julia> Lecture Six: Parallelization and UW Resources

Course: Computational Bootcamp
Name: John Higgins
Date: July 3, 2024

julia> Today's goals

1. Understand what parallel computing is and the basics of how it works
2. Understand how to utilize it to assist with research problems

# julia> Parallelization

- Central Processing Unit: Executes instructions of a computer program
- Modern CPUs have multiple processor cores (“processes”)
  - Different processes can execute different tasks at same time (multi-processing)
  - Each core has its own memory
- Modern cores/CPU execute several tasks at the same time (multi-threading)
  - Each task is called a thread
  - Threads will share memory
- Intuition:
  - Multi-threading: multiple people (cores) baking pizza (executing code) in the *same* kitchen (CPU)
  - Multi-processing: multiple people (cores) baking pizza (executing code) in *separate* kitchens (CPU)

julia> Multi-processing or multi-threading?

- When is multi-threading better than multi-processing?
- Multi-threading:
  - All workers share same memory
  - Faster set-up and faster to share memory across workers
  - Need to be careful when workers edit same memory (“data race”)
  - All workers need to be on same computer/server
- Multi-processing:
  - All workers have their own memory (kitchen, ingredients)
  - Slower to set up and share memory
  - Need to be careful about what information each worker needs
  - Workers can exist across computers/servers

# julia> Multi-processing or multi-threading?

- General rule of thumb:
  - If each task is relatively quick/small, multi-threading may be more efficient
  - If each task is quite intensive, multi-processing may be more efficient
- To belabor the pizza analogy:
  - It is cheaper to only run one kitchen (less equipment, less duplication), but can be inefficient (limited cooking space, limited ovens, more congestion)
  - It is more expensive to operate multiple kitchens, but each kitchen operates independently and is less congested

julia> What does this mean for you?

> Multi-processing and multi-threading allow you to complete tasks in parallel and speed up your code if done well
> So far, you are not taking full advantage of computation resources available to you
> Your laptop likely has several cores that can have two threads of execution
> UW-Madison has a lot of computing power available for you to use

julia> Social Science Computing Cooperative

> https://sscc.wisc.edu/sscc/pubs/grc/resources.html
  1. Linstat: 4 interactive Linux servers (
     > Each server has 36 or 48 cores, 500GB of memory
     > Good for checking that your code actually works on SSCC's servers before submitting to Slurm
  2. Slurm: 40 Servers with a total of 5,000 cores
     >
     > Most servers have 128 cores and at least 250GB of memory (1,024GB)
     > You can let your code run for up to 10 days

> Be mindful that other people need to use Linstat and Slurm, too

> SSCC has other resources for jobs that need lots of memory (Winstat BIG) and jobs that use confidential data (SILO)

> If you want to use the power of Linstat in an interactive environment like a Windows/Mac, you can use the SSCC's new Open OnDemand service: https://kb.wisc.edu/sscc/page.php?id=131429

julia> Center for High Throughput Computing

> https://chtc.cs.wisc.edu/
> High Performance Computing (HPC): Multi-processing tasks
> High Throughput Computing Cluster (HTC): Multi-threading tasks
> I am not so familiar with these resources

julia> Example: Solving capital savings

> Value function iteration to solve for $V$:

$$V_{n+1}(z,k) = \max_{c,k'} \{\log(c) + \beta E_{z'}[V_n(z',k') \mid z]\}$$

$$c + k' = zk^{\alpha} + (1-\delta)k$$

> Solving on some capital grid $\{k_1, \ldots, k_n\}$

> To use parallelization, we can ask different cores/workers to solve for $\{k'(z,k), c(z'k), V_{n+1}(z'k)\}$ at different initial $k$

> Then, we gather the information together and complete the next iteration

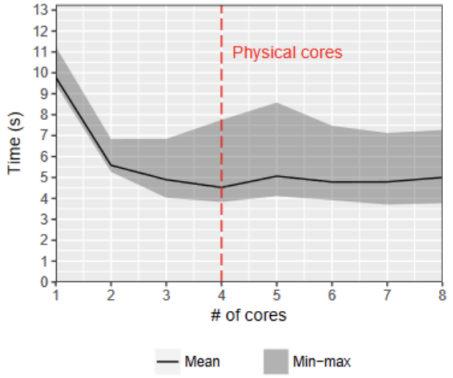
## julia> Example: Simulating capital savings

- Now we have solved for $V$
- Suppose we want to simulate lots of histories of $c$ and $k$
- We generate a bunch of random paths for productivity $\{z_t\}$
- Simulate paths of $c$ and $k$
- Instead of doing this all on one core, we can simulate many different paths at the same time using parallel computing

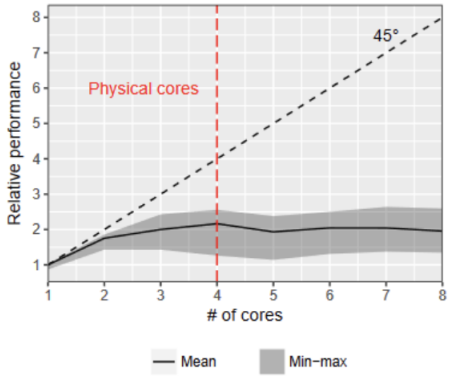
# julia> No free lunch

- Parallelization is not a free lunch because of over-head costs
- Having many cores running uses more memory
- Sharing memory across cores takes time
- If you are doing lots of calculations that take a long time to run, parallelization can help
- There are also diminishing returns that can kick in quickly for smaller problems

julia> Simple savings model with many cores




(a) Computing time (s)




(b) Performance gains.

Figure 8: Results in Julia with different number of processors. Number of experiments: 20.

https://www.sas.upenn.edu/~jesusfv/Guide_Parallel.pdf

## julia> Accessing Linstat and Slurm

- The easiest way to log in to Linstat/Slurm is to first log on to Winstat
- You should have made a Winstat account/password at first-year orientation
- https://kb.wisc.edu/sscc/using-winstat
- You can log into Winstat with Citrix Workspace or at the following website. Note: Web log-in cannot access local files. (https://winstat.ssc.wisc.edu/vpn/index.html)
- If you need to reset your password: https://www.sscc.wisc.edu/sscc_jsp/password/reset.jsp
- Note: you can also log into Linstat directly from your computer
  - https://sscc.wisc.edu/sscc/pubs/grc/linux.html
  - On Windows, need to install X-Win32
  - On Mac, can connect directly using command line
  - On either OS, you will need to use the SSCC VPN (https://kb.wisc.edu/sscc/connecting-to-the-sscc-network-via-vpn)

# julia> Linstat Cheat Sheet

> Open interactive Julia REPL:

```
julia
```

> Run a Julia script file (titled ‘‘script.jl’’) and save output to text file (titled ‘‘output.txt’’):

```
julia script.jl > output.txt &
```

> Change working directory to a specified folder (given by path ‘‘FolderPath’’ - e.g. FolderPath = ‘/project_folder/code/model/’):

```
cd FolderPath
```

julia> Linstat Cheat Sheet (continued)

> View all files in current folder:

```
ls
```

> Go up one level in the filesystem (e.g. if we're in '/project_folder/code/model', we may want to go to 'project_folder/code'):

```
cd ..
```

> More comprehensive list:
https://gist.github.com/khazeamo/f762f532bfbc17d5bf396e9d4c2a9586

> Note: alternatively, you can access and navigate your Linux directory through Winstat (it is located in your Z drive)

# julia> Linstat Cheat Sheet (continued)

> See jobs running on current Linstat server:

```
top
```

> To close top, press “q”
> Stop a job PID that is currently running:

```
kill PID
```

# julia> Slurm Cheat Sheet

> To run a script file on Slurm with $N$ cores and $M$ GB of memory:

```
ssubmit --cores=N --mem=Mg "julia_script.jl"
```

> To figure out how much memory you need:
>
> 1. Run your job on Linstat and check its memory use (RES)
> 2. Then, kill your job and send it to Slurm

> Slurm will send an email when your job is done running (either due to success or because it failed)

> You can submit to specific partitions of the server using the option ‘‘-part’’

julia> Slurm Cheat Sheet

> Note: Most pre-compiled Julia package images won't work directly on Slurm; this means you will have to include the ''-pkgimage=no'' command in order to get your code to run:

```
ssubmit --cores=C --mem=Mg "julia --pkgimage=no my_julia_script.jl"
```

> To see jobs currently running on Slurm, use:

```
squeue -a
```

> To cancel your job:

```
scancel JOBID
```

[~]$ _

# julia> Linstat vs. Slurm

> Linstat is good for:
>
> 1. Interactive coding
> 2. Checking that your code runs on SSCC servers
> 3. Seeing how much memory you are using
> 4. Sending jobs to Slurm

> Slurm is goo for big jobs that need lots of cores and run for a long time