УДК 004.91

ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ GO

В. С. Маширова
Донской государственный технический университет (г. Ростов-на-Дону, Российская Федерация)

Современные процессоры оснащаются все большим количеством CPU-ядер, чтобы удовлетворить растущие потребности в вычислительной мощности. К сожалению, многие наши программы не используют имеющиеся ресурсы должным образом. Для этого необходимо писать параллельные программы. Язык программирования Go, язык с открытым исходным кодом, разработанный компанией Google, обещает сделать параллельное программирование намного проще и надежнее.

Ключевые слова: Go, Golang, параллелизм, параллельное программирование, многопоточность.

PARALLEL PROGRAMMING IN GO

V.S. Mashirova
Don State Technical University (Rostov-on-Don, Russian Federation)

Today's processors are equipped with ever-growing amount of CPU cores to meet our growing need for computing power. Even though this is not a completely new development, many of our programs do not use the available resources properly. This requires us to write parallel programs. The Go programming language, an open-source language developed by Google, promises to make parallel programming much easier and more reliable.

Keywords: Go, Golang, Concurrency, Concurrent Programming, Multithreading.

Введение. Что такое параллельное программирование? Наши процессоры становятся все мощнее и мощнее. Чтобы высокой мощности достичь, производители создают процессоры со все большим количеством ядер. Это вызвано физическими ограничениями, которые не позволяют бесконечно увеличивать вычислительную мощность одного ядра. Но такие процессоры могли бы выполнять несколько вычислений параллельно [1]. Проблема лишь в том, что классические однопоточные программы не позволяют этого делать. Они состоят из инструкций, которые могут выполняться только последовательно на одном ядре. Чтобы использовать весь потенциал оборудования, нужно писать параллельные программы. По крайней мере, часть инструкций таких программ может выполняться несколькими ядрами параллельно. Здесь важно отметить разницу между параллелизмом и параллельностью, поскольку эти слова часто используются как синонимы. «Параллелизм обеспечивает способ структурирования для решения проблемы, которая может (но не обязательно) быть параллелизуемой» [2]. Исходя из этого определения, можно заключить, что параллельная программа структурирована таким образом, что определенные задачи могут выполняться параллельно. Но если доступно только одно ядро, задачи будут выполняться последовательно. Цель данной статьи — рассмотреть использование языка программирования Go для создания параллельного, более эффективного и простого параллельного программного обеспечения.

Основная часть. Одновременное программирование в классических языках. В большинстве языков программирования параллелизм достигается с помощью потоков. Потоки — это ресурс, предоставляемый операционной системой. По сути они представляют собой легковесные процессы. В отличие от процессов, потоки имеют доступ к общей памяти. Один процесс может иметь несколько потоков. Использование потоков может ускорить работу программы, но тут есть и некоторые трудности.

Несмотря на то, что потоки являются легкими, по сравнению с процессами, они все же являются дорогостоящими ресурсами, и многие операционные системы ограничивают количество доступных потоков на один процесс. Чаще всего программисты сталкиваются с ситуацией, когда есть несколько небольших задач, которые могут выполняться параллельно. Одним из возможных решений является запуск потока для каждой из этих задач. Но поскольку запуск нового потока сопровождается значительными накладными расходами, такое решение может быть даже медленнее последовательной реализации. Многие библиотеки включают Thread-Pool для решения этой проблемы. Thread-Pool содержит набор рабочих потоков, которые выполняют задания из очереди [3]. Это решает описанную проблему, но вводит новую. Если задача, например, посылает сетевой запрос и ждет ответа, она фактически блокирует работу одного из рабочих потоков. Если таких задач много, это может даже заблокировать весь Thread-Pool. В реальных приложениях чаще всего недостаточно иметь потоки, которые выполняют совершенно независимые задачи. Рано или поздно этим потокам понадобится способ взаимодействия. В классических многопоточных приложениях для этого используется общая память. Это означает, что потоки читают и записывают информацию из и в некоторые переменные, и структуры данных в общем пространстве памяти. Такой подход может привести к проблемам, если два потока одновременно записывают в один и тот же объект и, возможно, оставляют его в противоречивом состоянии. Это так называемое состояние гонки. Один из разработчиков концепции структурного программирования, исследователь формальной верификации и распределённых вычислений Эдсгер Вибе Дейкстра описал решение этой проблемы [4]. Поток может получить блокировку перед доступом к объекту. Это блокирует доступ всех других потоков к объекту до тех пор, пока блокировка не будет снята. Но это решение также имеет свои недостатки. Если поток хочет получить доступ к объекту, который в данный момент заблокирован, ему приходится ждать, пока блокировка не будет снята. Если такое происходит очень часто, то это замедляет работу программы. Другая проблема возникает, если есть несколько потоков, которым нужно заблокировать несколько объектов. Они могут оказаться в ситуации, когда каждый поток ждет, пока остальные освободят свои блокировки, что приведет к мертвой блокировке.

Подобные факты показывают, почему работа с потоками может быть сложной. Это привносит в программу много дополнительных сложностей и потенциальных ошибок. Именно поэтому многие программы не пишутся параллельно и, следовательно, не используют весь потенциал машины, на которой они работают. Многие программисты прибегают к многопоточному решению только в том случае, если оно дает значительные преимущества в производительности.

Одновременное программирование в Go. Разработано множество различных подходов к параллельному программированию. Язык программирования Go претендует на то, чтобы облегчить создание параллельного программного обеспечения и, следовательно, сделать его более эффективным. Go имеет открытый исходный код и активно разрабатывается командой Google [5]. Он статически типизирован и компилируется в нативный код.

Go предоставляет Goroutines для выполнения параллельных задач. Любая функция может быть параллельно выполнена как Goroutines, если добавить ключевое слово Go перед вызовом. Поскольку Goroutines очень эффективны, по сравнению с потоками операционной системы, даже небольшие задачи могут извлечь из этого выгоду. Под Go используют планировщик и пул потоков операционной системы. Но, в отличие от других пулов потоков, блокировка Goroutines не приводит к блокировке дорогостоящего потока операционной системы. Вместо этого поток продолжает выполнение других Goroutines, которые в данный момент не заблокированы. Как и

потоки, Goroutines также имеют доступ к общей памяти, а Go предоставляет блокировки для защиты объектов в общей памяти так же, как это делают классические языки программирования.

Общая память полезна во многих случаях, но Go предоставляет еще и другой способ обработки связи и синхронизации между Goroutines. Он основан на модели параллелизма Communicating Sequential Processes (CSP), описанной специалистом в области информатики и вычислительной техники, разработчиком алгоритма «быстрой сортировки» Ч. Э. Р. Хоаром [6]. Вместо того чтобы добиваться параллелизма путем наличия нескольких параллельных задач, которые изменяют одни и те же общие объекты, он предлагает модель независимых агентов. Каждый агент является последовательным процессом и имеет свою собственную память, которая не делится ни с одним из других агентов. Это устраняет необходимость в блокировке. Агенты общаются, посылая друг другу сообщения. Процесс отправки и получения сообщений по умолчанию является синхронным, поэтому отправитель сообщения ждет, пока оно будет получено, а получатель также ждет, пока не будет получено сообщение. Эти сообщения используются не только для коммуникации, но и для синхронизации. Преимущество этого подхода заключается в том, что агент может быть реализован как последовательная программа, без каких-либо трудностей, связанных с общей памятью и блокировкой. Общение и синхронизация осуществляются на более высоком уровне с помощью сообщений. В Go агент представляет собой Goroutine. Сообщениями можно обмениваться с помощью каналов. Канал — это блокирующая очередь сообщений с нулевым размером по умолчанию, которая встроена в язык [7].

Заключение. Go предлагает поддержку классического параллелизма с использованием общей памяти и блокировок. Таким образом, параллельные программы могут быть написаны так же, как и на классическом языке программирования с использованием потоков. Но Go избавляет от необходимости использовать Thread-Pools и беспокоиться о заблокированных потоках. Go также предлагает и вторую модель параллелизма, которая может оказаться даже более подходящей для некоторых случаев использования. Она устраняет трудности в работе с блокировками. Кроме того, Go упрощает параллелизм на синтаксическом уровне, делая параллельные программы более читаемыми. Можно сказать, что Go удалось сделать параллельное программирование более простым, чем оно было раньше в классических языках программирования.

Библиографический список

1. Язык Go для начинающих / Хабр : [сайт]. — URL: https://habr.com/ru/post/219459/ (дата обращения: 27.02.2022).

2. Donovan A. A. The Go Programming Language. / A. A. Donovan, B. W. Kernighan. — Addison-Wesley Professional. 2015. — 380 p.

3. Go / Википедия : [сайт]. — URL: https://ru.wikipedia.org/wiki/Go (дата обращения 27.02.2022).

4. Dijkstra E. W. Solution of a Problem in Concurrent Programming Control: Commun. / E. W. Dijkstra — ACM. 1965. — 569 p.

5. Go Programming Language / Xoriant. // geeksforgeeks.org [сайт]. — URL : https://www.xoriant.com/blog/product-engineering/go-programming-language-key-features.html (дата обращения 27.02.2022)

6. Hoare A. R. Communicating Sequential Processes: Commun. / A. R. Hoare — ACM. 1978. — Pp. 666-677.

7. Introduction to Concurrency / Golangbot. [сайт] — URL : https://golangbot.com/concurrency/ (дата обращения 27.02.2022)

Об авторе:

Маширова Вероника Сергеевна, магистрант кафедры «Медиатехнологии» Донского государственного технического университета (344003, РФ, г. Ростов-на-Дону, пл. Гагарина, 1), veronika_mashirova@mail.ru

About the Author:

Mashirova, Veronika S., Master's degree student, Department of Media Technology, Don State Technical University (1, Gagarin Square, Rostov-on-Don, 344003, RF), veronika_mashirova@mail.ru