# Performance Analysis für Aufgabe 5b Parallele Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

**Abgabe von:**
Bora Büyükbas,
Philip Seitz,
Fares Elkholy

**Übungsgruppe Nummer:** 3
**Semester:** SoSe 2024

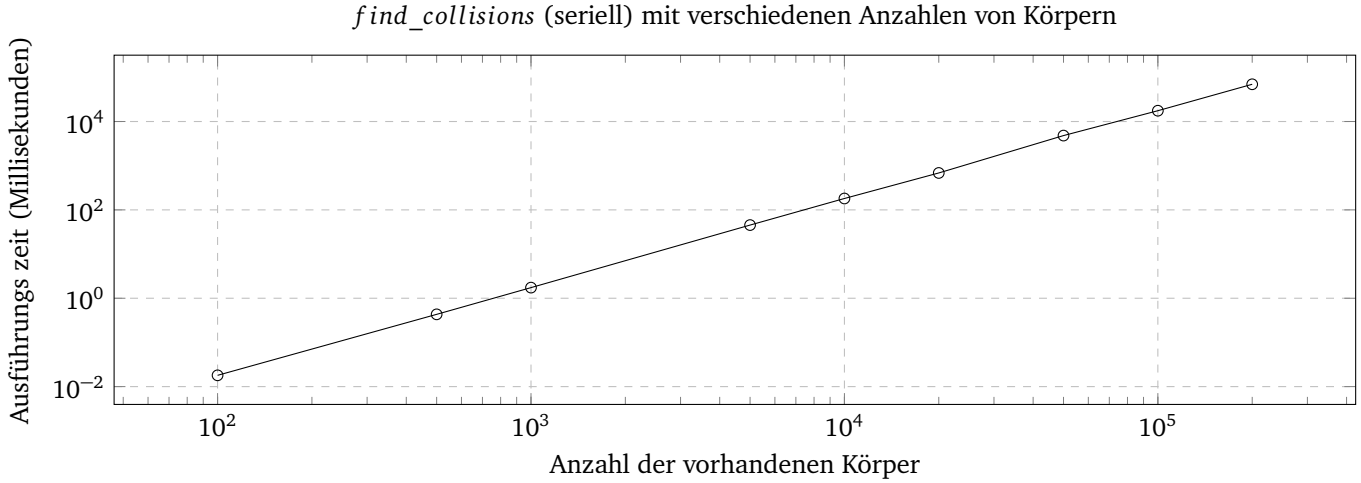
**Hinweis:** Alle Tests wurden im Lichtenberg-Hochleistungsrechner fünfmal für jeden Wert durchgeführt, und die Durchschnittswerte dieser Tests sind in den Grafiken dargestellt.

## 1 Rechenintensive Teile der Methode

Um einen vernünfitgen Speed up zu erreichen, haben wir zwei parallele Regionen in unserer Methode eingefügt.

Einmal haben wir die verschachtelte for-Schleife (beginnend auf Zeile 57 in `find_collisions()`) parallelisiert. Diese wird dafür verwendet, potenziel kollielierende Paare von Körper zu finden. Für jeden arbeitenden Thread ist dann eine `private_pair_set` vorgesehen, in der die gefundenen Paare gespeichert werden. Am Ende der parallelisierten for-Schleife werden die Ergebnisse der jeweiligen Threads in die `pair_set` zusammengeführt. Wir haben diese for-Schleife parallelisert, da sie bei $10^5$ Körper insgesamt $(10^5)^2 = 10^{10}$ Iterationen durchführt und somit sehr rechenintensiv ist. Wir unten zu sehen ist, ist dieser Teil der Methode tatsächlich das am rechenintensivsten. Zuletzt haben wir die für das Löschen der kollidierte Körper zuständige for-Schleife (beginnend auf Zeile 101) parallelisert. Da die Anzahl der Körper, die gelöscht werden müssen, in der Regel deutlich geringer ist als die Anzahl der Körper, die auf Kollisionen überprüft werden müssen, ist dieser Teil der Methode viel weniger rechenintensiv. Dennoch haben wir ihn parallelisiert, um den Speed up zu maximieren.

Um die Rechenintensivität dieser zwei Teile der Methode zu belegen, haben wir die Laufzeit der Methode für verschiedene Anzahlen von Körpern gemessen, wobei alle anderen Teile der Methode nicht geändert wurden. Die folgende Grafik zeigt die Laufzeit der Methode für $10^2$ bis $10^7$ Körper. Die Laufzeit der Methode steigt exponentiell mit der Anzahl der Körper.

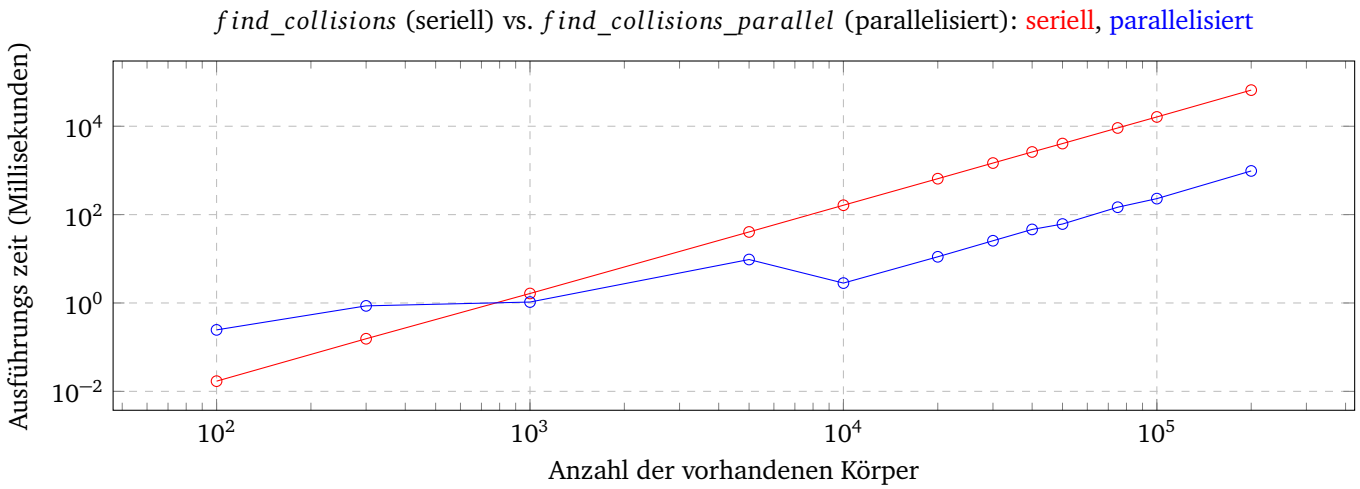


Die Messungen zeigen, dass die Ausführungszeit besonders ab etwa 5.000 Körpern deutlich zunimmt Weil diese zwei Teile der Methode stark von der Anzahl der Körper abhängig sind, zeigen diese Messungen auch, dass mit einer ansteigenden Anzahl von Körper diese Teile der Methode sehr rechenintensiv werden und daher eine Parallelisierung sinnvoll ist.

## 2 Effektivität der Parallelisierung durch OpenMP

Um die Effektivität der Parallelisierung zu zeigen, haben wir die Laufzeit der Methode für verschiedene Anzahlen von Körpern gemessen, wobei die Methode einmal parallelisiert wurde und einmal nicht.

Die folgende Grafik zeigt den jeweiligen Unterschied der Ausführungszeiten der Methode für verschiedene Anzahlen an Körper zwischen $10^2$ bis $10^5$.




Das Diagramm zeigt, dass zwar die paralleliserte Methode für kleine Anzahlen von Körpern ( $10^2$ Körper) langsamer ist als die serielle Methode, jedoch für größere Anzahlen von Körpern deutlich schneller ist. Dies ist darauf zurückzuführen, dass die serielle Methode für große Anzahlen von Körpern sehr rechenintensiv ist und die parallelisierte Methode die Rechenlast auf mehrere Threads verteilt.

Zum Beispiel wurde es für eine Körperanzahl von 100000 einen 50x-Speedup der Ausführungszeit der paralleliserten Methode im Vergleich zur serillen erreicht. Damit lässt sich abschließend behaupten, dass die Parallelisierung der Methode (für die meisten Fälle) sehr effektiv ist