Goメモリモデルの話

TODO

C++のメモリモデル論文を読む
メモリモデルドキュメントの同期の例を使ったストーリーにする

資料

- The Go Memory Model - The Go Programming Language Go1.19メモリモデル
    - 参考論文 Foundations of the C++ Concurrency Memory Model 次これ読む
- The Go Memory Model - The Go Programming Language Go1.18メモリモデル
- https://go-review.googlesource.com/c/go/+/381315
-

rscの論文

- research!rsc: Memory Models
    - research!rsc: Hardware Memory Models (Memory Models, Part 1)
    - research!rsc: Programming Language Memory Models (Memory Models, Part 2)
    - research!rsc: Updating the Go Memory Model (Memory Models, Part 3)

GitHub Discussion

Updating the Go memory model · Discussion #47141 · golang/go · GitHub

proposal 1

- doc: update Go memory model · Issue #50859 · golang/go · GitHub

proposal 2

- sync/atomic: add typed atomic values · Issue #50860 · golang/go · GitHub

にほんご記事

- リトマステスト早見表
- Go Memory Model (1.19)
- https://twitter.com/mattn_jp/status/1552511929381314562
-

research!rsc: Updating the Go Memory Model (Memory Models, Part 3)

Go1.18 メモリモデル

Happens before

- 1 goroutineでは、reads/writesはプログラムに書かれた順序で実行された「かのように」ふるまう
    - 「かのように」ふるまうをキープするreorderはおこなわれる
- あるgoroutineからみた順序とべつなgoroutineからみた順序は異なる場合がある
- requirementsを記述するためにhappens beforeという半順序関係をつかう
- 1 goroutineの中では、プログラムに書かれた順序でhappens-before関係が決まる
- rがwを観測しうる(allowed to observe)
    - r < wではない
    - w < w' かつ w' < r というw'が存在しない
- rがwを観測すると保証される(guaranteed to observe)
    - w < r
    - vへの他の書き込みw'は、w' < w か r < w' を満たす
- 1 goroutineしか存在しないときは < が全順序になるのでallowdとguaranteedは同値になる
- 複数goroutineの間でhappens-before関係を成立させたいなら同期イベントを使う
- var v intのようなゼロ値への初期化はwriteとして扱われる
- single machine wordよりも大きな値のread/writeは、特定されない順序による複数のmachine-word-sized演算として振る舞う

Synchronization

定義されているもののリストだけ

- Goroutine creation
- Goroutine destruction
- Channel Communication
- Locks(sync.Mutex, sync.RWMutex)
- Once(sync.Once)

Incorrect synchronization

Go1.19 メモリモデル

- Introduction
- Advice
- Informal Overview
- Memory Model
- Implementation Restrictions for Programs Containing Data Races
- Synchronization
- Incorrect synchronization
- Incorrect compilation
- Conclusion

Introduction

Goメモリモデルは、あるgoroutineにおけるある変数のreadが、別なgoroutineにおける同じ変数へのwriteされた値を観測することが保証されるための条件を特定する。

Advice

Don’t be clever.

Informal Overview

- data raceはほかのread/writeとconcurrentに発生するwriteとして定義される
- プログラマはdata raceを避けるべき。これには同期演算を使う。
- data raceが存在しなければ、Goプログラムはあたかもすべてのgoroutineが1つのプロセッサの上に多重化(multiplexed)されたかのようにふるまう
    - この性質はDRF-SCと呼ばれる
- data raceが存在するときのGo実装の振る舞いも制限されている（なんでもありではない）
- 処理系はdata raceを報告してプログラムを終了することが許される
- そうでなければ、それぞれのreadはそのmemory locationに実際に書き込まれた値を観測しなければならない
    - readはsingle-word-sizedかそれより小さいと仮定する
- このアプローチはJavaやJavaScriptに近い

Memory Model

- Foundations of the C++ Concurrency Memory Model と近い
- メモリモデルはprogram executionに対する要請を記述する。
    - program execution
        - goroutine executions
            - memory operations

memory operation

memory operationは
- kind
- プログラムlocation
- アクセスする変数
- 読み書きされる値

で区別される。

read-like and write-like operations

memory operationsはread-likeまたはwrite-likeである。両方に当てはまる演算もある。

read-like演算	write-like演算
通常のread	通常のwrite
atomic read	atomic write
mutex lock	mutex unlock
channel receive	channel send, channel close
atomic compare-and swap	atomic compare-and-swap
など	など

Goroutine execution

- 単一のgoroutineで実行されるメモリ演算の集合としてモデル化される

Go program execution

Goroutine executionの集合とmapping Wの組。
Wはread-like operation rにたいして、rが読むwrite-like opを対応付けるマッピング
Q.

要請1

それぞれのgoroutineにおけるメモリ演算は、メモリに読み書きされる値が与えられたときのそのgoroutineのただしい逐次的実行に対応する。
この実行はsequenced before関係と一貫しなければいけない。このsequenced before関係はGo言語仕様の制御フローや式の評価順序によって定まる半順序関係の要請によって定義される。

Go「プログラム実行(execution)」は、goroutine実行の集合と、マッピングWの組としてモデリングされる。マッピングWは、それぞれのread-like演算が読み取るwrite-like演算を特定するマッピングである。（同一のプログラムにたいする複数の実行は、異なる「プログラム実行」を構成する）

Σ

要請3

メモリ位置xにおける通常のread rに対して、W(r)はrにとって可視(visible)なwrite wであり、可視であるとは次の2つが成り立つことを意味する。

- w < r
- w はxに対するほかのどのwrite w'にたいしてもw<w'<rではない

< はhappens beforeとする

read-write data race

同一のメモリ位置xにたいするread r, write wに対して

- rとwの少なくともいずれかがnon-synchronizingである
- rとwがhappens beforeで順序付けられない

write-write data race

同一のメモリ位置xにたいするwrite w, w'に対して

- 少なくともいずれかがnon-synchronizingである
- wとw'がhappens beforeで順序付けられない

残りの段落

- メモリ位置xにおけるdata-raceが存在しなければ、xにおけるすべてのread rに対して唯一のW(r)が定まる
    - ?
- より一般的に、DRFなGoプログラムはある逐次一貫的goroutineインタリーブにより説明される結果をもたらす
    - その証明はBoehm Adveの論文と同様
- ある種の演算は同期演算として働く。これらはsynchronized-before半順序を作り出す効果を持つ。詳細はSynchronizationセクションに書いてある。

Implementation Restrictions for Programs Containing Data Races

Sync

Sync.Lock

sync.RWMutex変数

∀ l.Rlock ∃n such that (nth l.Unlock <sync l.RLock AND l.RUnlock <sync n+1 l.Lock)

Why execution concept?

Go1.18 - Go1.19 メモリモデルドキュメント差分

~Go 1.18 (May 31, 2014)	Go1.19 (June 6, 2022)	変わったところ
Introduction - Advice	Introduction - Advice	変わってない
	Introduction - Informal Overview	新項目 DRF-SCに言及
Happens Before	Memory Model	より発達したモデル（理論）に書き換えられた
	Implementation Restrictions for Programs Containing Data Races	データレースのあるプログラムに対する振る舞い
Synchronization	Synchronization	定義されるルールが増えた
Incorrect synchronization	Incorrect synchronization	変わってない
	Incorrect compilation	新項目 コンパイラが行ってはいけない最適化を記述
	Conclusion	新項目 DRF-SCアプローチに言及

Synchronizationの記述差分(太字がGo 2022で新しく追加された項目)

Go 2022	Go 2022での関係の種類・内容
Initialization	● init関数同士の関係: importされるパッケージのinitはimportするパッケージのinitよりもhappens before ● initすべての完了はmain.mainよりもsynchronized before
Goroutine creation	● go文はgoroutine実行スタートよりもsynchronized before
Goroutine destruction	● goroutineから抜ける事象にはどのような事象との間のsynchronized before関係も保証されない

Channel communication	● channelへの送信は対応する受信よりもsynchronized before ● channelのクローズはそのクローズによって生じるゼロ値の受信よりもsynchronized before ● バッファなしchannelからの受信は対応する送信の完了よりもsynchronized before ● capacity Cをもつchannelからのk番目の受信はk+C番目の送信の完了よりもsynchronized before
Locks	● n < mなる整数n, mについてn番目のUnlock呼び出しはm番目のLockが戻るよりもsynchronized before ● RWMutexに対する任意のRLock呼び出しについて、ある整数nが存在して、n番目のUnlock呼び出しがRLockが戻るよりもsynchronized beforeであり、対応するRUnlock呼び出しはn+1番目のLock呼び出しよりもsynchronized beforeである ● 成功した**TryLock(TryRLock)**よびだしは**Lock(RLock)**呼び出しと等価である ○ 成功しなかったものは同期効果をもたない
Once	● once.Do(f)で実行されるf()の完了はどのonce.Do(f)呼び出しのreturnよりもsynchronized before
Atomic Values	● **sync/atomic**の**API**は**goroutine**間の同期に使える**atomic**演算である ● **A**と**B**を**atomic**演算として**A**の効果が**B**から観測されるならば**B**は**A**よりも**synchronized before** ● すべての**atomic**演算はある逐次一貫的順序で実行されたかのようにふるまう ● ちなみにこれは**C++**の逐次一貫的**atomics**や**Java(2004)**の**volatile**と同じセマンティクス
Finalizers	● **runtime.SetFinalizer**はあるオブジェクトが到達不能になったときに呼ばれる**finalizer**を追加する関数である ● **SetFinalizer(x, f)**の呼び出しは対応する**f(x)**よりも**synchronized before**
Additional Mechanisms	● **sync**パッケージには他にも同期機能をもつ**API**があるが、それはそれぞれのドキュメンテーションで記述される ● 他のパッケージが同期機能を提供するならば、そのパッケージはその効果を同様にドキュメントするべきである

Foundations of the C++ Concurrency Memory Model

Abstract

1. Introduction

わかりやすいモデルと難しいモデルがある

2. C++ Model Without Low-Level Atomics

- sequentially consistent atomics
- sequenced-before relation
- thread execution = memory actionsのsetと、seq-before順序と対応する半順序
- プログラムの逐次一貫的execution
  - thread executionのsetと<_{T} というtotal order, すべてのmemory action

3. Making Trylock Efficient

4. The Cost of Sequentially Consistent Atomics

5. Semantics of Data Races

6. C++ Model Supporting Low-Level Atomics

7. Sequential Consistency for Data-Race-Free Programs

定理7.1
section 6で定義したモデルはそのconsistent executionがtype 2 raceを含まないようなプログラムにsequential consistency を提供する

8. Equivalence of Race Definitions

Model Adjustments for Low-Level Atomics

Conclusions

参考文献ツリー

- The Go Memory Model - The Go Programming Language Goメモリモデル
  - Foundations of the C++ Concurrency Memory Model C++メモリモデルの論文
- https://go-review.googlesource.com/c/go/+/381315 メモリアップデートのCL
- research!rsc: Memory Models proposalの前に出されたrscの論文
  - research!rsc: Hardware Memory Models (Memory Models, Part 1)
    - How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs 逐次一貫性の正確な定式化
    - A Tutorial Introduction to the ARM and POWER Relaxed Memory Models Contents
    - CiteSeerX — Weak Ordering - A New Definition
  - research!rsc: Programming Language Memory Models (Memory Models, Part 2)
    - Memory Models: A Case For Rethinking Parallel Languages and Hardware | August 2010 | Communications of the ACM
    - N2176: Memory Model Rationales
    - Foundations of the C++ Concurrency Memory Model

        - Common Compiler Optimisations are Invalid in the C11 Memory Model and what we can do about it
        - The Problem of Programming Language Concurrency Semantics
        - [1805.07886] Constructing a Weak Memory Model
        -
    - research!rsc: Updating the Go Memory Model (Memory Models, Part 3)
- Updating the Go memory model · Discussion #47141 · golang/go · GitHub
    - doc: update Go memory model · Issue #50859 · golang/go · GitHub
    - sync/atomic: add typed atomic values · Issue #50860 · golang/go · GitHub
- Looking inside a Race Detector race detectorの素晴らしい解説、メモリモデルの理解にも最適
    - Vector clock - Wikipedia race detectorに使われるvector clockの解説
- 日本語資料
    - Go 1.19のメモリ周りの更新(Future Tech Blog) 澁川さんの記事. メモリモデルとrsc論文のまとめ
    - リトマステスト早見表 筆者資料. ハードウェアメモリモデルのまとめ
-

メモ欄

- Hyrum's law
- Q. Java(2004)もsubsequenceとhappens-beforeという2種類の順序関係だけで記述しているのに、なぜGo1.19は3種類あるのか？
    - Java(2004)に1種類ではなく2種類あるのはsubsequenceのほうに全順序公理（勝手に命名）があるからだけど
- https://www.microsoft.com/en-us/research/uploads/prod/2016/12/How-to-Make-a-Multiprocessor-Computer-That-Correctly-Executes-Multiprocess-Programs.pdf
    - 逐次一貫性
    - Sequential consistnecyが結局ちゃんとわかった気にならないと思っていたけどこの論文はまずSequential processorを定義している（シングルスレッドで正しい）
    - we describe a method of interconnecting sequential processors with memory modules that insures the sequential consistency of the resulting multiprocessor.
        - https://lamport.azurewebsites.net/pubs/proving.pdf
    - Sequentiality
        - プログラムの実行結果が、プログラムに書かれた順序で演算が実行された場合の結果と同一であることが保証されること
            - ※シングルスレッドで実行された場合に対する概念のはず（そうでなければ、「プログラムに書かれた順序」とは何かがわからないから）
    - Sequentialなプロセッサを共通メモリモジュールに繋いだコンピュータを考える
    - 要請は2つある
        - R1: それぞれのプロセッサはプログラムに書かれた順序でメモリリクエストを発行する。

- R2: メモリモジュールに対して発行されたメモリリクエストは単一のFIFOキューによって実行される。メモリリクエストの発行とはこのキューにリクエストを投入することである。

メモリ演算の種類	通常の演算	同期演算
具体例	通常のreadと通常のwrite	channelの操作 sync/atomicの関数 sync.Mutexの演算
コード例	x = 1 y = 0	close(done) mu.Lock()