Parallelization, Multi-threading, and Multi-processing

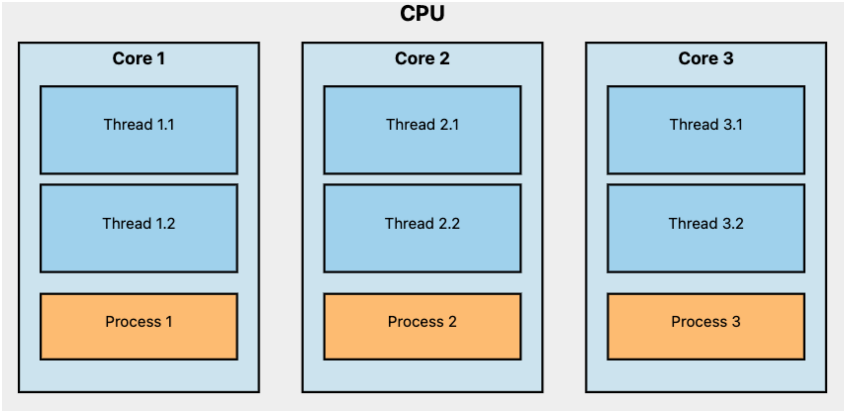
John Ryan
john.p.ryan@wisc.edu

September 5, 2025

Introduction to Parallelization

- Parallelization: Executing multiple tasks simultaneously
- Goal: Improve performance and efficiency
- Two main approaches:
  - Multi-threading
  - Multi-processing

CPUs - Cores and Threads



Multi-threading

- Allows a single process to execute multiple tasks concurrently across threads
- Logical threads: Lightweight units of execution within a process
- Share the same memory space
- Ideal for tasks that require frequent communication
- Challenges: Race conditions, deadlocks

Multi-processing

- Processes: Independent units of execution
- Each process runs on different CPU cores
- Each process has its own memory space
- Ideal for CPU-intensive tasks with minimal inter-process communication
- Can scale higher
  - Slurm has 40 servers with total 5,000 cores
  - See SSCC for distributed computing options
- Challenges: Higher overhead, complex data sharing

Julia's Distributed Package

- Provides tools for distributed computing
- Key features:
  - Remote references and remote calls
  - Parallel map and loops
- ‘@distributed‘ & ‘@everywhere‘ macros or ‘pmap‘ function do most of the work
- Use SharedArrays package for sharing arrays across procs

Codealong

Neoclassical Growth Model: Sequence Problem

The Social Planner faces the following optimization problem:

$$\begin{aligned} \max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} & \sum_{t=0}^{\infty} \beta^t u(c_t) \\ \text{s.t. } & c_t + k_{t+1} = A k_t^{\alpha} + (1-\delta) k_t \\ & c_t, k_{t+1} \geq 0 \\ & k_0 > 0 \text{ given} \end{aligned}$$

where $\alpha, \beta \in (0,1)$, $\delta \in (0,1]$, $A > 0$, $u'(\cdot) > 0$, $u''(\cdot) < 0$ and satisfied Inada conditions.

Neoclassical Growth Model: Dynamic Programming

- The Recursive Formulation:

$$V(k) = \max_{c,k'} \{u(c) + \beta V(k')\}$$
$$\text{s.t. } c + k' = Ak^{\alpha} + (1-\delta)k$$

- The Bellman operator T is defined as:

$$TV(k) = \max_{k'} \{u(Ak^{\alpha} + (1-\delta)k - k') + \beta V(k')\}$$

- The fixed point of this operator is the solution to our dynamic programming problem

Contraction Mapping

- The Bellman operator is a contraction mapping
- Blackwell's sufficient conditions:
  - Monotonicity: If $V \leq W$, then $TV \leq TW$
  - Discounting: $T(V + a) = TV + \beta a$ for constant a
- These properties guarantee:
  - Existence and uniqueness of a fixed point
  - Convergence of value function iteration

Value Function Iteration Algorithm

1. Initialize a grid of possible capital values $K_{grid} = \{k_0, k_1, ..., k_N\}$
2. Choose an initial guess $V_0(k)$ over K_{grid} and a tolerance level ϵ
3. For iteration n, use Bellman operator:

$$V_{n+1}(k) = \max_{k'} \{u(Ak^{\alpha} + (1-\delta)k - k') + \beta V_n(k')\}$$

4. Check if $\|V_{n+1} - V_n\| < \epsilon$
   - If yes, stop
   - If no, set $n = n + 1$ and go to step 3
5. The resulting V_{n+1} is an approximation of the true value function. The argmax of Bellman is the policy function $k'(k)$

Bellman Operator Algorithm: Grid Search

- Discretize choice set and compute the objective value at each grid point
  - The slowest, but simplest and stablest method - brute force

$$V_{n+1}(k) = \max_{k'}\{u(Ak^{\alpha} + (1-\delta)k - k') + \beta V_n(k')\}$$

1. Take as given $V_n(k)$, model parameters β, α, δ, etc.
2. Initialize V_{n+1} with a low guess $(-\infty)$
3. For each element $k_i \in K_{grid}$:
   - For each element $k'_j \in K_{grid}$:
     - Compute $\tilde{V}_i(k'_j) = u(Ak_i^{\alpha} + (1-\delta)k_i - k'_j) + \beta V_n(k'_j)$
   - $V_{n+1}(k_i) := \max_{k'_j \in K_{grid}} \tilde{V}_i(k'_j)$

Stochastic Neoclassical Growth Model

Extension to include TFP (productivity) shocks:

$$V(k,z) = \max_{k'}\{u(c) + \beta\mathbb{E}[V(k',z')|z]\}$$
$$\text{s.t. } c + k' = z * k^{\alpha} + (1-\delta)k$$

- where z is the productivity shock, which follows some Markov process.
- We use Tauchen's method & assume that z follows a discrete state Markov chain
- Value function and policy functions will be a matrix now, rather than a vector
- Have to loop over productivity and capital grids
  - Hint: parallelize on z, computed expected value function