Ausarbeitung Betriebssysteme Sommersemester 2022

Parallelisierung des KNN-Verfahrens

Name: Wessendorf

Vorname: Jona

Studiengang: KoI

Matrikelnummer: 3217753

Hiermit erkläre ich, dass ich das Programm und die vorliegende Ausarbeitung selbstständig verfasst habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen in Programm und in der Ausarbeitung, die anderen Quellen entnommen wurden, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Erklärung gilt auch ohne meine Unterschrift, sobald ich das Programm und die Ausarbeitung über die E-Prüfung der Vorlesung Betriebssysteme im Lernraum des eKVV an der Universität Bielefeld und unter Angabe meiner Matrikelnummer in der Ausarbeitung eingereicht habe.

1 Thread-Pool

Da die vier zu implementierenden Funktionen des Thread-Pools alle die `thread_pool_t` Struktur nutzen, werde ich zuerst erläutern wie diese aufgebaut ist. In dieser wird in einem Array eine Referenz zu allen threads gespeichert. Dazu wird dann auch die Größe des Thread-Pools gespeichert. Außerdem werden zwei Queues von Tasks gespeichert. Die eine enthält alle unbearbeiteten Tasks und die andere erhält alle, die bereits ausgeführt wurden.

In der `thread_pool_init` Methode werden alle benötigten Threads einmalig erstellt. Um geschützt auf Variablen zuzugreifen, werden Locks und Conditional Variabeln initialisiert. Außerdem werden noch die beiden Queues initialisiert. Beim Starten der Threads wird die `worker_function` zum Ausführen übergeben. Diese wartet immer so lange bis mindestens ein Task in der Queue ist, führt diesen aus und speichert die Ergebnisse in der Queue der erledigten Tasks.

Die `thread_pool_enqueue` Methode erstellt einen neuen Task und speicher diesen in der Queue der zu erledigenden Tasks. Anschließend signalisiert diese den Workerfunctions, dass ein neuer Task zur Verfügung steht.

Die `thread_pool_wait` Methode gibt das Ergebniss eines berechneten Tasks zurück. Dazu wartet sie so lange, bis die Queue der erledigten Tasks nicht leer ist. Sobald mindestens ein Element vorhanden ist, wird die Referenz zu diesem zurückgegeben und es wird aus der Queue entfernt.

Die `thread_pool_shutdown` Methode beendet alle gestarteten Threads und zerstört die Mutex Lock und Conditional Variabeln.

2 Ablauf

In der ersten Phase werden für alle Daten (Vektorn) die k-nächsten Nachbarn gefunden und jeweils in einem sortiertem Array gespeichert. Die zweite Phase berechnet die Klassifikation für jeden Verktor in Abhängigkeit von k. Dazu wird jeweils das Array mit den k-nächsten Nachbarn genutzt und es wird für jedes k die entsprechende Klassifikation berechnet. So wird auch in dieser Phase ein Array von ints mit der Länge k zurückgegeben. Die letzte Phase überprüft für jeden Vektor jede Klassifikation in Abhängigkeit von k. Stimmen diese mit der eigenen Klassifikation überein, so wird 1 gespeichert, sonst 0.

Nach Beendigung der drei Phasen wird für jedes k gezählt wie viele Vektoren richtig geschätzt wurden und diese Anzahl wird durch die Gesamtanzahl der Vektoren geteilt.

3 Aufteilung innerhalb der Berechnungsphasen

Für die Zuweisung der Aufgaben wird die Datenstruktur `func_args_t` genutzt. Diese speichert ein Array mit Pointern auf Datensätze (Vektoren). Dazu speichert sie den Index des für den Rechenschritt aktuellen Vektors. Außerdem wird ein Array von ints mit der Größe `2` (in Phase 1) oder `k_max` (in Phase 2-3) gespeichert. In der ersten Phase wird das Array genutzt, um die Grenzen des Intervalls des Testblocks zu übergeben. Die zweite Phase nutzt das Array für die Indices der k-nächsten Nachbarn. Die dritte Phase nutzt das Array für die Klasse der k-nächsten Nachbarn. Zuletzt wird die aktuelle Berechnungsphase gespeichert.

4 Koordination der Berechnungsphasen

Zuerst wird für jeden Vektor der erste Berechnungsschritt im Thread-Pool enqueued. Danach wird eine Schleife mit der Länge 3 * n durchlaufen, in der jeweils auf einen fertig bearbeiteten Task gewarted wird. Die Phase des fertig berechneten Tasks wird um 1 inkrementiert und es wird ein neuer Task mit der funktion für die nächste Phase enqueued. Dabei wird der Rückgabewert der vorherigen Phase zum Inputarray für die nächste Phase. Die Parameter für den Index des aktuellen Elements und für die Datenvektoren bleiben die gleichen.

5 Schwachstellen und Verbesserungsmöglichkeiten

Für sehr große n ist das Programm extrem langsam, da für jeden Vektor die Distanz zu allen anderen Vektoren berechnet. Da ich die Ergebnisse nicht speicher wird jede Distanz zweifach berechnet. Hier könnte es also möglich durch mehr Speichernutzung die Laufzeit des Algorithmus zu verringern.

6 Analyse der Laufzeiten

Bei der Analyse der Laufzeit habe ich als n `50.000` gewählt. Das Testsystem verfügt über einen Prozessor mit 6 Kernen, die sich in 12 Threads aufteilen lassen. In der folgenden Tabelle sind die Laufzeiten relativ zur Laufzeit bei der sequentiellen Abarbeitung dargestellt.

n_threads	time relative to n_threads = 0
0	1
1	1,01
2	0,51
4	0,26
6	0,18
8	0,17
10	0,16
12	0,15
14	0,15
20	0,15
100	0,16
1000	0,17

Es ist zu erkennen, dass bei der Verwendung von nur einem Thread das Programm etwas langsamer läuft, als bei der sequentiellen Abarbeitung. Dies liegt vermutlich daran, dass der Thread von System erstellt werden muss, was mehr Zeit braucht. Ab einer Anzahl von zwei Threads ist eine Verringerung der Laufzeit bis zu einer Anzahl von 12 Threads zu sehen. Bei mehr als 12 Threads verändert sich die Laufzeit erstmal nicht so viel. Erst wenn die Anzahl der Threads deutlich größer wird, erhöht sich die benötigte Zeit wieder. Dies liegt vermutlich daran, dass es mehr Zeit braucht diese Threads zu erstellen.

7 Begrenzende Faktoren

Wie es bei der Analyse der Laufzeit schon angedeutet wurde, ist der Wert von `n_threads` auf dem Testsystem bei 12 optimal. Dies ist der Fall, da der Prozessor nur über 12 Threads verfügt. Es können also nicht mehr als 12 Threads gleichzeitig auf dem System ausgeführt werden. Deshalb wird es ab 12 nicht mehr schneller.

So ist also der begrenzende Faktor also die Anzahl der Threads über die, die CPU des Testcomputers verfügt.

Quellen

[1] Wikipedia entry “K-nearest neighbors algorithm”, konsultiert am 03. August 2022, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm