**Проба распараллеливания**

Описание функций, приведенных ниже, более подробно можно изучить по ссылке - <https://docs.julialang.org>

**Вспомогательная функция для тестирования**

Для тестирования распараллеливания реализована функция:

function WaitAndRetunArray(sec, N)

Array = rand(N)

sleep(sec)

return(Array)

end

WaitAndReturnArray – ждет установленное количество секунд и возвращает вектор из N элементов.

Предполагается, за установленное время проводить какие-либо исследования поведения распараллеливания в Julia.

Важно, что импортировать фунцию WaitAndReturnArray нужно при помощи макроса @everywhere. Иначе другие процессы не будут знать о существовании функции.

Если функция WaitAndReturnArray описана в файле “WaitFunc.jl”, то импорт будет выглядеть следующим образом:

@everywhere include("waitFunc.jl")

**Описание исследуемых функций в Julia**

Каждый процесс в Julia имеет свой идентификатор id. Процесс в котором пользователь осуществляет взаимодействие с Julia (интерактивная среда) всегда имеет id = 1.

workers() – возвращает массив содержащий массив номеров (id) доступных процессов Julia;

addprocs(n) – добавляет n процессов Julia. Если выполнить без аргументов – добавляет столько процессов, сколько ядер на процессоре пользователя;

rmprocs(id) – удаляет процесс по id;

@everywhere – макрос, исполняет команду на всех доступных процессах;

remotecall(func, id, funcArg1, funcArg2, …) – удаленно, на процессе с индексом id, вызывает функцию func() с аргументами funcArg (аргументы функции перечисляются через запятую) и возвращает Future, как ссылку на будущий результат удаленных вычислений хранящихся на удаленном процессе.

Важно, что в коде программы, после того, как выполнится remotecall, программа переходит на следующую строчку кода, без ожидания результата расчета;

fetch(Future) – получает результат вычислений при помощи Future. Если вычисления ещё продолжаются, программа ожидает результата;

remotecall\_fetch(func, id, funcArg1, funcArg2, …) – эквивалентно fetch(remotecall(func, id, funcArg1, funcArg2, …)) , но, по заявлению разработчиков, работает эффективнее;

**Проба распараллеливания матричного умножения**

Реализована функция матричного умножения. По результатам тестов реализованная функция не уступает по времени работы встроенному в Julia матричному умножению.

Попробуем распараллелить на процессы.

@time multiple(A,B)

7.737250 seconds (6 allocations: 7.630 MiB, 0.06% gc time)

1000×1000 Array{Float64,2}.

**Установка пакета PyPlot**

Pkg.add(“PyPlot”)

Возможна ошибка при использовании команды “plot”:

R6034: с++ runtime…

Исправление:

ENV["PYTHON"]=""

Pkg.build("PyCall")