Graduate School Class Reminders

- Maintain six feet of distancing
- Please sit in the same chair each class time
- Observe entry/exit doors as marked
- Use hand sanitizer when you enter/exit the classroom
- Use a disinfectant wipe/spray to wipe down your learning space before and after class
- Media Services: 414 955-4357 option 2

Documentation on the web

- CRAN: `http://cran.r-project.org`
- R manuals: `https://cran.r-project.org/manuals.html`
- SAS: `http://support.sas.com/documentation`
- Step-by-Step Programming with Base SAS 9.4 (SbS): `https://documentation.sas.com/api/docsets/basess/9.4/content/basess.pdf`
- SAS 9.4 Programmer s Guide: Essentials (PGE): `https://documentation.sas.com/api/docsets/lepg/9.4/content/lepg.pdf`
- Wiki: `https://wiki.biostat.mcw.edu` (MCW/VPN)

Multi-threading example: the Mandelbrot set

- To study multi-threading, a computational example is helpful that takes some time to complete, but NOT too much time
- Complex numbers are of the form: $c = x + y\mathbf{i}$ where $\mathbf{i}^2 = -1$
  $Re(c) = x, Im(c) = y,\ Mod(c) = \sqrt{x^2 + y^2}$
  R functions: Real part `Re`; Imaginary `Im`; and Modulus `Mod`
- Multiplying two complex numbers together has different behavior than real numbers, e.g., the product's modulus need NOT be greater than the terms multiplied even if both quantities have real and imaginary parts greater than one
- The Mandelbrot set contains the following $c = x + y\mathbf{i}$ values
- Suppose $d_0 = 0 + 0\mathbf{i}$
- Iterate the relation: $d_n = d_{n-1}^2 + c$ for $n = 1, \ldots, N$
- It is known that if $Mod(d_n) > 2$, then c is NOT in the set
- How many iterations, n, does it take d_n to diverge for each c?
- Heat map: $Re(c)$, $Im(c)$ and n are the x, y and z axes

Myles Harrison at `http://everydayanalytics.ca`

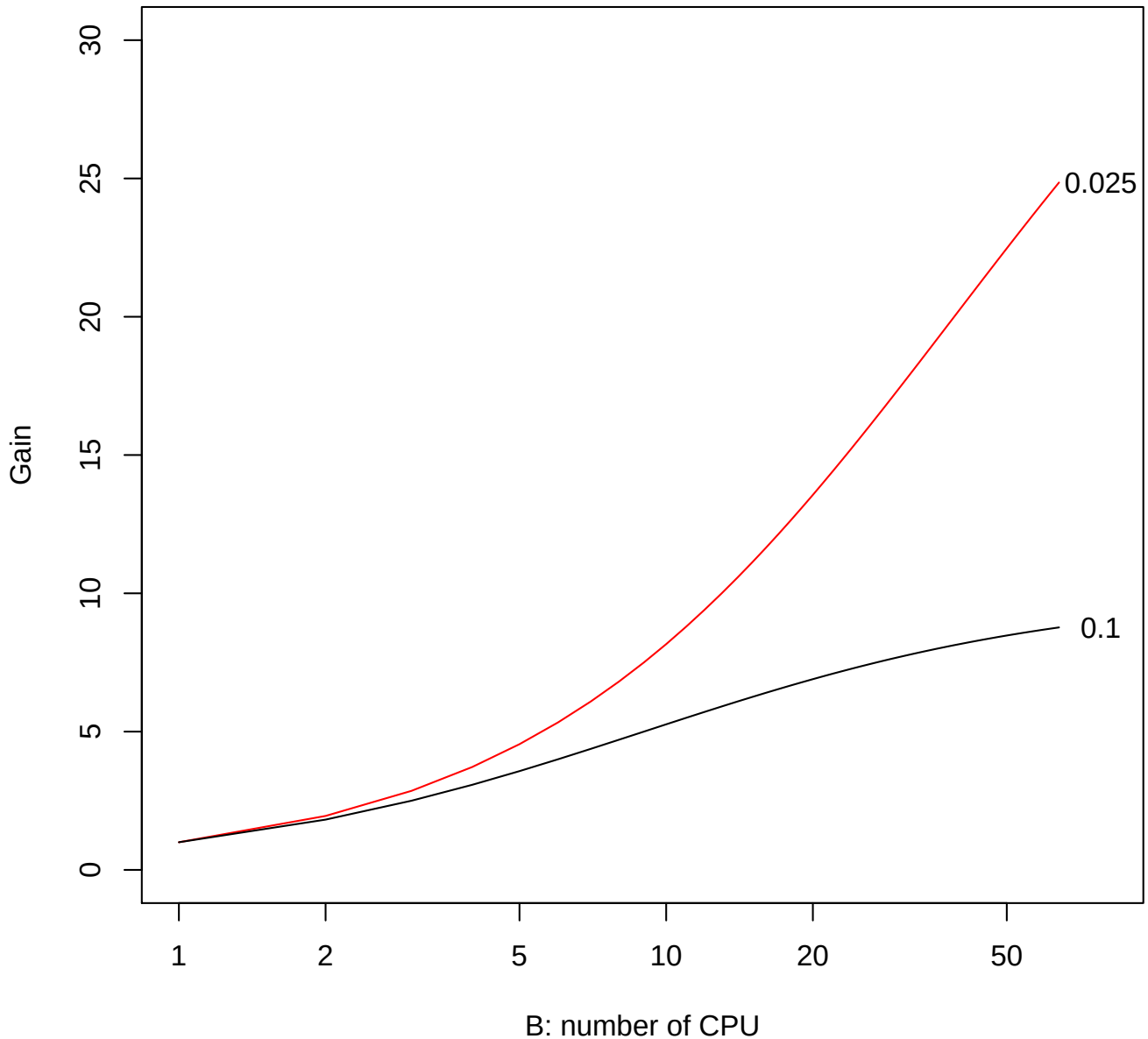
HW: the Mandelbrot set

- Use multi-threading to speed up the calculations
- Divide up the plane into 4 quadrants
- Submit each quadrant in a separate thread

Multi-threading and symmetric multi-processing

- Multi-threading and symmetric multi-processing are advanced technology that are surprisingly easy to use today
- Multi-threading emerged quite early in the digital computer age with the groundwork laid in the 1960s
- In 1961, Burroughs released the B5000 which was the first commercial hardware capable of multi-threading
- The B5000 performed asymmetric multiprocessing which is commonly employed in modern hardware like numerical co-processors and/or graphical processors today
- In 1962, Burroughs released the D825 which was the first commercial hardware capable of symmetric multiprocessing (SMP) with CPUs
- In 1967, Gene Amdahl derived the theoretical limits for multi-threading which came to be known as Amdahl's law
- If B is the number of CPUs and b is the fraction of work that can't be parallelized, then the gain due to multi-threading is $((1-b)/B+b)^{-1}$

Amdahl's law: $((1-b)/B+b)^{-1}$ where $b \in \{0.025, 0.1\}$



Multi-threading and symmetric multi-processing

- Hardware and software architectures in current use directly, and indirectly, led to wide availability of multi-threading today
- In 2000, Advanced Micro Devices (AMD) created AMD64 new 64-bit x86 instructions to coexist with 16-/32-bit x86
- An important advance since 64-bit math is capable of addressing vastly more memory than 16-/32-bit ($\mathbf{2^{64}}$ vs. $\mathbf{2^{16}}$ or $\mathbf{2^{32}}$) since multi-threading inherently requires more memory resources
- In 2003, version 2.6 of the Linux kernel incorporated full SMP support: prior Linux kernels had none/limited/crippled support
- From 2008 to 2010, Intel brought to market AMD64 Xeon chips with their hyper-threading technology that allows each core to issue two instructions per clock cycle: 4 cores (8 threads) in 2008 and 8 cores (16 threads) in 2010
- Today, most off-the-shelf hardware available features 1 to 4 CPUs each of which is capable of multi-threading: in the span of a few years, multi-threading rapidly trickled down from higher-end servers to mass-market desktops and laptops

Multi-threading with `parallel` package

- The `mcparallel` function uses *forking* to facilitate multi-threading (forking is NOT available on Windows)
- *Fork* is an operation where a process creates a copy of itself
- A *forked* R *child* process has memory address *pointers* to all of the objects known to the *parent* such as loaded packages, function definitions, data frames, etc.
- But, these *shared* objects are NOT copied into memory for each child: that would be a huge waste of resources!
- Each child has a memory address *pointer* to these objects
- However, R has a *copy on write* philosophy
- If a child writes to an object owned by the parent, a copy is made for the child while the parent retains the original
- This is convenient, but can be dangerous with multiple threads
- For example, if this is a big object, now that object has multiple instances which might consume a lot of memory

The `detectCores` function

- returns the number of threads that the computer is capable of
- the number of *threads* rather than the number of *cores* since they are not necessarily one-to-one
- `detectCores()` will occasionally return `NA`
- `detectCores(TRUE)` *might* fix that
- often, just calling `detectCores()` again will work
- however, this is an annoying bug/feature
- these are sporadic failures rather than reproducible failures
- reproducible failures can typically be *debugged*, but often sporadic cannot

The `mcparallel` function and nice

```
library(parallel) ## an example of multi-threading
library(tools)
for(i in 1:mc.cores) mcparallel({psnice(value=19); expr})
obj.list = mccollect()
...
```

- `expr` is processed `mc.cores` times each in their own threads

Paraphrasing the `psnice` documentation
Unix schedules processes to execute according to their priority. Priority is assigned values from 0 to 39 with 20 being the normal priority and (counter-intuitively) larger numeric values denoting lower priority. Adding to the complexity, there is a *nice* value: the amount by which the priority exceeds 20. Processes with higher nice values will receive less CPU time than those with normal priority. Generally, processes with nice value 19 are only run when the system would otherwise be idle to enhance system interactivity.

The `mccollect` function

- `mccollect` returns a list of return values from each thread
- in my experience, these are returned last in, first out (LIFO) the reverse from what we might have expected
- occasionally, a sporadic failure in one, or more, of the threads failed component(s) are missing from the list of return values
- if it is sporadic: re-running without any changes will succeed
- `class(obj)[1]!=type` is likely an error message so return it

```
obj.list = mccollect() ## last in, first out
obj = obj.list[[1]]
if(mc.cores==1 | class(obj)[1]!=type) {
    return(obj)
} else {
    m = length(obj.list)
    if(mc.cores!=m)
        warning(paste0("The number of items is only ", m))
    ...
}
```

The `mcparallel` function and random number generation

- We want each thread to have its own *stream* of random numbers that is reproducible
- There is a special random number generator for this purpose
- L'Ecuyer's combined multiple-recursive generator (CMRG)

```
library(parallel)
library(tools)
RNGkind("L'Ecuyer-CMRG")
set.seed(seed)
mc.reset.stream()
for(i in 1:mc.cores) mcparallel({psnice(value=19); expr})
```

PBS, TORQUE and SLURM

- In 1991, the Portable Batch System (PBS) started at NASA
- In 1998, OpenPBS was released as an open source version
- In 2003, many more enhancements of OpenPBS culminated in the Terascale Open-source Resource and QUEue Manager (TORQUE) (which was open source until 2018)
- In 2015, the Research Computer Center (RCC) TORQUE cluster debuted
- In 2016, the Division of Biostatistics TORQUE cluster debuted
- The RCC is transitioning to a new $1.4M turn-key Slurm cluster: on schedule for a January 2021 launch

Division of Biostatistics Cheese cluster

- 1 master node: gouda.biostat.mcw.edu
  Not for heavy computation: could crash the whole cluster
- 5 slave nodes:
  cheddar.biostat.mcw.edu, colby.biostat.mcw.edu, kingkong.pcor.mcw.edu, megatron.pcor.mcw.edu and savage.pcor.mcw.edu
- Interactive (colby): `$ qsub -I -X -l nodes=1:ppn=8`

Queue	Max Walltime Hours	Threads	Jobs/User
interactive	24	8	1
xxsmall	1536	4	10
xsmall	768	8	8
small	384	16	6
medium	192	32	4
large	96	64	3
xlarge	48	128	2
xxlarge	24	256	1

TORQUE bash shell script

```
#!/bin/bash
#PBS -l nodes=N:ppn=B     ## N for number of nodes
                          ## B for processes per node
#PBS -l mem=Mgb           ## M for memory
#PBS -l walltime=H:00:00  ## H for hours
cd $PBS_O_WORKDIR         ## move to current directory
module load R/V           ## V is the R version
time R --no-save < name.R >& name.Rout
```

- The setting `N` is typically 1 for R jobs
- The setting `B` is typically 8 due to Amdahl's law
- The setting `B` determines which queue your job goes to
- Each queue has a max walltime that you can use for `H`
- The setting `M` is based on the size of the objects that you have which you can check with `> object.size(object)`
- The setting `V` is typically the latest/greatest: now 3.6.2

TORQUE commands and documentation

- To submit a bash shell script TORQUE job:
  `$ qsub name`
  `548.gouda.biostat.mcw.edu`
- To kill the job: `$ qdel 548`
- And monitor its progress: `$ qstat 548`
- There are man pages for these `qCOMMANDS`
- And for `#PBS -l` resource list options:
  `man pbs_resources_sunos4`
  Not a typo: apparently this naming convention comes from an early open source implementation on UNIX
- More information available at
  `https://wiki.biostat.mcw.edu`