Programming Language Basics, Parallelism & Concurrency

Presentation by: Sam Barton, Chad Davidson, Matt MacNeil

Overview

- JIT (Just In Time) Compilation
- Free and Open Source
- Scientific Computing
- Designed for parallelism and distributed computation

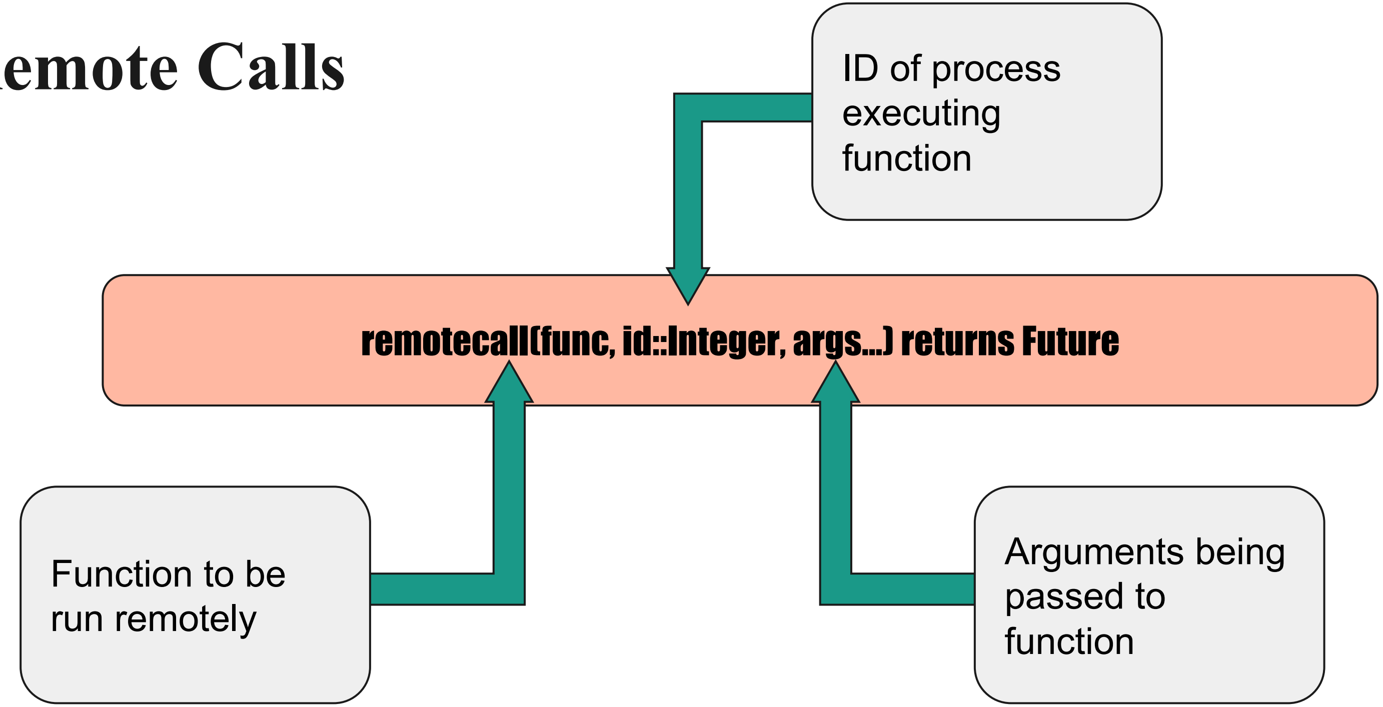
Interaction

- Can directly call C or Fortran functions without a wrapper
- Has support for unicode
- Can perform read–eval–print loop (REPL) in an interactive session shell
- Has a source-to-source compiler that allows it to be compiled to c code for better cross-platform compatibility (Julia2C)

Parallel

- Generally, communication is “one-sided”
    - Target of communication is not involved
- Built on two primitives:
    - Remote References
    - Remote Calls

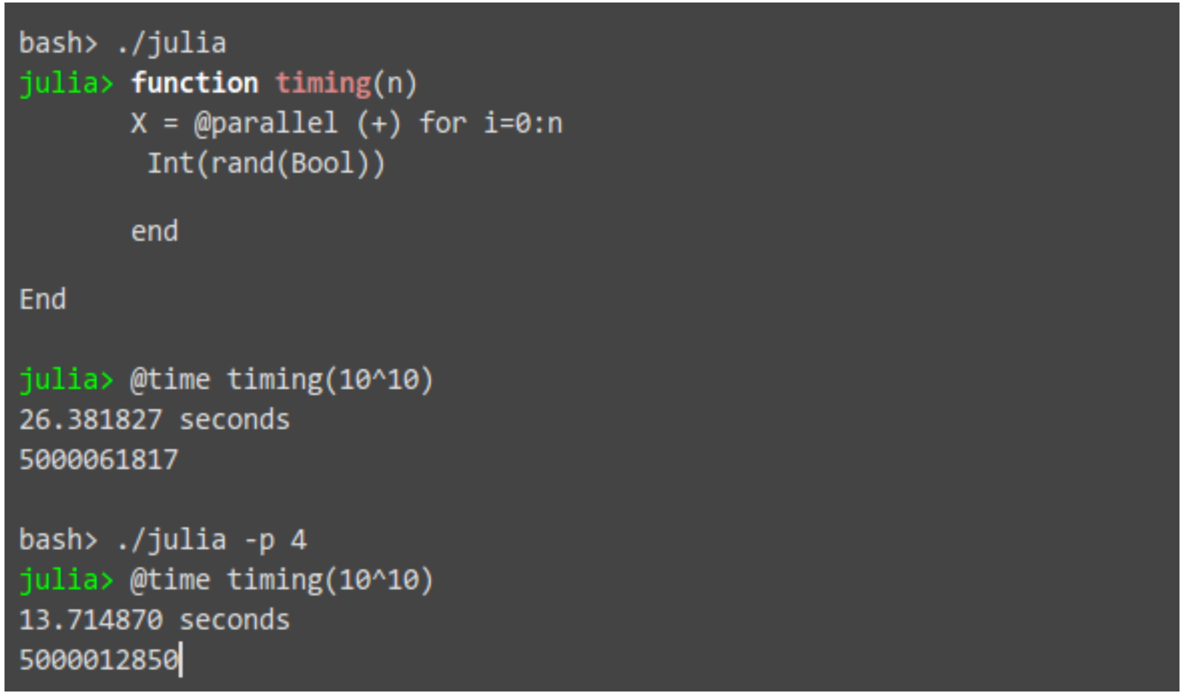
Remote Calls


ID of process executing function
remotecall(func, id::Integer, args...) returns Future
Function to be run remotely
Arguments being passed to function

Macros

- Macros may be used instead of Remote Calls
- @spawn macro will automatically choose an available processor as opposed to Remote Calls where process must be explicitly stated
- @everywhere macros will execute an expression on all processes
- @parallel
- @sync macro placed before an @parallel macro will wait for all processes to finish before the parallel region ends
- @async is similar to @spawn but only runs tasks on the local process

Example Parallel Function



```
bash> ./julia
julia> function timing(n)
       X = @parallel (+) for i=0:n
        Int(rand(Bool))

       end

End

julia> @time timing(10^10)
26.381827 seconds
5000061817

bash> ./julia -p 4
julia> @time timing(10^10)
13.714870 seconds
5000012850
```

Where to go for more information

If you enjoyed this presentation or just want to learn more

https://julialang.org/

http://julia-wf.readthedocs.io/zh_CN/latest/stdlib/parallel.html

https://arxiv.org/pdf/1209.5145.pdf