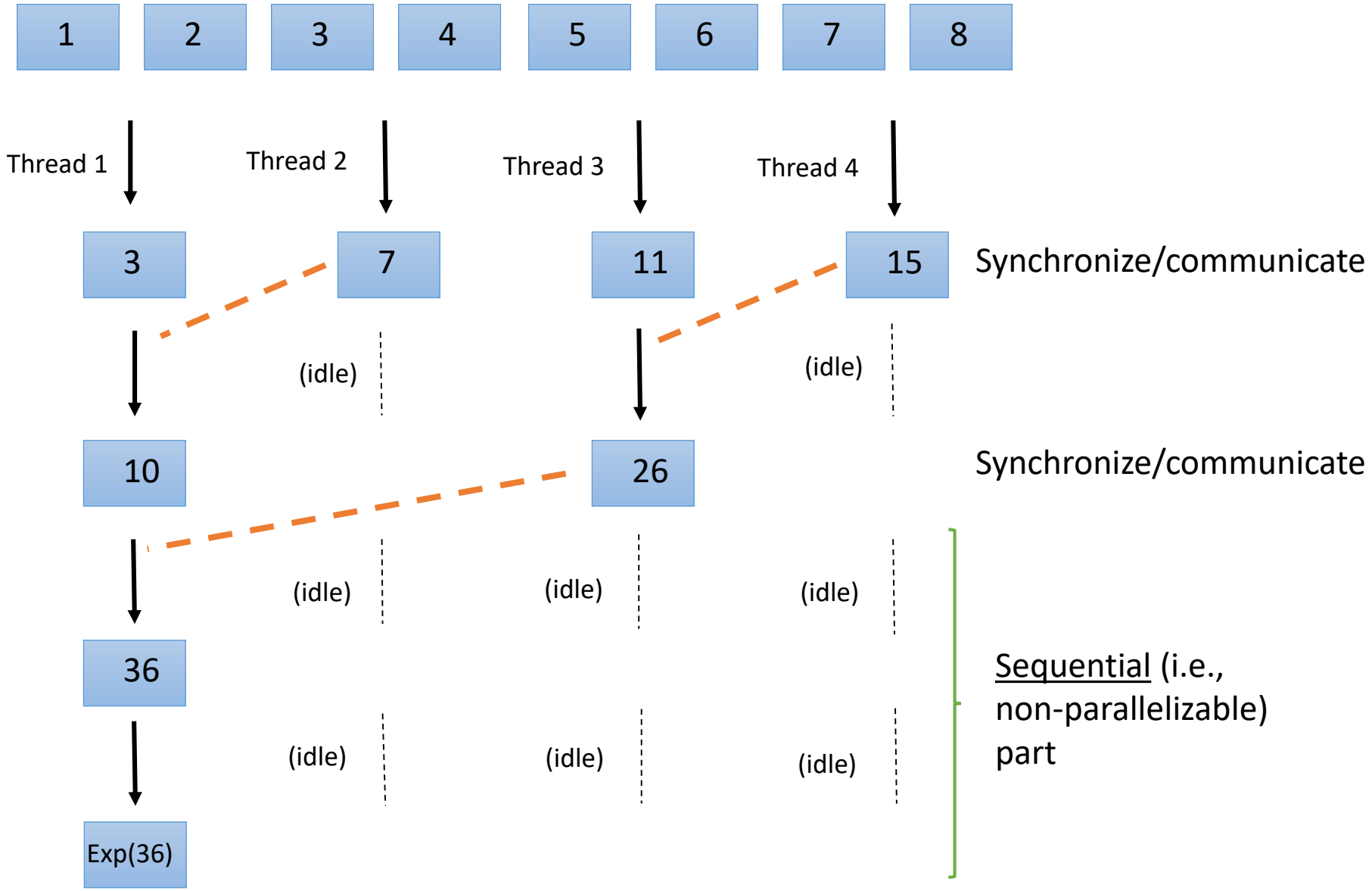


- Given a ‘big task’ at hand, make multiple CPU cores work in parallel to achieve the solution in time shorter than what would be needed if only a single core were used
- Constraints:
- CPUs and memory possibly scattered over multiple networked machines
- Core number and memory limits on individual machines
- A process (with all its threads and memory) can only run on one machine
- No direct data sharing between processes

Parallelize the problem!

Parallelizing a problem: a silly (but complex) example

Sum up a bunch of numbers (here: from 1 to 8) and calculate the Exp od the sum using 4 threads



Programmer's perspective: planning complex parallelization

Algorithm design

- ❑ Identify parallelizable portions of the problem
- ❑ Minimize the sequential (non-parallelizable) part
- ❑ Consider/minimize synchronization and inter-thread communication
- ❑ Avoid race conditions
- ❑ Avoid simultaneous I/O by multiple threads
- ❑ Threads organization
  - Single process with multiple threads
  - Multiple single-threaded processes
  - Multiple multi-threaded processes

Constraints

- ❑ CPUs and memory possibly scattered over multiple networked machines
- ❑ #threads <= #cores (on each machine)
- ❑ Combined memory taken up by all processes not to exceed total machine's memory
- ❑ Storage capacity and access
- ❑ A process (with all its threads and memory) can only run on one machine
- ❑ No direct data sharing between processes

Programmer's perspective: tools

For complicated algorithms with varying levels of parallelism and communication, programs are typically written using appropriate parallelization tools (libraries of functions). By design, these programs fall into one of the following categories:

- ❑ Single multi-threaded process (by far the largest class)
  - Sometimes called shared memory model
  - Tools: **pthreads**, **OpenMP**
  - Advantage: all threads have access to same memory – no or easy communication
  - Disadvantage: can only run on one machine (but really no problem if machine huge)

- ❑ Multiple single-thread processes
  - Sometimes called distributed memory model
  - Tools: Message-Passing Interface (**MPI**) (Implementations: **OpenMPI**, **mpich2**)
  - Advantage: can run on a single machine and/or across multiple machines
  - Disadvantage: no direct access to process memory by other processes – data must be passed using messages – costly, especially between machines

- ❑ Multiple multi-threaded processes
  - Tools: combination of **OpenMP**, **pthreads**, **MPI**
  - Advantages: optimized, high-level parallelism possible
  - Advantage: can run on a single machine and/or across multiple machines

Find out how a program is parallelized

(easy only for executables using shared libraries)

```
[root@cbsuxm01 ~]# ldd /programs/bin/blast+/blastx

        linux-vdso.so.1 =>  (0x00007ffd0f79b000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007facee470000)
        libz.so.1 => /lib64/libz.so.1 (0x00007facee25a000)
        libbz2.so.1 => /lib64/libbz2.so.1 (0x00007facee04a000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007facede46000)
        libnsl.so.1 => /lib64/libnsl.so.1 (0x00007facedc2c000)
        libm.so.6 => /lib64/libm.so.6 (0x00007faced92a000)
        libc.so.6 => /lib64/libc.so.6 (0x00007faced55c000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007faced346000)
        /lib64/ld-linux-x86-64.so.2 (0x00007facee68c000)
```

```
[root@cbsuxm01 ~]# ldd /programs/discovar/bin/Discovar

        linux-vdso.so.1 =>  (0x00007fff1cd8a000)
        libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007fc9ba791000)
        libm.so.6 => /lib64/libm.so.6 (0x00007fc9ba48f000)
        libgomp.so.1 => /lib64/libgomp.so.1 (0x00007fc9ba269000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007fc9ba053000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fc9b9e37000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fc9b9a6a000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc9baa98000)
```

Find out how a program is parallelized

(easy only for executables using shared libraries)

```
[root@cbsuxm01 ~]# ldd /programs/ima2p/bin/IMa2p

        linux-vdso.so.1 =>  (0x00007ffd843b9000)
        libm.so.6 => /lib64/libm.so.6 (0x00007f5100e44000)
        libmpi_cxx.so.1 => /usr/lib64/openmpi/lib/libmpi_cxx.so.1 (0x00007f5100c29000)
        libmpi.so.12 => /usr/lib64/openmpi/lib/libmpi.so.12 (0x00007f5100945000)
        libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00007f510063e000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00007f5100428000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f510020c000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f50ffe3f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f5101146000)
        libopen-rte.so.12 => /usr/lib64/openmpi/lib/libopen-rte.so.12 (0x00007f50ffbc3000)
        libopen-pal.so.13 => /usr/lib64/openmpi/lib/libopen-pal.so.13 (0x00007f50ff91f000)
        libdl.so.2 => /lib64/libdl.so.2 (0x00007f50ff71b000)
        librt.so.1 => /lib64/librt.so.1 (0x00007f50ff513000)
        libutil.so.1 => /lib64/libutil.so.1 (0x00007f50ff310000)
        libhwloc.so.5 => /lib64/libhwloc.so.5 (0x00007f50ff0d3000)
        libnuma.so.1 => /lib64/libnuma.so.1 (0x00007f50feec7000)
        libltdl.so.7 => /lib64/libltdl.so.7 (0x00007f50fecbd000)
```

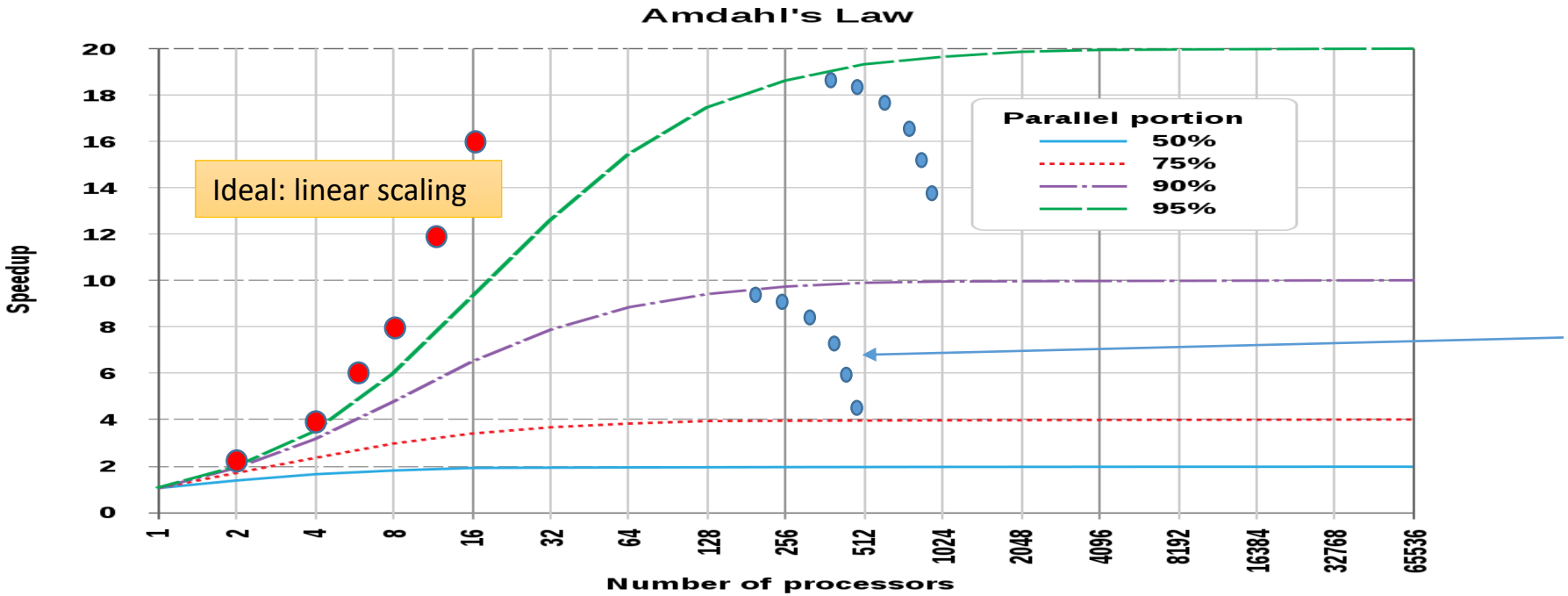
Amdahl's Law: More threads not always better

Suppose the total execution time of a program consists of non-parallelizable part t_{seq} and a part that can be parallelized, t_{par}. Then for number of threads N we have

Time on a single thread: $T_1 = t_{\text{seq}} + t_{\text{par}}$

Time on N threads: $T_N = t_{\text{seq}} + \frac{t_{\text{par}}}{N}$ (assuming no communication or other delays)

Speedup on N threads: $S_N = \frac{T_1}{T_N} \xrightarrow[\text{large } N]{} 1 + \frac{t_{\text{par}}}{t_{\text{seq}}}$



Performance <u>deterioration</u> possible due to sync/communication/IO

Example: speedup in BLAST

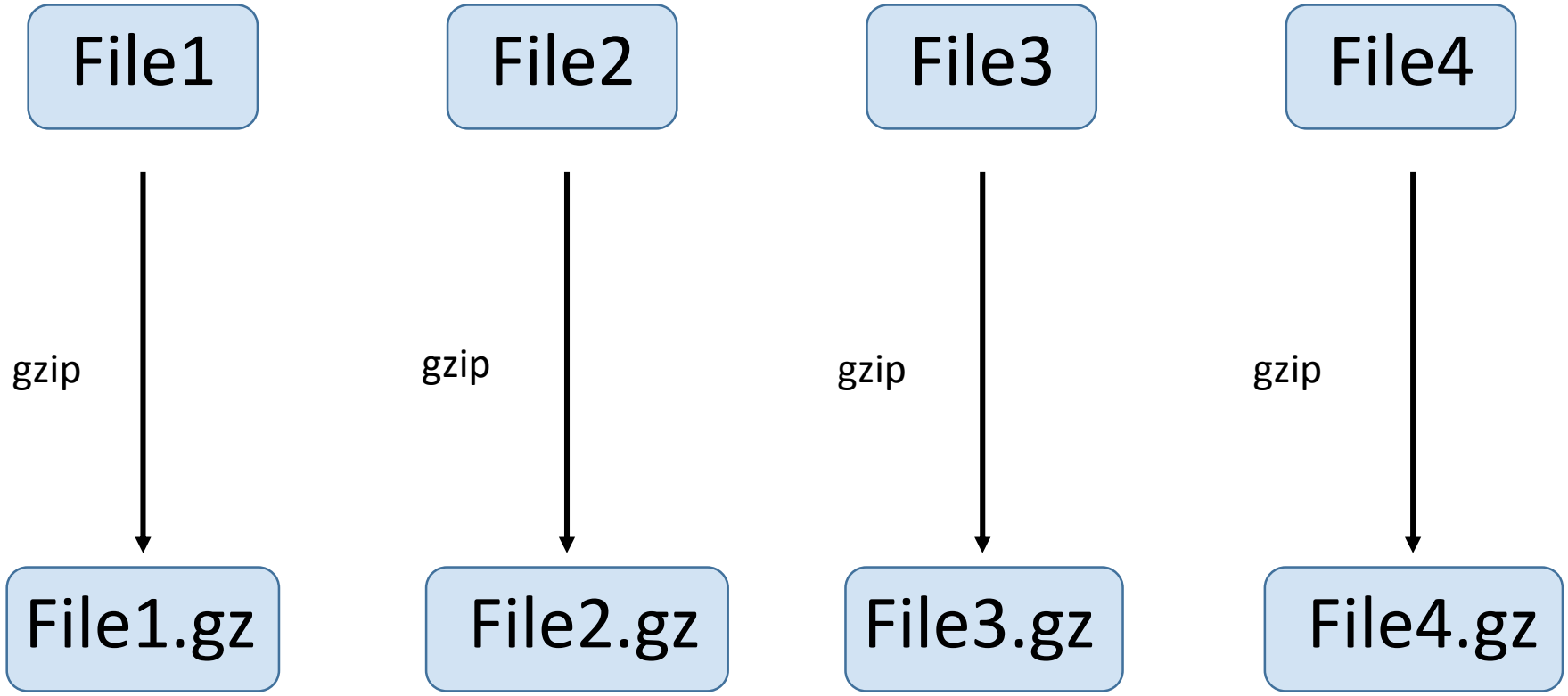
Using **BLAST** to search **swissprot** database for matches of 10,000 randomly chosen human cDNA sequences (swissprot is a good example of a small memory footprint).

machine	CPU available	cores available	cores used	time (hrs)	speedup (in machine)
cbsulm10	4	64	**64**	0.931	27.506
cbsulm10	4	64	**16**	1.962	13.056
cbsulm10	4	64	**1**	25.619	1.000
cbsumm15	2	24	**24**	2.058	12.117
cbsumm15	2	24	**12**	2.593	9.616
cbsumm15	2	24	**1**	24.930	1.000
cbsum1c2b008	2	8	**8**	4.193	6.717
cbsum1c2b008	2	8	**1**	28.161	1.000

Using **BLAST** to search **nr** database for matches of 2,000 randomly chosen human cDNA sequences (nr is a good example of a large memory footprint).

machine	CPU available	cores available	cores used	time (hrs)	speedup (in machine)
cbsulm10	4	64	**64**	10.97	2.222
cbsulm10	4	64	**16**	24.37	1.000
cbsumm15	2	24	**24**	26.10	2.140
cbsumm15	2	24	**12**	55.85	1.000

Parallelizing a problem: 'embarrassingly parallel' case



Single-threaded processes run simultaneously

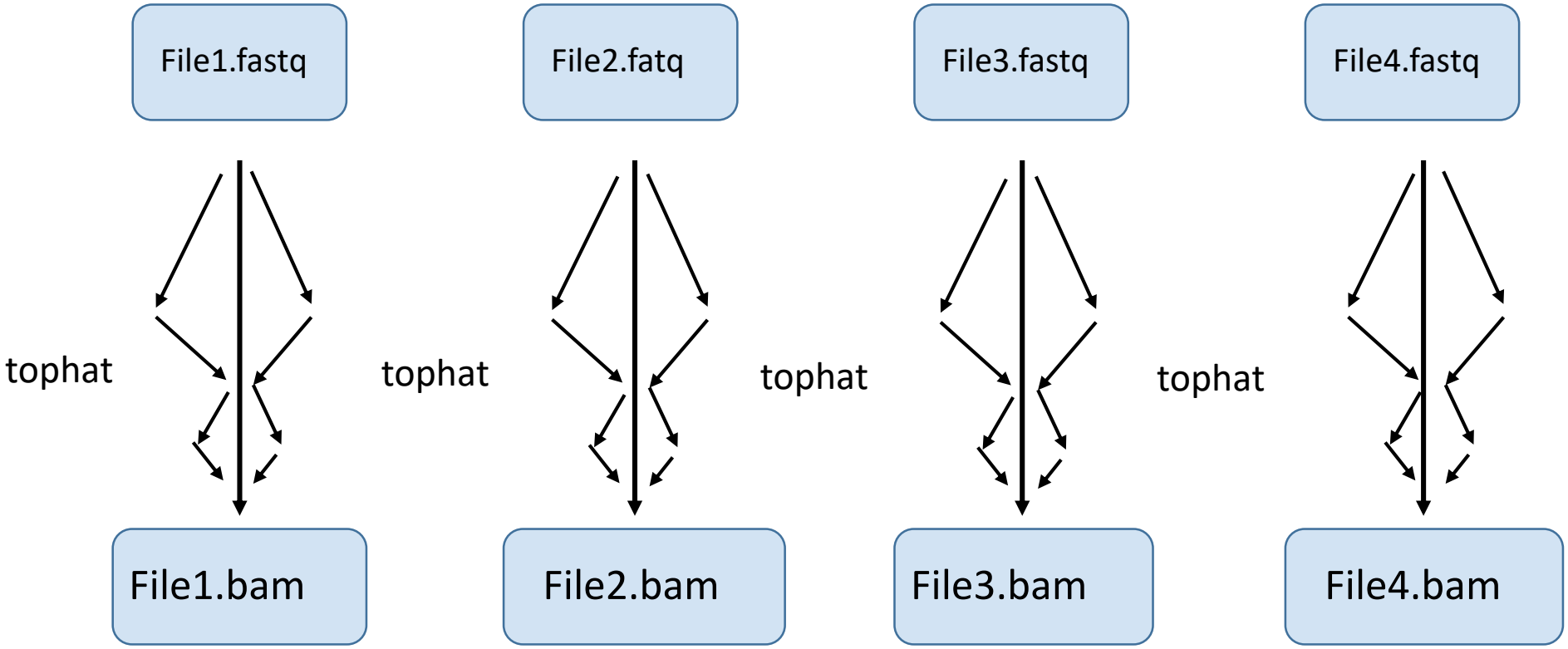
```
gzip File1 &
gzip File2 &
gzip File3 &
gzip File4 &
```

Simple! No communication, no sync, no sequential part – least susceptible to Amdahl's law

BUT

Threads compete for disk access if too many!

Parallelizing a problem: ‘not so embarrassingly parallel’ case



Complex multi-threaded processes, <u>independent from one another</u>

Simple! No communication **between processes**, no sync

BUT

Processes compete form disk access if too many!

Mixed parallelization: running several simultaneous multi-threaded tasks (each processing different data) on a large machine (here: 64-core)

```
tophat -p 7 -o B_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7073_C3AR7ACXX_B_L1-1_ATCACG_R1.fastq.gz \
   fastq/2284_6063_7073_C3AR7ACXX_B_L1-1_ATCACG_R2.fastq.gz >& B_L1-1.log &
tophat -p 7 -o B_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7076_C3AR7ACXX_B_L1-2_TGACCA_R1.fastq.gz  \
   fastq/2284_6063_7076_C3AR7ACXX_B_L1-2_TGACCA_R2.fastq.gz >& B_L1-2.log &
tophat -p 7 -o B_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7079_C3AR7ACXX_B_L1-3_CAGATC_R1.fastq.gz  \
   fastq/2284_6063_7079_C3AR7ACXX_B_L1-3_CAGATC_R2.fastq.gz >& B_L1-3.log &
tophat -p 7 -o L_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7074_C3AR7ACXX_L_L1-1_CGATGT_R1.fastq.gz \
   fastq/2284_6063_7074_C3AR7ACXX_L_L1-1_CGATGT_R2.fastq.gz >& L_L1-1.log &
tophat -p 7 -o L_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7077_C3AR7ACXX_L_L1-2_ACAGTG_R1.fastq.gz  \
   fastq/2284_6063_7077_C3AR7ACXX_L_L1-2_ACAGTG_R2.fastq.gz >& L_L1-2.log &
tophat -p 7 -o L_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7080_C3AR7ACXX_L_L1-3_ACTTGA_R1.fastq.gz \
   fastq/2284_6063_7080_C3AR7ACXX_L_L1-3_ACTTGA_R2.fastq.gz >& L_L1-3.log &
tophat -p 7 -o S_L1-1 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7075_C3AR7ACXX_S_L1-1_TTAGGC_R1.fastq.gz \
   fastq/2284_6063_7075_C3AR7ACXX_S_L1-1_TTAGGC_R2.fastq.gz >& S_L1-1.log &
tophat -p 7 -o S_L1-2 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7078_C3AR7ACXX_S_L1-2_GCCAAT_R1.fastq.gz \
   fastq/2284_6063_7078_C3AR7ACXX_S_L1-2_GCCAAT_R2.fastq.gz >& S_L1-2.log &
tophat -p 7 -o S_L1-3 --transcriptome-index genome/transcriptome/ZmB73_5a_WGS \
   --no-novel-juncs genome/maize \
   fastq/2284_6063_7081_C3AR7ACXX_S_L1-3_GATCAG_R1.fastq.gz \
   fastq/2284_6063_7081_C3AR7ACXX_S_L1-3_GATCAG_R2.fastq.gz >& S_L1-3.log &
```

Faster than **`tophat -p 63`** !

Common situation in ‘end user’ bioinformatics

- Instances of complex, multi-threaded applications <u>run concurrently on distinct sets of input data</u>
  - Examples: BLAST, bwa, tophat, STAR, Trinity, …..
  - applications ‘pre-programmed’ for us by software developers

- What we need to know about each instance of the application
  - how to run the application, know/control number of threads it uses
  - memory, disk, disk I/O, time requirements of the application (may depend on number of threads)
  - optimal number of threads for given input data, machine

  Run, monitor, observe, extrapolate…

- Load balancing: How to manage multiple instances subject to resource constraints
  - (#instances) X (#threads_per_instance) < #cores on each machine
  - (memory_per_instance) X (#instances) < total_machine_memory
  - competition for I/O bandwidth
  - sufficient scratch disk storage

Running multi-threaded applications

Parallelism is typically controlled by a program option

- read documentation to find out if your program has this feature
- Look for keywords like “multithreading”, “parallel execution”, “multiple processors”, etc.

A few examples:

```
blastall -a 8 [other options]

blastx -num_threads 8 [other options]

tophat -p 8 [other options]

cuffdiff -p 8 [other options]

bwa -t 8 [other options]

bowtie -p 8 [other options]
```

Remember speedup is not perfect, so optimal number of threads needs to be optimized by trial and error using subset of input data

Running multi-threaded applications

```
blastx -num_threads 2 -db ./databases/swissprot -query seq_tst.fa
```

bukowski@cbsum1c1b003:/workdir/bukowski/blast_test

```
top - 15:42:34 up 49 days, 19:59,  3 users,  load average: 0.87, 0.77, 0.78
Tasks: 306 total,   1 running, 305 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.0 us,  2.0 sy,  0.0 ni, 64.5 id,  6.6 wa,  0.0 hi,  1.9 si,  0.0 st
KiB Mem : 16264868 total,  2582488 free,   582616 used, 13099764 buff/cache
KiB Swap: 19455996 total, 19454452 free,     1544 used. 14229552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26831 bukowski  20   0  451108  47712  40184 S 200.3  0.3   0:08.00 blastx
21297 root      20   0       0      0      0 S   5.0  0.0 236:10.04 socknal_sd02_00
21715 bukowski  20   0  356256  10400   5360 D   3.7  0.1 139:10.07 tracker-store
21305 root      20   0       0      0      0 S   2.0  0.0  34:26.34 ptlrpcd_00_00
  491 root      20   0   67020  15580   1440 S   1.0  0.1 577:19.78 plymouthd
12338 root      20   0       0      0      0 S   1.0  0.0  29:12.95 ldlm_cb02_002
 8295 gdm       20   0  727644  21144   3368 S   0.7  0.1 198:58.38 gsd-color
21298 root      20   0       0      0      0 S   0.7  0.0  35:12.57 socknal_sd02_01
21306 root      20   0       0      0      0 S   0.7  0.0  34:16.95 ptlrpcd_00_01
21344 root      20   0       0      0      0 S   0.7  0.0  29:14.19 ldlm_cb02_000
21345 root      20   0       0      0      0 S   0.7  0.0  29:10.91 ldlm_cb02_001
    9 root      20   0       0      0      0 S   0.3  0.0  10:24.98 rcu_sched
 4291 root      20   0   90500   2976   2120 S   0.3  0.0  10:32.26 rngd
 6292 root      20   0       0      0      0 S   0.3  0.0   6:55.95 ldlm_bl_04
```

- >100% CPU indicates the program is **multithreaded**
  - Multiple threads within a single process rather than multiple processes

What if the number of threads is not specified?

Default number of threads for a multi-threaded programs

- Depends on the program's author(s)
- Sometimes 1
- Sometimes equal to the number of cores found on machine (rather nasty in shared environment)
- Programs parallelized with **OpenMP** 'obey' environment variable **`OMP_NUM_THREADS`**

  ```
  export OMP_NUM_THREADS=10
  ```

  will make such program use up to 10 threads (BioHPC default: 1)

- Programs parallelized with Intel's **Math Kernel Library (MKL)** require variable **`MKL_NUM_THREADS`** (BioHPC default: 1) in addition to **`OMP_NUM_THREADS`**

- Programs parallelized with **pthreads**: you are at the developer's mercy….

Running MPI applications

Message-Passing Interface (MPI)

- Used to create programs running as multiple interacting processes
- May run across multiple machines (Distributed Memory) – may use huge number of cores (in principle)
- Interaction between processes by sending/receiving **messages**
  - mechanism dependent on where processes are running (one or multiple machines), but generally costly…
- Each MPI process may be multithreaded (i.e., use pthreads and/or OpenMP)
- Various implementations (**OpenMPI** and **mpich2** most popular – both available on BioHPC cloud)

Running MPI programs

Programs using MPI are started using a launcher program **`mpirun`** (some variations on that name are possible, depending on MPI implementation)

Run using 10 processes on the local machine (the one the command is run on)

```
mpirun -np 10 myprogram >& somefile.log &
```

To run on multiple machines, construct a file with a list of machines, **`mymachines`**, possibly specifying some limits on number of processes to be allowed

```
cbsum1c1b001 slots=4 max_slots=4
cbsum1c2b003 max_slots=4
cbsum1c2b002 slots=4
```

NOTE: each MPI process may be multi-threaded!

Then, for example, the command

```
mpirun -hostfile mymachines -np 14 myprogram >& somefile.log &
```

will launch 4 processes on **`cbsum1c1b001`**, 4 more on **`cbsum1c2b003`**, and 6 on **`cbsum1c2b002`** (oversubscription possible on this node)

Plenty of other options for distributed processes on nodes.

Killing parallel tasks may be tricky

- If the application is running in the **background** (i.e., with "**&**"), it can be stopped with the **`kill`** command

```
kill -9 <PID>
```

Where <PID> is the process id obtained rom the **`ps`** command. For example,

```
kill -9 18817
```

- To kill a parallel application consisting of <u>multiple processes</u>, use the PID of the top parent process, preceded by a dash

```
kill -9 -18817
```

(technically, this kills all processes in the <u>process group</u> 18817)

- If some processes, still left over, you may have to track them down (with **`ps`**) and kill individually

Monitoring a running task using top

```
blastx -num_threads 2 -db ./databases/swissprot -query seq_tst.fa
```

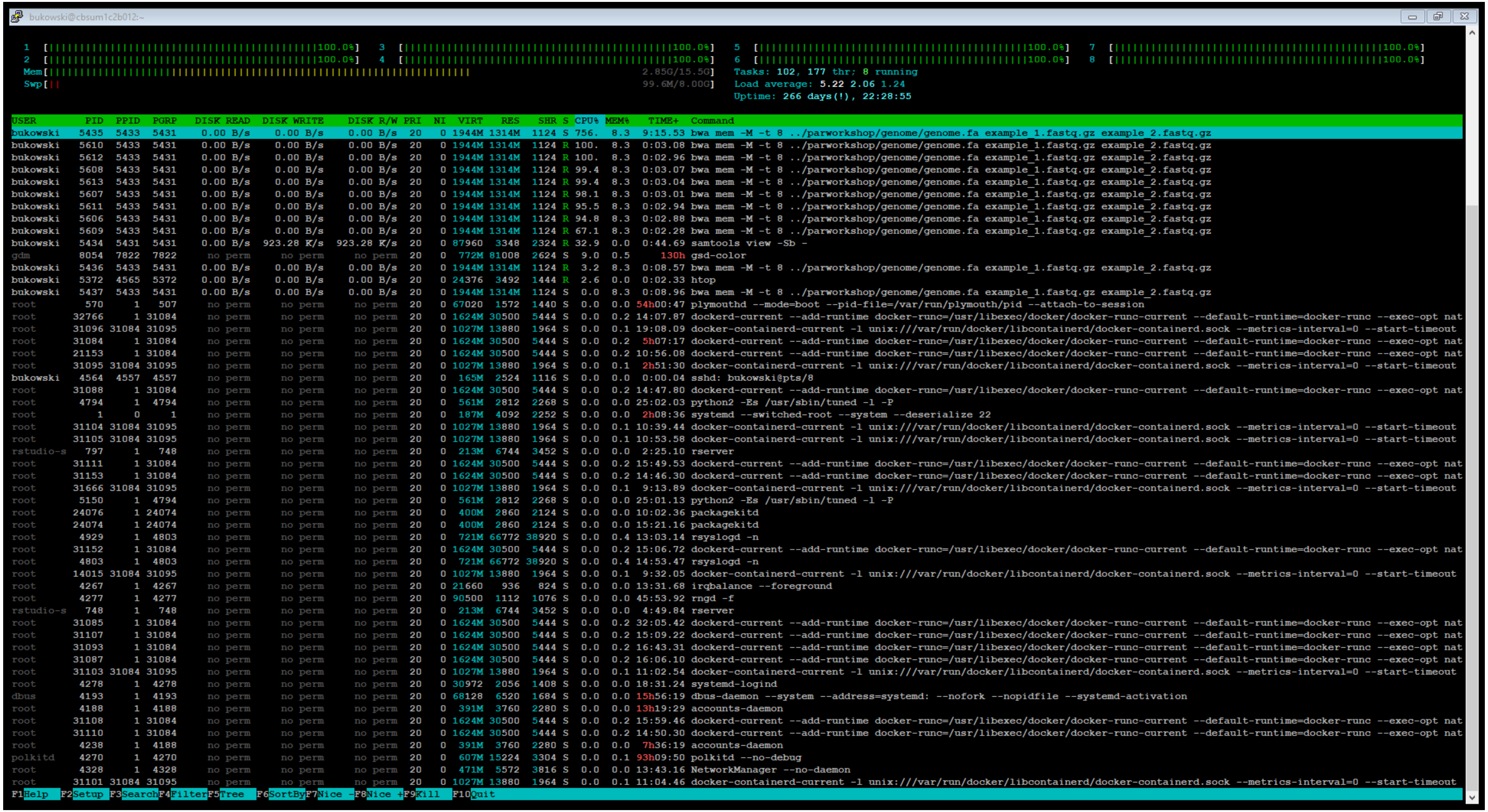
bukowski@cbsum1c1b003:/workdir/bukowski/blast_test

```
top - 15:42:34 up 49 days, 19:59,  3 users,  load average: 0.87, 0.77, 0.78
Tasks: 306 total,   1 running, 305 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.0 us,  2.0 sy,  0.0 ni, 64.5 id,  6.6 wa,  0.0 hi,  1.9 si,  0.0 st
KiB Mem : 16264868 total,  2582488 free,   582616 used, 13099764 buff/cache
KiB Swap: 19455996 total, 19454452 free,     1544 used. 14229552 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26831 bukowski  20   0  451108  47712  40184 S 200.3  0.3   0:08.00 blastx
21297 root      20   0       0      0      0 S   5.6  0.0 236:10.04 socknal_sd02_00
21715 bukowski  20   0  356256  10400   5360 D   3.7  0.1 139:10.07 tracker-store
21305 root      20   0       0      0      0 S   2.0  0.0  34:26.34 ptlrpcd_00_00
  491 root      20   0   67020  15580   1440 S   1.0  0.1 577:19.78 plymouthd
12338 root      20   0       0      0      0 S   1.0  0.0  29:12.95 ldlm_cb02_002
 8295 gdm       20   0  727644  21144   3368 S   0.7  0.1 198:58.38 gsd-color
21298 root      20   0       0      0      0 S   0.7  0.0  35:12.57 socknal_sd02_01
21306 root      20   0       0      0      0 S   0.7  0.0  34:16.95 ptlrpcd_00_01
21344 root      20   0       0      0      0 S   0.7  0.0  29:14.19 ldlm_cb02_000
21345 root      20   0       0      0      0 S   0.7  0.0  29:10.91 ldlm_cb02_001
    9 root      20   0       0      0      0 S   0.3  0.0  10:24.98 rcu_sched
 4291 root      20   0   90500   2976   2120 S   0.3  0.0  10:32.26 rngd
 6292 root      20   0       0      0      0 S   0.3  0.0   6:55.95 ldlm_bl_04
```

- >100% CPU indicates the program is **multithreaded**
  - Multiple threads within a single process rather than multiple processes

Monitoring a running task using `htop`



```
  1  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   3  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   5  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   7  [||||||||||||||||||||||||||||||||||||||||||||100.0%]
  2  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   4  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   6  [||||||||||||||||||||||||||||||||||||||||||||100.0%]   8  [||||||||||||||||||||||||||||||||||||||||||||100.0%]
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||                          2.85G/15.5G]   Tasks: 102, 177 thr; 8 running
  Swp[||                                                                                             99.6M/8.00G]   Load average: 5.22 2.06 1.24
                                                                                                                      Uptime: 266 days(!), 22:28:55

USER       PID  PPID  PGRP  DISK READ  DISK WRITE    DISK R/W PRI  NI  VIRT   RES   SHR S CPU% MEM%    TIME+ Command
bukowski  5435  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 S 756.  8.3  9:15.53 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5610  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 100.  8.3  0:03.08 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5612  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 100.  8.3  0:02.96 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5608  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 99.4  8.3  0:03.07 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5613  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 99.4  8.3  0:03.04 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5607  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 98.1  8.3  0:03.01 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5611  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 95.5  8.3  0:02.94 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5606  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 94.8  8.3  0:02.88 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5609  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R 67.1  8.3  0:02.28 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5434  5431  5431   0.00 B/s  923.28 K/s  923.28 K/s  20   0 87960  3348  2324 R 32.9  0.0  0:44.69 samtools view -Sb -
gdm       8054  7822  7822    no perm     no perm     no perm  20   0  772M 81008  2624 S  9.0  0.5     130h gsd-color
bukowski  5436  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 R  3.2  8.3  0:08.57 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
bukowski  5372  4565  5372   0.00 B/s    0.00 B/s    0.00 B/s  20   0 24376  3492  1444 R  2.6  0.0  0:02.33 htop
bukowski  5437  5433  5431   0.00 B/s    0.00 B/s    0.00 B/s  20   0 1944M 1314M  1124 S  0.0  8.3  0:08.96 bwa mem -M -t 8 ../parworkshop/genome/genome.fa example_1.fastq.gz example_2.fastq.gz
root       570     1   507    no perm     no perm     no perm  20   0 67020  1572  1440 S  0.0  0.0 54h00:47 plymouthd --mode=boot --pid-file=/var/run/plymouth/pid --attach-to-session
root     32766     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 14:07.87 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31096 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1 19:08.09 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
root     31084     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2  5h07:17 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     21153     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 10:56.08 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31095 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1  2h51:30 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
bukowski  4564  4557  4557    no perm     no perm     no perm  20   0  165M  2524  1116 S  0.0  0.0  0:00.04 sshd: bukowski@pts/8
root     31088     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 14:47.80 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root      4794     1  4794    no perm     no perm     no perm  20   0  561M  2812  2268 S  0.0  0.0 25:02.03 python2 -Es /usr/sbin/tuned -l -P
root         1     0     1    no perm     no perm     no perm  20   0  187M  4092  2252 S  0.0  0.0  2h08:36 systemd --switched-root --system --deserialize 22
root     31104 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1 10:39.44 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
root     31105 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1 10:53.58 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
rstudio-s  797     1   748    no perm     no perm     no perm  20   0  213M  6744  3452 S  0.0  0.0  2:25.10 rserver
root     31111     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 15:49.53 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31153     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 14:46.30 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31666 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1  9:13.89 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
root      5150     1  4794    no perm     no perm     no perm  20   0  561M  2812  2268 S  0.0  0.0 25:01.13 python2 -Es /usr/sbin/tuned -l -P
root     24076     1 24074    no perm     no perm     no perm  20   0  400M  2860  2124 S  0.0  0.0 10:02.36 packagekitd
root     24074     1 24074    no perm     no perm     no perm  20   0  400M  2860  2124 S  0.0  0.0 15:21.16 packagekitd
root      4929     1  4803    no perm     no perm     no perm  20   0  721M 66772 38920 S  0.0  0.4 13:03.14 rsyslogd -n
root     31152     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 15:06.72 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root      4803     1  4803    no perm     no perm     no perm  20   0  721M 66772 38920 S  0.0  0.4 14:53.47 rsyslogd -n
root     14015 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1  9:32.05 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
root      4267     1  4267    no perm     no perm     no perm  20   0 21660   936   824 S  0.0  0.0 13:31.68 irqbalance --foreground
root      4277     1  4277    no perm     no perm     no perm  20   0 90500  1112  1076 S  0.0  0.0 45:53.92 rngd -f
rstudio-s  748     1   748    no perm     no perm     no perm  20   0  213M  6744  3452 S  0.0  0.0  4:49.84 rserver
root     31085     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 32:05.42 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31107     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 15:09.22 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31093     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 16:43.31 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31087     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 16:06.10 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31103 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1 11:02.54 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
root      4278     1  4278    no perm     no perm     no perm  20   0 30972  2056  1408 S  0.0  0.0 18:31.24 systemd-logind
dbus      4193     1  4193    no perm     no perm     no perm  20   0 68128  6520  1684 S  0.0  0.0 15h56:19 dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
root      4188     1  4188    no perm     no perm     no perm  20   0  391M  3760  2280 S  0.0  0.0 13h19:29 accounts-daemon
root     31108     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 15:59.46 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root     31110     1 31084    no perm     no perm     no perm  20   0 1624M 30500  5444 S  0.0  0.2 14:50.30 dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt nat
root      4238     1  4188    no perm     no perm     no perm  20   0  391M  3760  2280 S  0.0  0.0  7h36:19 accounts-daemon
polkitd   4270     1  4270    no perm     no perm     no perm  20   0  607M 15224  3304 S  0.0  0.1 93h09:50 polkitd --no-debug
root      4328     1  4328    no perm     no perm     no perm  20   0  471M  5572  3816 S  0.0  0.0 13:43.16 NetworkManager --no-daemon
root     31101 31084 31095    no perm     no perm     no perm  20   0 1027M 13880  1964 S  0.0  0.1 11:04.46 docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-timeout
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Monitoring a single task using `/usr/bin/time` tool

```
/usr/bin/time -v blastx -db ./databases/swissprot -num_alignments 1 -num_threads 3 -query
seq_tst.fa -out seq_tst.fa.hits.txt >& run.log

Command being timed: "blastx -db ./databases/swissprot -num_alignments 1 -num_threads 3 -query
seq_tst.fa -out seq_tst.fa.hits.txt"
        User time (seconds): 35.86
        System time (seconds): 0.15
        Percent of CPU this job got: 292%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:12.31
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 208488
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 59067
        Voluntary context switches: 51
        Involuntary context switches: 147
        Swaps: 0
        File system inputs: 0
        File system outputs: 312
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

Shows ‘user’ time combined over all threads

Max memory the process used in its lifetime

Content of `run.log`

Assess I/O activity using `iostat`

No significant I/O

bukowski@cbsuem02:/workdir/bukowski/blast_test

```
[bukowski@cbsuem02 blast_test]$ iostat -y -d 3
Linux 3.10.0-957.10.1.el7.x86_64 (cbsuem02.biohpc.cornell.edu)  05/01/20        _x86_64_        (112 CPU)

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
sdc               0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
md1               0.00         0.00         0.00          0          0
md0               0.00         0.00         0.00          0          0
dm-0              0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
sdc               3.00         0.00        10.33          0         31
sdb               3.00         0.00        10.33          0         31
md1               2.00         0.00         8.00          0         24
md0               0.00         0.00         0.00          0          0
dm-0              0.00         0.00         0.00          0          0

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
sdc               0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
md1               0.00         0.00         0.00          0          0
md0               0.00         0.00         0.00          0          0
dm-0              0.00         0.00         0.00          0          0

^C
[bukowski@cbsuem02 blast_test]$
[bukowski@cbsuem02 blast_test]$
[bukowski@cbsuem02 blast_test]$
```

Monitoring I/O using `iotop` tool

```
/programs/bin/labutils/iotop -o -u bukowski
```

```
root@cbsuem02:~
Total DISK READ :       0.00 B/s | Total DISK WRITE :       0.00 B/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      19.70 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
```

No I/O-intensive processes running

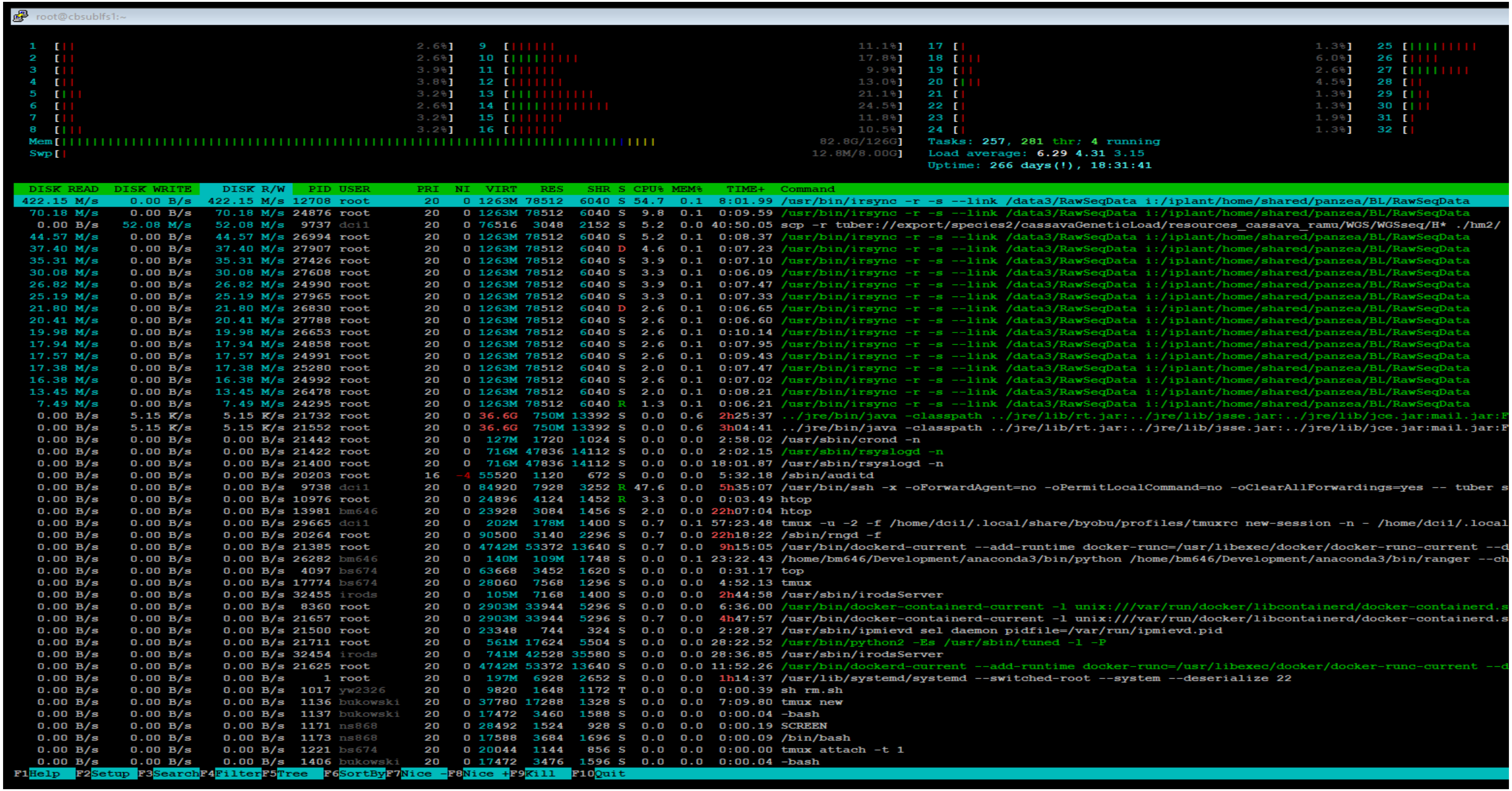
```
root@cbsuem02:~
Total DISK READ :       0.00 B/s | Total DISK WRITE :       4.13 M/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      71.27 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 81177 be/4 bukowski    0.00 B/s 1395.44 K/s  0.00 %  0.00 % gzip BBB_1
 81178 be/4 bukowski    0.00 B/s 1440.46 K/s  0.00 %  0.00 % gzip BBB_2
 81179 be/4 bukowski    0.00 B/s 1395.44 K/s  0.00 %  0.00 % gzip BBB_3
```

Three gzip processes running

Read/write rate

% time spent
waiting for I/O

Monitoring I/O using `htop` tool



```
  1  [||                                        2.6%]   9  [||||||                                   11.1%]   17 [|                                                 1.3%]   25 [|||||||||
  2  [||                                        2.6%]   10 [|||||||||                                17.8%]   18 [|||                                               6.0%]   26 [||||
  3  [||                                        3.9%]   11 [||||||                                    9.9%]   19 [||                                                2.6%]   27 [||||||||
  4  [||                                        3.8%]   12 [|||||||                                  13.0%]   20 [|||                                               4.5%]   28 [||
  5  [|||                                       3.2%]   13 [|||||||||||                              21.1%]   21 [|                                                 1.3%]   29 [|||
  6  [||                                        2.6%]   14 [|||||||||||||                            24.5%]   22 [|                                                 1.3%]   30 [|||
  7  [||                                        3.2%]   15 [|||||||                                  11.8%]   23 [|                                                 1.9%]   31 [|
  8  [|||                                       3.2%]   16 [||||||                                   10.5%]   24 [|                                                 1.3%]   32 [|
  Mem[|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||          82.8G/126G]   Tasks: 257, 281 thr; 4 running
  Swp[|                                                                                       12.8M/8.00G]   Load average: 6.29 4.31 3.15
                                                                                                            Uptime: 266 days(!), 18:31:41

  DISK READ  DISK WRITE    DISK R/W   PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 422.15 M/s   0.00 B/s  422.15 M/s 12708 root       20   0 1263M 78512  6040 S 54.7  0.1  8:01.99 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  70.18 M/s   0.00 B/s   70.18 M/s 24876 root       20   0 1263M 78512  6040 S  9.8  0.1  0:09.59 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
   0.00 B/s  52.08 M/s   52.08 M/s  9737 dci1       20   0 76516  3048  2152 S  5.2  0.0 40:50.05 scp -r tuber://export/species2/cassavaGeneticLoad/resources_cassava_ramu/WGS/WGSseq/H* ./hm2/
  44.57 M/s   0.00 B/s   44.57 M/s 26994 root       20   0 1263M 78512  6040 S  5.2  0.1  0:08.37 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  37.40 M/s   0.00 B/s   37.40 M/s 27907 root       20   0 1263M 78512  6040 D  4.6  0.1  0:07.23 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  35.31 M/s   0.00 B/s   35.31 M/s 27426 root       20   0 1263M 78512  6040 S  3.9  0.1  0:07.10 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  30.08 M/s   0.00 B/s   30.08 M/s 27608 root       20   0 1263M 78512  6040 S  3.3  0.1  0:06.09 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  26.82 M/s   0.00 B/s   26.82 M/s 24990 root       20   0 1263M 78512  6040 S  3.9  0.1  0:07.47 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  25.19 M/s   0.00 B/s   25.19 M/s 27965 root       20   0 1263M 78512  6040 S  3.3  0.1  0:07.33 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  21.80 M/s   0.00 B/s   21.80 M/s 26830 root       20   0 1263M 78512  6040 D  2.6  0.1  0:06.65 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  20.41 M/s   0.00 B/s   20.41 M/s 27788 root       20   0 1263M 78512  6040 S  2.6  0.1  0:06.60 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  19.98 M/s   0.00 B/s   19.98 M/s 26653 root       20   0 1263M 78512  6040 S  2.6  0.1  0:10.14 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  17.94 M/s   0.00 B/s   17.94 M/s 24858 root       20   0 1263M 78512  6040 S  2.6  0.1  0:07.95 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  17.57 M/s   0.00 B/s   17.57 M/s 24991 root       20   0 1263M 78512  6040 S  2.6  0.1  0:09.43 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  17.38 M/s   0.00 B/s   17.38 M/s 25280 root       20   0 1263M 78512  6040 S  2.0  0.1  0:07.47 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  16.38 M/s   0.00 B/s   16.38 M/s 24992 root       20   0 1263M 78512  6040 S  2.6  0.1  0:07.02 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
  13.45 M/s   0.00 B/s   13.45 M/s 26478 root       20   0 1263M 78512  6040 S  2.0  0.1  0:08.21 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
   7.49 M/s   0.00 B/s    7.49 M/s 24295 root       20   0 1263M 78512  6040 R  1.3  0.1  0:06.21 /usr/bin/irsync -r -s --link /data3/RawSeqData i:/iplant/home/shared/panzea/BL/RawSeqData
   0.00 B/s   5.15 K/s    5.15 K/s 21732 root       20   0 36.6G  750M 13392 S  0.0  0.6  2h25:37 ../jre/bin/java -classpath ../jre/lib/rt.jar:../jre/lib/jsse.jar:../jre/lib/jce.jar:mail.jar:F
   0.00 B/s   5.15 K/s    5.15 K/s 21552 root       20   0 36.6G  750M 13392 S  0.0  0.6  3h04:41 ../jre/bin/java -classpath ../jre/lib/rt.jar:../jre/lib/jsse.jar:../jre/lib/jce.jar:mail.jar:F
   0.00 B/s   0.00 B/s    0.00 B/s 21442 root       20   0  127M  1720  1024 S  0.0  0.0  2:58.02 /usr/sbin/crond -n
   0.00 B/s   0.00 B/s    0.00 B/s 21422 root       20   0  716M 47836 14112 S  0.0  0.0  2:02.15 /usr/sbin/rsyslogd -n
   0.00 B/s   0.00 B/s    0.00 B/s 21400 root       20   0  716M 47836 14112 S  0.0  0.0 18:01.87 /usr/sbin/rsyslogd -n
   0.00 B/s   0.00 B/s    0.00 B/s 20203 root       16  -4 55520  1120   672 S  0.0  0.0  5:32.18 /sbin/auditd
   0.00 B/s   0.00 B/s    0.00 B/s  9738 dci1       20   0 84920  7928  3252 R 47.6  0.0  5h35:07 /usr/bin/ssh -x -oForwardAgent=no -oPermitLocalCommand=no -oClearAllForwardings=yes -- tuber s
   0.00 B/s   0.00 B/s    0.00 B/s 10976 root       20   0 24896  4124  1452 R  3.3  0.0  0:03.49 htop
   0.00 B/s   0.00 B/s    0.00 B/s 13981 bm646      20   0 23928  3084  1456 S  2.0  0.0 22h07:04 htop
   0.00 B/s   0.00 B/s    0.00 B/s 29665 dci1       20   0  202M  178M  1400 S  0.7  0.1 57:23.48 tmux -u -2 -f /home/dci1/.local/share/byobu/profiles/tmuxrc new-session -n - /home/dci1/.local
   0.00 B/s   0.00 B/s    0.00 B/s 20264 root       20   0 90500  3140  2296 S  0.7  0.0 22h18:22 /sbin/rngd -f
   0.00 B/s   0.00 B/s    0.00 B/s 21385 root       20   0 4742M 53372 13640 S  0.7  0.0  9h15:05 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --d
   0.00 B/s   0.00 B/s    0.00 B/s 26282 bm646      20   0  140M  109M  1748 S  0.0  0.1 23:22.43 /home/bm646/Development/anaconda3/bin/python /home/bm646/Development/anaconda3/bin/ranger --ch
   0.00 B/s   0.00 B/s    0.00 B/s  4097 bs674      20   0 63668  3452  1620 S  0.0  0.0  0:31.17 top
   0.00 B/s   0.00 B/s    0.00 B/s 17774 bs674      20   0 28060  7568  1296 S  0.0  0.0  4:52.13 tmux
   0.00 B/s   0.00 B/s    0.00 B/s 32455 irods      20   0  105M  7168  1400 S  0.0  0.0  2h44:58 /usr/sbin/irodsServer
   0.00 B/s   0.00 B/s    0.00 B/s  8360 root       20   0 2903M 33944  5296 S  0.0  0.0  6:36.00 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.s
   0.00 B/s   0.00 B/s    0.00 B/s 21657 root       20   0 2903M 33944  5296 S  0.7  0.0  4h47:57 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.s
   0.00 B/s   0.00 B/s    0.00 B/s 21500 root       20   0 23348   744   324 S  0.0  0.0  2:28.27 /usr/sbin/ipmievd sel daemon pidfile=/var/run/ipmievd.pid
   0.00 B/s   0.00 B/s    0.00 B/s 21711 root       20   0  561M 17624  5504 S  0.0  0.0 28:22.52 /usr/bin/python2 -Es /usr/sbin/tuned -l -P
   0.00 B/s   0.00 B/s    0.00 B/s 32454 irods      20   0  741M 42528 35580 S  0.0  0.0 28:36.85 /usr/sbin/irodsServer
   0.00 B/s   0.00 B/s    0.00 B/s 21625 root       20   0 4742M 53372 13640 S  0.0  0.0 11:52.26 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --d
   0.00 B/s   0.00 B/s    0.00 B/s     1 root       20   0  197M  6928  2652 S  0.0  0.0  1h14:37 /usr/lib/systemd/systemd --switched-root --system --deserialize 22
   0.00 B/s   0.00 B/s    0.00 B/s  1017 yw2326     20   0  9820  1648  1172 T  0.0  0.0  0:00.39 sh rm.sh
   0.00 B/s   0.00 B/s    0.00 B/s  1136 bukowski   20   0 37780 17288  1328 S  0.0  0.0  7:09.80 tmux new
   0.00 B/s   0.00 B/s    0.00 B/s  1137 bukowski   20   0 17472  3460  1588 S  0.0  0.0  0:00.04 -bash
   0.00 B/s   0.00 B/s    0.00 B/s  1171 ns868      20   0 28492  1524   928 S  0.0  0.0  0:00.19 SCREEN
   0.00 B/s   0.00 B/s    0.00 B/s  1173 ns868      20   0 17588  3684  1696 S  0.0  0.0  0:00.09 /bin/bash
   0.00 B/s   0.00 B/s    0.00 B/s  1221 bs674      20   0 20044  1144   856 S  0.0  0.0  0:00.00 tmux attach -t 1
   0.00 B/s   0.00 B/s    0.00 B/s  1406 bukowski   20   0 17472  3476  1596 S  0.0  0.0  0:00.04 -bash
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

Monitoring I/O

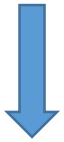
- Notoriously hard, because
  - most I/O operations are <u>buffered</u> and <u>cached</u>, i.e., go through memory if enough available
  - I/O behavior of a single task not always representative of that of concurrent tasks
  - performance dependent on disk hardware
    - slow on **`cbsum1c*`** machines
    - very fast on the newest machines with NVMEs (SSDs with fast connect)
  - performance dependent on data structure (a lot of small files vs few large files)

- Indications of heavy I/O problem:
  - small **`%CPU`** compared to number of threads in **`top`** or **`htop`** report
  - large **`%IO`** in **`iotop`** output (% of time the process spends waiting for I/O operation)
  - continuously high Read-Write rates in **`iotop`** or **`htop`** report

Ultimate test: monitor performance as a function of number of concurrent tasks

Balancing the load: multiple independent tasks

- Suppose we monitored/profiled our application and we <u>already know</u>
  - memory needed per instance
  - optimal number of threads per instance
  - at least a vague idea about I/O needs per instance



  - **`N`** - number of instances to be run concurrently

What if the total number of tasks we have is >> **`N`** ?

Example: compress 9 files, running at most 3 instances of **`gzip`** at a time

Balancing the load: pedestrian way

Example: 9 tasks, 3 at a time

```
#!/bin/bash

gzip [options] file1 &
gzip [options] file2 &
gzip [options] file3 &

wait

gzip [options] file4 &
gzip [options] file5 &
gzip [options] file6 &

wait

gzip [options] file7 &
gzip [options] file8 &
gzip [options] file9 &
```

(NOTE: **`wait`** – makes the script wait for everything before it to finish before proceeding)

Not too efficient, if compressing different **`file*`** takes different amounts of time

`wait` needs to wait for the <u>slowest</u> of the three instances

Load balancing using GNU `parallel`

https://www.gnu.org/software/parallel/

Using a text editor, create a file called (for example) **`TaskFile`**
(This is **NOT** a script, just a list of commands to run)

```
gzip file1
gzip file2
gzip file3
gzip file4
gzip file5
gzip file6
gzip file7
gzip file8
gzip file9
```

A longer file could be created, for example, using a shell script similar to:

```
#!/bin/bash

rm -f TaskFile
for i in {1..3000}
do
        echo gzip file${i} >> TaskFile
done
```

Load balancing using GNU `parallel` tool

Then run the command (assuming the **`TaskFile`** and all **`file*`** files are in the current directory)

```
parallel -j NP < TaskFile >& log &
```

where **`NP`** is the number of instances to use (e.g., 3)

- **`parallel`** will execute tasks listed in **`TaskFile`** using up to **`NP`** instances at a time
  - The first **`NP`** tasks will be launched simultaneously
  - The **`(NP+1)`** th task will be launched right after one of the initial ones completes and a core becomes available
  - The **`(NP+2)`** nd task will be launched right after another core becomes available
  - …… etc., until all tasks are distributed

- Only up to **`NP`** tasks are running at a time (less at the end)

- All **`NP`** cores always kept (on average) busy (except near the end of task list) – **Load Balancing**

GNU `parallel`: general idea and syntax

Suppose **`someprog`** is a program taking one argument, and we want to run it **`N`** times with N values of that argument:

```
someprog a1
someprog a2
someprog a3
…
someprog aN
```

GNU parallel can help:

```
parallel [options] someprog ::: a1 a2 a3 … aN
```

will start these commands running <u>concurrently</u>

`[options]` are there to control things (examples later)

(so, in essence, parallel just concatenates **`someprog`** with each of **`ai`** and treats those as commands to run)

GNU `parallel`: general idea and syntax

Instead of listing arguments, we can put them in a file, say **`argfile`**, listing one argument per line like this:

```
a1
a2
a3
…
aN
```

Then run **`parallel`** like this (note the <u>four</u> colons `::::`)

```
parallel [options] someprog :::: argfile
```

Equivalent forms:

```
parallel [options] -a argfile someprog
```

```
cat argfile | parallel [options] someprog
```

```
parallel [options] someprog < argfile
```

GNU `parallel`: general idea and syntax

Remember the 'original' command we introduced **`parallel`** with?

```
parallel -j 10 < TaskFile
```

where **`TaskFile`** was

```
gzip file1
gzip file2
…
gzip file3000
```

This is like running

```
parallel -j 10 someprog :::: TaskFile
```

with <u>empty</u> **`someprog`** and 'arguments' in the form **`gzip file1`**

GNU `parallel`: general idea and syntax

What is the **`someprog`** command needs more than 1 argument?

```
parallel -N2 someprog ::: a1 a2 a3 a4 a5 a6
```

will produce the following commands:

```
someprog a1 a2
someprog a3 a4
someprog a5 a6
```

GNU `parallel`: general idea and syntax

What if we need to run a not one, but a few commands?

```
parallel someprog1 {}\; someprog2 {} ::: a1 a2 a3
```

(`{}` represents the argument, if only one)

will result in

```
someprog1 a1; someprog2 a1     # run one after the other, but concurrently with other such pairs
someprog1 a2; someprog2 a2
someprog1 a3; someprog2 a3
```

Another example: **`someprog1`** and **`someprog2`** run on <u>different</u> arguments

```
parallel -N2 someprog1 {1}\; someprog2 {2} ::: a1 a2 a3 a4
```

(`{1},{2}` represent individual arguments, if multiple)

will result in

```
someprog1 a1; someprog2 a2
someprog1 a3; someprog2 a4
```

GNU `parallel`: more control through <u>options</u>

```
parallel -j 4 --delay 5 --load 200% --memfree 2G  someprog :::: argfile
```

`-j 4`	run up to 4 commands concurrently
`--delay 5`	start each command 5 seconds after previous one
`--load 200%`	start command only if load on the machine is not more than 2 threads
`--memfree 2G`	start command only if there is at least 2G of RAM available

Caution:

If **`someprog`** is multi-threaded, it will ‘occupy’ not 4, but (4 x number_threads_per_task) CPU cores !!!

GNU `parallel`: remote execution (and more options)

```
parallel -j 2 \
-S machine1 -S machine2 \
--transferfile BBB_{} \
--return BBB_{}.gz \
--workdir /workdir/bukowski \
--cleanup \
--joblog run.log \
gzip ::: 1 2 3
```

What will happen here:

- Commands **`gzip BBB_1`**, **`gzip BBB_2`**, and **`gzip BBB_3`** will be run, at most 2 at a time, using machines **`machine1`**, **`machine2`**, accessed via ssh
- Files **`BBB_1`**, **`BBB_2`**, and `BBB_3` will be transferred from the current directory to the relevant machine to directory **`/workdir/bukowski`**, and the 'gzipping' will take place there.
- Upon completion, compressed files **`BBB_1.gz`**, **`BBB_2.gz`**, and **`BBB_3.gz`** will be transferred back to the current directory.
- Files on the remote machines will be cleaned up
- Log of the entire operation, with some useful timing information, will be saved in file **`run.log`** (in the current directory on the current machine, from which **`parallel`** was submitted)

NOTE: user should have passwordless ssh access set up between the machines to avoid being asked for password…

GNU `parallel`: killing tasks

Find the process ID (PID) of the parallel process

```
ps -ef | grep parallel
bukowski 28310  1710  1 13:50 pts/13   00:00:00 perl /programs/parallel/bin/parallel -j 2 gzip BBB_{} ::: 1 2 3
bukowski 28558  1710  0 13:50 pts/13   00:00:00 grep --color=auto parallel
```

Now send the SIGTERM signal to the process c- this will ‘drain the queue’ (allow tasks already running to finish)

```
kill -15 28310
parallel: SIGTERM received. No new jobs will be started.
parallel: Waiting for these 2 jobs to finish. Send SIGTERM again to stop now.
```

Send the SIGTERM signal <u>again</u> to kill off the remaining running processes

```
kill -15 28310
```

`xargs` – 'older brother' of GNU `parallel`

Functionality of **`xargs`** similar (but more limited) than that of **`parallel`**
some options of **`parallel`** designed to mimic those of **`xargs`**

Example:

Let **`TaskFile`** contain a list of files

```
file1
file2
file3
```

```
cat TaskFile | xargs gzip
```

will construct (and run) the following, using a single process (i.e., 3 **`gzip`** operations one after another)

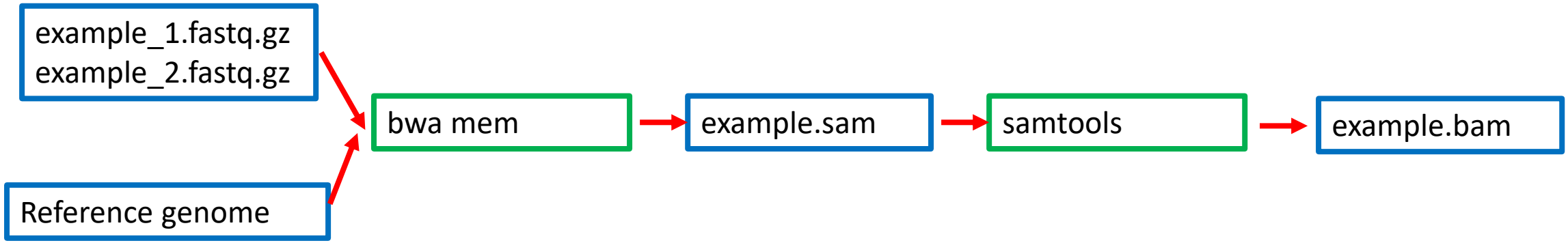
```
gzip file1 file2 file3
```

```
cat TaskFile | xargs -n 1 -P 2 gzip
```

will construct (and run) the following, using up to 2 processes at a time

```
gzip file1
gzip file2
gzip file3
```

Exercise 3: timing `bwa mem` alignment



Pipe to avoid explicit creation of **`example.sam`**

```
bwa mem -M -t 8 genome example_1.fastq.gz example_2.fastq.gz | samtools view -Sb - -o example.bam
```

parallel (here: 8 threads)

sequential

Objective:

run this on varying numbers of threads

measure time, memory, I/O as functions of that number

GNU `parallel` limitations

- Parallel is a clever tool for submitting multiple commands in the background, possibly on multiple machines

- Very useful extra options, such as (and there are many more):

`-j 4`	limit the number of commands run concurrently (here: 4)
`--delay 5`	start each command some time (here: 5 seconds) after previous one
`--load 200%`	start command only if load on the machine is not more than some number (here: 2) of threads
`--memfree 2G`	start command only if there is at least some mount (here: 2G) of RAM is available
`--timeout 60`	impose timeout (here: 60 s) on each command

- But there are limitations to what **`parallel`** can do for you:

  - Once a command is running, no control over how many cores or how much memory it uses (may overwhelm machine)
  - Can't control individual commands
  - No way to enforce fair sharing of resources among multiple users and/or user groups

Need a SCHEDULER to deal with these!

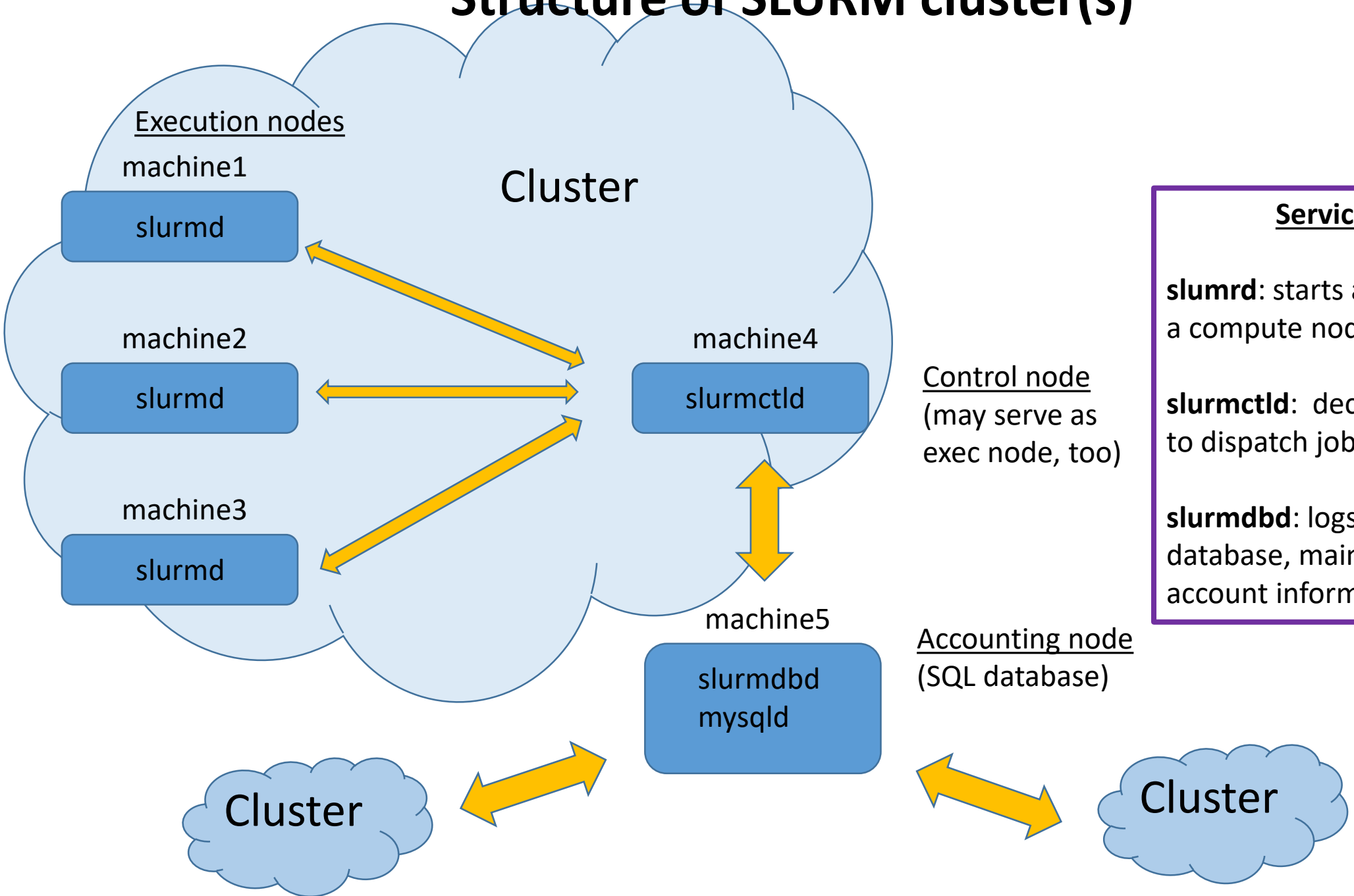
GNU `parallel` vs a full scheduler

Functionality	Parallel	Scheduler
Start multiple jobs on limited resources	yes	yes
Terminate individual jobs	no (or hard)	yes
Control #cores and memory of running jobs	no	yes
Prioritize jobs of different users, groups	no	yes
Control job timeout	yes	yes
Streamline submission based on job requirements	no	yes
File pre-staging	yes	yes (sort of)
Job staggering	yes	yes ?
Job accounting	no (or limited)	yes

Some popular schedulers

- Obsolete or commercialized
  - **PBS**: Portable Batch System
  - **SGE**: Sun Grid Engine
  - **LFS**: Load Sharing Facility
  - **Lava**: light version of LSF
  - **TORQUE** (version) of PBS
  - **UNIVA**: commercial fork of SGE
- Free, modern, and actively developed
  - SLURM: Simple Linux Utility for Resource Management

Structure of SLURM cluster(s)



Services (daemons)

slumrd: starts and manages jobs on a compute node

slurmctld: decides where and when to dispatch jobs, tracks jobs

slurmdbd: logs jobs information into database, maintains user and account information

SLURM setup is an admin task

- Non-trivial setup and maintenance
  - Require extra pieces of software installed <u>running</u> on all machines involved
  - To be started, configured, and maintained by an administrator (users generally cannot do it)
  - Takes significant know-how and work to set up and configure
  - Configuration typically tailored specifically to a particular cluster/lab/group/institution
- Users need to follow usage guidelines for the specific scheduler configuration
  - Learning curve involved – different for each cluster

Configuration of resources in SLURM

- Nodes (machines) grouped into partitions (queues)
  - typically collect similar nodes, or nodes with similar function
  - each node may belong to multiple partitions
  - partition may have per job limits and defaults (run time, memory, max #cores, etc)
  - User needs to specify which partition their job is to be submitted to
  - One partition is ‘default’
- Cluster may be configured to grant jobs either whole nodes, or node ‘slices’ (i.e., some #cores + some memory)
  - jobs are restricted to #cores and RAM requested at submission – will not use more (may crash on attempt!)
  - #cores and RAM allocated to a running job are subtracted from the node’s totals – only available resources are offered to new jobs
- Users organized in (trees of) accounts (e.g., lab groups), with defined shares, determining usage priorities
- Per user and/or per group limits or privileges may be defined (QOS – quality of service)

SLURM at BioHPC

- **Permanent clusters**, made up of Lab- or Department-owned machines, customized to serve those Labs or Departments. Access for lab members only.

  - BSCB cluster (cbsubscb): 15 nodes, 1136 CPU cores, 5.8 TB RAM
  - cbsuxu
  - cbsuorm
  - cbsugaurav
  - others welcome – contact us to discuss/set up

- **'SLURM on demand'** clusters:

  - possible to spin up by <u>any user</u> on their reserved or Lab-owned machine(s)
  - access for all users with reservations on these machines
  - temporary – will disappear upon the end of reservation
  - not configurable (at present, only a single configuration is offered)

  - What are they good for:

    load balancing of single or multiple users' jobs (like **`parallel`**, but with more control)
    re-using SLURM scripts brought from elsewhere (some customization typically be required)
    running pipelines which require SLURM for load balancing (and some do)

‘SLURM on demand’ at BioHPC

- ❑ Reserve one or more machines
- ❑ Log in to one of the reserved machines
- ❑ Use **`manage_slurm`** tool to spin up and control (some aspects of) the cluster

```
[bukowski@cbsum1c2b005 ~]$ manage_slurm
Usage: manage_slurm <action> [args]
manage_slurm new machine1,machine2,...
     •  to create a SLURM cluster on the named machines (need an active reservation on all machines). The first node
        will be the "master node". All users with active reservations on the full set of machines will be given
        access to the cluster, and will automatically be added or removed as their reservation status changes.
manage_slurm kill masterNode
     •  to end the slurm cluster identified by the master node
manage_slurm addNode masterNode machine
     •  adds the machine to the cluster identfied by the masterNode. Need an active reservation on this machine for
        all current cluster users.
manage_slurm add Node-force masterNode machine
     •  Like addNode, but will remove cluster access for any users necessary to add the node (including deleting any
        of their submitted jobs). Try addNode first to get a list of users that will be removed
manage_slurm removeNode masterNode machine
     •  Remove a node from the cluster identified by masterNode. The machine should not be the masterNode; removing
        the masterNode will kill entire cluster, so use "kill" command instead.
manage_slurm list
     •  List and decribe all clusters that you have access to. Reports the list of machines, number of CPUs/memory,
        and list of authorized users
```

More info: https://biohpc.cornell.edu/lab/userguide.aspx?a=software&i=689#c

‘SLURM on demand’ at BioHPC configuration

At present, only one configuration offered:

- One partition (‘regular’)
  - contains all nodes
  - no per job time limit
  - no per job CPU core limits
  - 4 GB RAM per job default
- One ‘account’, containing all users having reservations on all machines of the ‘cluster’
  - Fairshare scheduling policy with all users ‘equal’ (more details later)
  - No per user limits

SLURM: know your cluster – partitions, nodes summary

Info on ‘current cluster’, which the node belongs to

```
[bukowski@cbsum1c2b003 ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
regular*     up   infinite      4   idle cbsum1c2b[003-004,006-007]
```

Info on other clusters

```
[bukowski@cbsum1c2b003 slurm]$ sinfo --cluster=cbsubscb
CLUSTER: cbsubscb
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
short*       up    4:00:00     16    mix cbsubscb[01-15],cbsubscbgpu01
regular      up 1-00:00:00     16    mix cbsubscb[01-15],cbsubscbgpu01
long7        up 7-00:00:00     15    mix cbsubscb[01-15]
long30       up 30-00:00:0     15    mix cbsubscb[01-15]
gpu          up 3-00:00:00      1    mix cbsubscbgpu01
```

SLURM: know your cluster – node details

```
[bukowski@cbsum1c2b003 slurm]$ scontrol show nodes=cbsum1c2b006
NodeName=cbsum1c2b006 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUTot=8 CPULoad=0.01
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=128.84.181.157 NodeHostName=cbsum1c2b006
   OS=Linux 3.10.0-957.10.1.el7.x86_64 #1 SMP Mon Mar 18 15:06:45 UTC 2019
   RealMemory=15883 AllocMem=0 FreeMem=3254 Sockets=8 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=regular
   BootTime=2020-04-02T12:56:07 SlurmdStartTime=2020-05-10T08:50:09
   CfgTRES=cpu=8,mem=15883M,billing=8
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

SLURM: know your cluster – partition details

```
[bukowski@cbsum1c2b003 slurm]$ scontrol show partitions
PartitionName=regular
   AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
   AllocNodes=ALL Default=YES QoS=N/A
   DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=0 LLN=NO
MaxCPUsPerNode=UNLIMITED
   Nodes=cbsum1c2b003,cbsum1c2b004,cbsum1c2b006,cbsum1c2b007
   PriorityJobFactor=1 PriorityTier=1 RootOnly=NO ReqResv=NO OverSubscribe=NO
   OverTimeLimit=NONE PreemptMode=OFF
   State=UP TotalCPUs=32 TotalNodes=4 SelectTypeParameters=NONE
   JobDefaults=(null)
   DefMemPerNode=4096 MaxMemPerNode=UNLIMITED
```

SLURM and you – typical scenario

- ❑ Determine job's CPU-cores and RAM requirements

- ❑ Write a shell script that will
  - create a job directory on local scratch file system
  - prepare (copy) input files to job's scratch
  - launch the application (output to be written to job's scratch)
  - copy output files back to permanent storage (e.g., home directory)

- ❑ Submit script using **`sbatch`** command with desired options (#cores, RAM, partition, nodes, …)
  - may embed SLURM options in the script header
  - interactive session may be requested using **`srun`** command
  - submit as many jobs as you need

- ❑ Jobs are queued up and wait for resources and their turn to start on some node (competing with other jobs)

- ❑ Check on your jobs using **`squeue`**, state of cluster using **`sinfo`**, **`scontrol`**

- ❑ Control/cancel your jobs (**`scontrol update`**, **`scancel`**)

- ❑ Get information about finished jobs using **`sacct`**

- ❑ Handy summary of SURM commands: http://slurm.schedmd.com/pdfs/summary.pdf

SLURM: typical shell script

Typical shell script, call it **my_script.sh**

Make job inherit your login environment

Integer unique for every SLURM job

```
#!/bin/bash -l

# Create a scratch directory for this job
WDIR=/workdir/bukowski/$SLURM_JOB_ID
mkdir -p $WDIR
cd $WDIR

#Copy input to the scratch directory
cp /home/bukowski/inputs/input.file .

# Run the computation
/home/bukowski/programs/my_program < input.file >& output.file.$SLURM_JOB_ID

# Copy results back to permanent storage
cp output.file.$SLURM_JOB_ID /home/bukowski/outputs

# Clean up after job
cd /workdir/bukowski
rm -Rf $SLURM_JOB_ID
```

SLURM: submitting a script

```
sbatch [options] my_script.sh [arguments]
```

Here are some more important **[options]** with examples:

`--nodes=1`	(number of nodes, must be 1 for all non-MPI jobs)
`--ntasks=8`	(number of tasks; task=1 slot=1 thread; default: 1)
`--mem=8000`	(request 8 GB of memory for this job; default: 4GB)
`--time=1-20:00:00`	(wall-time limit for job; here: 1 day and 20 hours)
`--partition=regular,long30`	(request partition(s) a job can run in; here: **regular** and **long30**)
`--account=bscb09`	(project to charge the job to)
`--chdir=/home/bukowski/slurm`	(start job in specified directory; default is the directory in which **sbatch** was invoked)
`--job-name=jobname`	(name of job)
`--output=jobname.out.%j`	(write stdout+stderr to this file; `%j` will be replaced by job ID)
`--mail-user=email@address.com`	(set your email address)
`--mail-type=ALL`	(send email at job start, end or crash - do not use if this is going to generate thousands of e-mails!)

So, the submission command could look like

```
sbatch --nodes=1 --ntasks=6 --mem=4000 my_script.sh
```

Many options have <u>shorthand notation</u>, e.g.,

```
sbatch -N 1 -n 6 --mem=4000 my_script.sh
```

SLURM: specifying option in script header

Typical script header may specify submission options after **#SBATCH** keyword

```
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --mem=8000
#SBATCH --time=1-20:00:00
#SBATCH --partition=regular,long30
#SBATCH --account=bscb09
#SBATCH --chdir=/home/bukowski/slurm
#SBATCH --job-name=jobname
#SBATCH --output=jobname.out.%j
#SBATCH --mail-user=email@address.com
#SBATCH --mail-type=ALL

# Rest of the script goes here…
```

Options given directly on **sbatch** command line supersede the ones in header

SLURM: other interesting options/comments

`--ntasks`

used to request the number of threads (cores) for a job (also, set **`OMP_NUM_THREADS`** accordingly)

`-N 1`

non-MPI jobs **must** run on a single machine (otherwise some cores may be allocated on other nodes and won't be useful)

`--nodelist=cbsubscb11`

run on a specific node (otherwise it will start on <u>some node</u> within requested partitions – <u>unknown in advance</u>!)

`--exclude=cbsubscb10,cbsubscbgpu01`

exclude specified nodes

`--chdir` and **`--output`**

directories specified in these options must be present on all nodes where a job can start (e.g., **`$HOME`**)

Alternative core and memory specifications

```
sbatch –N 1 -n 8 --cpus-per-task=3 --mem-per-cpu=2G my_script.sh

sbatch –n 24 --mem=48G my_script.sh
```

SLURM: where to submit jobs <u>from</u>?

This is configurable, but at BioHPC

- Any node of the cluster (if you have **ssh** login access to it)

```
sbatch [options] my_script.sh [arguments]
```

- Any machine configured to use the same **`slurmdbd`** (database) service as the cluster in question, e.g., the login node **`cbsulogin.biohpc.cornell.edu`**. Use the **`--clusters`** option to indicate the cluster to submit to:

```
sbatch --clusters=cbsum1c2b003 [options] my_script.sh [arguments]
```

- Directory a job is submitted from becomes the job's 'startup directory' and so it <u>must exist on all nodes</u> the job may start on. **`$HOME`** is a good choice
  - job's 'startup directory' may be changed using option **`--chdir`**
  - jobs should use local scratch storage (rather than 'startup directory') for I/O-intensive computations

(some of) Environment Variables available within a SLURM job

`SLURM_JOB_CPUS_PER_NODE` : number of CPU cores (threads) allocated to this job

`SLURM_NTASKS` : number of tasks, or slots, for this job (as given by **`--ntasks`** option)

`SLURM_MEM_PER_NODE` : memory requested with --mem option

`SLURM_CPUS_ON_NODE` : total number of CPUs on the node (not only the allocated ones)

`SLURM_JOB_ID` : job ID of this job; may be used, for example, to name a scratch directory (subdirectory of **`/workdir`**, or output files) for the job. For array jobs, each array element will have a separate **`SLURM_JOB_ID`**

`SLURM_ARRAY_JOB_ID` : job ID of the array master job

`SLURM_ARRAY_TASK_ID` : task index of a task within a job array

`SLURM_ARRAY_TASK_MIN`, **`SLURM_ARRAY_TASK_MAX`** : minimum and maximum index of jobs within the array

Complete list – in section 'OUTPUT ENVIRONMENT VARIABLES' of https://slurm.schedmd.com/sbatch.html.

SLURM: Job arrays (array jobs?)

Consider script **my_array_script.sh** (objective: compress files **BBB_1**, **BBB_2**, **BBB_3**)

```
#!/bin/bash

# Prepare the scratch directory for the job and 'cd' to it
WDIR=/workdir/$SLURM_JOB_ID
mkdir -p $WDIR
cd $WDIR

# Copy the file to gzip from network-mounted directory
cp /shared_data/Parallel_workshop/BBB_$SLURM_ARRAY_TASK_ID .

# Run the compression
gzip BBB_$SLURM_ARRAY_TASK_ID

# Copy the result back into the result directory (here: same as submission dir)
cp BBB_${SLURM_ARRAY_TASK_ID}.gz $SLURM_SUBMIT_DIR

# Clean up
cd $SLURM_SUBMIT_DIR; rm -Rf $WDIR
```

Now submit this script as <u>job array</u>:

```
sbatch --array=1-3 [other_options] my_array_script.sh
```

This single command will submit <u>three separate jobs</u>, each running **my_array_script.sh**, but in each the value of **SLURM_ARRAY_TASK_ID** will be different (one of 1, 2, or 3) . This variable is provided as a result of the **--array** option.

SLURM: interactive jobs

```
srun –n 2 –N 1 --mem 2G --pty --preserve-env --cpu-bind=no /bin/bash
```

This will create a job (can check with **`squeue`**) and open an interactive **`bash`** shell on a machine picked by SLURM

This shell will be constrained to the number of cores an memory requested

After interactive work finished, exit the shell (Ctrl-D or ‘exit’)

On some clusters (cbsubscb), **`salloc`** may be configured to automatically execute the **`srun`** command as above

`srun`: very much like **`sbatch`**, except it works in real time rather than batch mode

SLURM: checking on jobs using `squeue`

```
[bukowski@cbsum1c2b003 slurm]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                19   regular blast_te bukowski PD       0:00      1 (Resources)
                18   regular bwa_test bukowski  R       0:48      1 cbsum1c2b003
              20_1   regular array_te bukowski  R       0:10      1 cbsum1c2b004
              20_2   regular array_te bukowski  R       0:10      1 cbsum1c2b004
              20_3   regular array_te bukowski  R       0:10      1 cbsum1c2b004
```

SLURM: checking on jobs using `squeue`

Default output from **`squeue`** is not that informative. It is better to use a more detailed format

```
squeue -o '%.18i %.10n %.20x %m %.30P %.8j %.8u %.8T %.10M %.9l %.6D %C %R %p %S'
```

or simply use the wrapper script (which we wrote)

```
squeue_l
```

```
[bukowski@cbsum1c2b003 slurm]$ ~bukowski/SLURM*/scr*/squeue_l -cluster=cbsum1c2b003
CLUSTER: cbsum1c2b003
             JOBID  REQ_NODES            EXC_NODES MIN_MEMORY             PARTITION     NAME     USER    STATE       TIME TIME_LIMI  NODES CPUS NODELIST(REASON) PRIORITY START_TIME
                24 cbsum1c2b003                   2G                        regular blast_te bukowski  PENDING       0:00 UNLIMITED      1 8 (Resources) 0.00000145519152 2021-05-10T11:51:37
                23                                4G                        regular bwa_test bukowski  RUNNING       0:57 UNLIMITED      1 8 cbsum1c2b003 0.00000145519152 2020-05-10T11:51:11
              25_1                                4G                        regular array_te bukowski  RUNNING       0:31 UNLIMITED      1 1 cbsum1c2b004 0.00000142958015 2020-05-10T11:51:37
              25_2                                4G                        regular array_te bukowski  RUNNING       0:31 UNLIMITED      1 1 cbsum1c2b004 0.00000142958015 2020-05-10T11:51:37
              25_3                                4G                        regular array_te bukowski  RUNNING       0:31 UNLIMITED      1 1 cbsum1c2b004 0.00000142958015 2020-05-10T11:51:37
```

SLURM: retrieving job accounting information using `sacct`

```
[bukowski@cbsum1c2b003 slurm]$ sacct
       JobID    JobName  Partition    Account  AllocCPUS      State ExitCode
------------ ---------- ---------- ---------- ---------- ----------- --------
3              bwa_test    regular acc_cbsum+          4  COMPLETED      0:0
3.batch           batch            acc_cbsum+          4  COMPLETED      0:0
4              bwa_test    regular acc_cbsum+          1  COMPLETED      0:0
4.batch           batch            acc_cbsum+          1  COMPLETED      0:0
5              bwa_test    regular acc_cbsum+          1  COMPLETED      0:0
5.batch           batch            acc_cbsum+          1  COMPLETED      0:0
```

For each job, all job steps are listed (job itself + the batch) – hence two lines per job.

SLURM: more job accounting information using `sacct_l`

`sacct_l`: a simple wrapper around **`sacct`**, giving more useful information

Produces rather long lines. In the example below, the lines are broken in half

Left-most columns of **`sacct_l`** output for two jobs

```
 JobID              JobIDRaw          JobName      User         NodeList     ReqMem
-------------------- -------------------- ---------- --------- --------------- --
 15                   15            bwa_test   bukowski    cbsum1c2b004        4Gn
 15.batch             15.batch      batch                  cbsum1c2b004        4Gn
 16_1                 17            array_test bukowski    cbsum1c2b005        4Gn
 16_1.batch           17.batch      batch                  cbsum1c2b005        4Gn
```

Right-most columns of **`sacct_l`** output for same two jobs

```
MaxRSS  MaxVMSize       NCPUS               Start    CPUTime   TotalCPU    UserCPU    Elapsed      State
---------- ---------- ---------- ------------------- ---------- ---------- ---------- ---------- --------
                          8   2020-05-10T11:29:53   00:19:36   12:26.591  12:13.738   00:02:27    COMPLETED
1402336K 2055932K         8   2020-05-10T11:29:53   00:19:36   12:26.591  12:13.738   00:02:27    COMPLETED
                          1   2020-05-10T11:30:12   00:01:08   01:03.112  01:02.051   00:01:08    COMPLETED
   1980K  132900K         1   2020-05-10T11:30:12   00:01:08   01:03.112  01:02.051   00:01:08    COMPLETED
```

SLURM: use accounting info to evaluate performance

Signature of an efficient job:

`CPUTime (= NCPU*Elapsed) ≈ TotalCPU ≈ UserCPU`
`MaxRSS < ReqMem`

Signs of inefficiency:

`TotalCPU < CPUTime`	inefficient parallelization (e.g., due to sequential part of jab, like file copying)
`UserCPU < TotalCPU`	lot of time spent is system calls (typically due to inefficient I/O)
`MaxRSS ≈ ReqMem`	job may have not enough memory, may crash or swap; increase **`--mem`**

`MaxRSS`: use this to estimate memory needs of your job

- Run test requesting a lot of memory, then for production runs set **`--mem`** to a value <u>slightly</u> larger (1.2 times?) than **`MaxRSS`** obtained from the test run

SLURM scheduler

What happens to queued jobs?

- Scheduler (part of **`slurmctld`** service daemon) runs periodically (once in about 1 minute)
  - keep track of running jobs and their allocated resources
  - keep track of available resources
  - compute <u>job priorities</u>
  - examine each waiting job, check if requested resources available
  - if multiple jobs compete, submit the ones with <u>highest priorities</u>
  - <u>backfilling:</u> ‘small’ jobs with lower priority may get ahead of ‘big’ jobs with higher priority if it does not affect the start time of the latter
    - accurate timing requests are necessary effective backfilling

Multi-factor job priority

```
Job_priority =
        site_factor +
        (PriorityWeightAge) * (age_factor) +
        (PriorityWeightAssoc) * (assoc_factor) +
        (PriorityWeightFairshare) * (fair-share_factor) +
        (PriorityWeightJobSize) * (job_size_factor) +
        (PriorityWeightPartition) * (partition_factor) +
        (PriorityWeightQOS) * (QOS_factor) +
        SUM(TRES_weight_cpu * TRES_factor_cpu,
           TRES_weight_<type> * TRES_factor_<type>,
           ...)
        - nice_factor
```

Association = (user, account, cluster, partition)

QOS: Quality of Service (set of limits or privileges)

TRES: trackable resource

JobSize: related to #cores requested

Fairshare: reflects proportion of resources consumed by user to user's 'share' in the cluster

Probably the most important factor in multi-user, multi-group clusters

Weigths: large integer numbers

Factors: numbers between 0 and 1

Fair Tree Fairshare example

AccountA (10 shares)

User1 (40 shares), usage=20
User2 (30 shares), usage=10

AccountB (20 shares)

User3 (50 shares), usage=30
User4 (60 shares), usage=40
User5 (10 shares), usage=0

Relative share S:

AccountA: 10/(10+20) = 1/3
AccountB: 20/(10+20) = 2/3
User1: 40/(30+40) = 4/7
User2: 30/(30+40) = 3/7
User3: 50/(50+60+10) = 5/12
User4: 60/(50+60+10) = 6/12
User5: 10/(50+60+10) = 1/12

Relative usage U:

AccountA: 30/(30+70) = 3/10
AccountB: 70/(30+70) = 7/10
User1: 20/(10+20) = 2/3
User2: 10/(10+20) = 1/3
User3: 30/(30+40) = 3/7
User4: 40/(30+40) = 4/7
User5: 0/(30+40) = 0

Level Fairshare LF = S/U:

AccountA: 10/9
AccountB: 20/21
User1: 6/7
User2: 8/7
User3: 35/36
User4: 42/48
User5: Infinity

First sort **accounts** according to their LF, **then** sort **users** within accounts according to their LF:

AccountA (LF=10/9)

User2 (LF=8/7) Priority: 5/5
User1 (LF=6/7) Priority: 4/5

AccountB (LF=20/21)

User5 (LF=Infinity) Priority: 3/5
User3 (LF=35/36) Priority: 2/5
User4 (LF=42/48) Priority: 1/5

fair-share_factor
~ Rank

What is ‘Usage’ anyway?

Usage for a running job during time period **`Dt`**

```
U_job = [#cores*core_weigth + RAM*RAM_weigth] * Dt
```

(defaults: **`core_weigth=1`**, **`RAM_weigth=0`**, **`Dt=5 min`**)

Total usage **`U_user`** for a user: sum of **`U_job`** over all user’s jobs

Taking into account past usage:

```
U_user = U_now + d*U_now_1 + d*U_now_2 + d*d*d*U_now_3 + …
```

Where

`U_now_N` is the user’s usage **`U_user`** from time period **`N*Dt`** before the present one

`d` is set based on the assumed usage half-life time, **`T_half`** (e.g., 1 week), i.e.,

```
d = (1/2)^(Dt/T_half) < 1
```

SLURM documentation

- Version installed on BioHPC: https://slurm.schedmd.com/archive/slurm-19.05.2/
  - upgrade needed soon…
- Man pages for individual commands: https://slurm.schedmd.com/archive/slurm-19.05.2/man_index.html
- SURM command summary handout: http://slurm.schedmd.com/pdfs/summary.pdf

- Formal documentation very thorough, but rather formal, with few specific examples
  - often ‘googling’ a specific subject or command will yield more clear info from ‘SLURM practitioners’