Verteilte Softwaresysteme

Blatt 0

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 25.03.2020 08:44

Lesen Sie sich unbedingt vor Bearbeitung der Aufgaben auf diesem bzw. den folgenden Blättern die Informationen zu den Praktikumsaufgaben durch und beachten Sie diese.

Gemeinsames Repository

Es gibt ein gemeinsames Repository für die Veranstaltung. Damit ich weiß wer von Ihnen am Praktikum teilnimmt um einen Schein zu bekommen, gibt es zwei Möglichkeiten darauf zuzugreifen:

1. Wenn Sie **keinen** Schein benötigen, greifen Sie einfach direkt auf das Repository zu.
2. Wenn Sie **einen Platz im ZPA zugewiesen bekommen haben** und einen Schein bekommen wollen, tun Sie folgendes:
   a) Treten Sie über den GitHub-Classroom-Link `https://classroom.github.com/g/2yRsKclq` dem **einzigen** Team `20ss` bei.
   b) Schreiben Sie sich in dem Moodle-Kurs Lehrveranstaltungen (Oliver Braun, FK07) ein. Der Zugangsschlüssel ist *g3h31m*.
   c) Füllen Sie im Moodle-Kurs das Feedback *Zuordnung GitHub-Account* aus, damit ich weiß wer hinter welchem GitHub-Account steckt.

Tools

Auf der DevBox-VM ist bereits Go sowie Visual Studio Code mit der Go-Extension installiert. Es lohnt sich aber auf jeden Fall immer auf die aktuellen Versionen upzugraden.

Von JetBrains gibt es GoLand, wenn Sie eine IDE à la IntelliJ IDEA o.ä. bevorzugen. Das müssten Sie sich, auch auf der DevBox, noch selbst installieren.

Einarbeitung in Go

Über die Go-Website finden Sie eine Menge hilfreicher Ressourcen um sich in Go einzuarbeiten. Einen ersten *Berieselungseindruck* bekommen Sie beispielsweise durch das Video von Russ Cox auf YouTube. Eine sehr gute Möglichkeit für erste eigene Schritte in Go bietet die Tour of Go, die Sie im Browser, inkl. dem Ausführen von Go-Code, durchlaufen können.

Aufgaben

Wenn Sie sich gleich mit einer etwas konkreteren Aufgabenstellung einarbeiten wollen, können Sie die beiden folgenden Aufgaben bearbeiten. Sie bekommen dazu über den GitHub-Classroom-Link `https://classroom.github.com/a/Dwm8MRPC` ein Repository mit dem Sie arbeiten können. Die Aufgaben müssen Sie nicht abgeben.

Die beiden im Folgenden zu implementierenden Algorithmen sind in `https://link.springer.com/book/10.1007/978-3-8274-2804-2` beschrieben. Sie können das Buch als PDF im VPN der Hochschule herunterladen.

Aufgabe 1 — paralleles Quicksort

Implementieren Sie das rekursive Quicksort-Verfahren, so wie im o.a. Buch auf Seite 93ff beschrieben. In diesem Fall soll das Slice *in-situ* sortiert, also tatsächlich verändert werden.

Die Quicksort-Funktion hat die Signatur

```
func Quicksort(s []int)
```

Die eigentliche Sortierung soll dann in der Funktion

```
func quicksortParallel(a []int, wg *sync.WaitGroup)
```

durchgeführt werden. Nachdem in dieser Aufgabe keine Kommunikation durch Channels erfolgt, sondern die einzelnen Instanzen von `quicksortParallel` einfach ihren Teil des Slices bearbeiten, nutzen Sie die `WaitGroup`, damit die jeweilige Funktion auf die Beendigung der von ihr gestarteten Go-Routinen warten kann.

Da in Go ein Slice nur ein Ausschnitt eines darunter liegenden Arrays ist, können Sie einfach das zu bearbeitende Slice übergeben und müssen nicht, wie im Buch beschrieben, den Bereich durch explizite Angabe der rechten und linken Grenze einschränken.

Im Repository sind bereits einige Testfälle enthalten, die Sie natürlich gerne erweitern können.

Aufgabe 2 — paralleles Mergesort

Implementieren Sie den rekursive 2-Wege-Mergesort wie im o.a. Buch auf Seite 113ff beschrieben.

Die zu implementierende Funktion im Package `sorting` hat folgende Signatur:

```
func Mergesort(s []int) []int
```

Das übergebene Slice soll unverändert bleiben und ein sortiertes Slice als Ergebnis zurück gegeben werden.

Die eigentliche Sortierung soll in der Funktion (eigentlich eine Prozedur ;-))

```
func mergesortParallel(s []int, c chan<- int)
```

durchgeführt werden. Der Parameter `s` ist das zu sortierende Teilslice, der Channel `c` soll für die sortierte Rückgabe genutzt werden. Jeder Rekursionsschritt kommuniziert mit seinem Aufrufer über einen eigenen Channel. Die `Mergesort`-Funktion befüllt dann ein neues Slice mit den Werten die sie über den Channel bekommt. Dazu muss natürlich jedes `mergesortParallel` als eigene Go-Routine ausgeführt werden. Implementieren Sie Ihre `mergesortParallel`-Funktion so, dass sie keinen einzigen Wert speichern muss. Verwenden Sie ausschließlich *unbuffered* Channels.

Im Repository sind bereits einige Testfälle enthalten, die Sie natürlich gerne für Ihre Zwecke erweitern können.