3. Лабораторная работа №1 язык Julia

3.1. Предварительные пояснения

Замер времени работы не такая тривиальная вещь, какой кажется на первый взгляд. В языке Julia она еще осложняется тем, что стадия компиляции скрыта от пользователя. Она происходит при первом запуске программы и может занимать много времени, поэтому для правильного замера времени работы, следует придерживаться следующих правил.

- Выделить код, производительность которого вы хотите замерить, в отдельную функцию.
- Всегда замерять работу данной функции, а не все программы целиком.
- Перед замером вызвать функцию как минимум один раз, чтобы компилятор заранее скомпилировал ее код.

3.2. Задание №1

Любая программа, использующая параллельные вычисления, имеет смысл только если она работает быстрее, чем программа, делающая тоже самое, но последовательно. Так что замерять время выполнения крайне важно. Прежде, чем замерять время работы параллельных программ, надо научится правильно замерять время работы последовательных программ. В Julia задача облегчается наличием готовых функций и макросов для этих целей.

- Напишите простейшую подпрограмму, которая ждет некоторое время, например 1 секунду. Для ожидания используете встроенную функцию `sleep`. Можно ли передавать ей дробные значения? Прочтите справку функции `time_ns` и замените ею функцию `sleep`.
- Замерьте время работы функции 10 раз, распечатывая результаты. Используйте `@time`. Время выполнения рознится или все время одинаковое?
- Сделайте замеры времени для 10^6 запусков. Ждать одну секунду уже не получится, поэтому замените `sleep` на `time_ns`. Замерьте время работы программы, распечатайте в виде облака точек (scatter) и гистограммы.

Для замеров времени можно использовать `BenchmarkTools`, который может строить гистограммы времени работы функций, но нужно разобраться как он работает.

3.3. Задание №2

В Julia нет реализации OpenMP, однако концепции, положенные в основу многопоточности и в Julia и в OpenMP общие.

- Создайте программу которая порождает потоки. Распечатайте количество созданных потоков. С помощь какой функции это делается?
- Julia позволяет задавать количество потоков в момент старта. Есть несколько способов это сделать: с помощью переменной окружения и с помощью параметра командной строки. Создайте примеры, иллюстрирующие оба способа. Запустите программу несколько раз, всякий раз изменяя количество потоков и проверяя, что потоки запущены. Возможно пригодится функция `sleep()`.
- Сколько потоков эффективно поддерживает ваш процессор? Как можно узнать эту информацию? Как в Julia можно автоматически создать такое количество потоков, которое процессор поддерживает оптимально?
- Создайте многопоточную программу с четырьмя потоками, которая принимает на вход массив целых чисел. Нужно вручную распределить работу между потоками. Первый поток должен просуммировать 1, 5, 9 и т.д. числа; второй поток — 2, 6, 10 и т.д.; третий — 3, 7, 11; четвертый — 4, 8, 12 и т.д. Результаты суммирования распечатываются с указанием, какой поток какой результат получил.

- Напишите автоматические тесты для данной программы, которые проверяют ее работоспособность для разных последовательностей чисел.

3.4. Задание №3

Все встроенные математические функции языка Julia поддерживают векторные действия с массивами, то есть если им в качестве аргумента передать не скалярное значение, а массив, то функция будет вычислена от каждого элемента массива. Для этого после названия функции нужно указать точку. Например, если мы хотим вычислить синус от каждого элемента массива `A`, то должны вызвать функцию так: `sin.(A)`.

Потенциально, такие функции могут использовать SIMD регистры процессора, что должно давать выигрыш в производительности по сравнению с вычислениями в цикле.

В качестве третьего задания придумайте некоторую свою функцию, которая манипулирует числовым массивом и вычисляет какое-то значения. Составьте ее из элементарных математических функций и вычислите ее значение от массива двумя способами.

- Первый способ заключается в вычислении значений функций от элементов массива в цикле, передавая каждый элемент массива в функцию по отдельности.
- Второй способ заключается в передаче всего массива в виде аргумента.

Второй способ теоретически должен дать выигрыш в производительности. Замерьте время выполнения с помощью `@time`.

Прочитайте в официальной документации о макросах `@inbounds`, `@fastmath` и `@simd` и примените их в своем цикле. Обязательно проверьте корректность вычислений при использовании `@fastmath` (как это сделать?)

3.5. Задание №4

Julia обладает большой коллекцией встроенных функций, которые позволяют писать код в функциональном стиле. Написанный в таком стиле код очень хорошо поддается параллелизации, так как используемые функции являются чистыми и не должны модифицировать свои аргументы, что предотвращает гонку данных.

- Изучите функцию `reduce`. Расскажите своими словами как она работает.
- Создайте небольшой массив целых чисел, такой, чтобы можно было проверить корректность вычислений. С помощью `reduce` сделайте с ним следующие действия.
  - Найдите все положительные числа, отрицательные числа, четные, нечетные, делящиеся без остатка на 7.
  - Затем с получившимися в результате такой фильтрации последовательностями проделайте следующие операции: просуммируйте, найдите максимум, минимум, среднее, выборочную дисперсию.
- Попробуйте сразу создать массив с перечисленными выше условиями, то есть например такой, который состоит из целых чисел, делящихся без остатка на 7.
- Объедините все условия вместе, то есть например найдите сумму всех элементов, которые делятся на 7 без остатка, при этом положительные, не больше какого-то числа.
- Задание можно выполнить множеством способов. Защита лабораторной пройдет проще, если будут использованы анонимные функции, оператор `|>` и функции из модуля `Base.Iterators`.

Вдобавок, каждое задание должно быть реализовано в одну строчку. То есть например создание массива, нахождение всех положительных чисел и их суммирование должно занимать одну строку кода. Точки с запятой стараться не использовать.