Verteilte Systeme

Übung A2

Sommersemester 2018

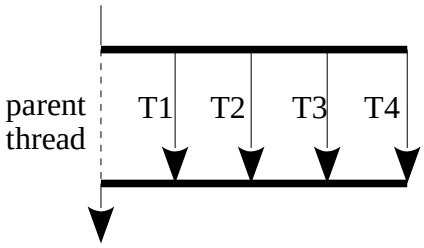
Sascha Baumeister

Diese Übung adressiert Grundlagen zum Multi-Threading und zur Vektor-Prozessierung.

1 Software-Simulation von Vektor-Prozessierung

Kopiert die Klasse `VectorMathSingleThreaded` nach `VectorMathMultiThreaded`. Die beiden Methoden `add()` und `mux()` modellieren Vektor-Addition bzw. Vektor-Multiplexen. Sie sollen so umgeformt werden, dass die Berechnung der Ergebniswerte in genau so vielen Threads erfolgt wie der ausführende Computer Prozessoren besitzt (siehe `Runtime.getRuntime().availableProcessors()`).

Vor der Rückgabe des Ergebnisses soll mittels Futures gewartet werden bis alle im Rahmen der Methodenausführung gestarteten Threads wieder beendet sind. Siehe die Demo-Klasse `ResyncThreadByFutureInterruptibly` für ein Beispiel zur Synchronisierung mittels Futures:



Beachtet dabei:

- Es wäre äußerst ungeschickt das Ergebnis stückchenweise in den Child-Threads anzulegen, und im Parent-Thread zu einem Ergebnis zusammenzusetzen. Besser ist es das Ergebnis Parent-Thread als Ganzes zu erzeugen, und in den Child-Threads nur die Elemente zu setzen!
- Beim Synchronisieren müssen im Fehlerfall die Child-Threads beendet werden bevor die Methode im Parent-Thread verlassen wird. Zudem muss dabei die Child-Thread Exception im Parent-Thread erneut geworfen werden (*manueller Precise-Retrow* – siehe o.g. Beispiel)!
- Wenn man versucht jeden Thread in etwa mit der gleichen Anzahl an aufeinander folgenden Elementen zu beaufschlagen, dann hat jeder Thread (je nach Vektor-Dimension) N oder N+1 Ergebnis-Elemente zu berechnen. N ist dabei immer das Ergebnis der Ganzzahldivision (/) aus Anzahl Elemente und Anzahl Threads. Die Anzahl der Threads die N+1 Elemente bearbeiten müssen beträgt dabei den Modulo (%) aus Anzahl Elemente und Anzahl Threads.
- Alternativ kann man auch einfach den Ergebnisvektor streifenweise berechnen, in diesem Fall reicht eine einfache Addition und Multiplikation zur Indexberechnung aus.
- Beachtet in allen Fällen dass Laptops (und die meisten Desktops) nicht wirklich für Vollast-Berechnungen mittels mehrerer CPU-Kerne über signifikante Speicherbereiche hinweg ausgelegt sind – das RAM ist dazu häufig viel zu langsam! Daher ist ein Skalierungserfolg bei der `add()`-Methode unter keinen Umständen zu erwarten, und bei der `mux()`-Methode nur bei großen Matrizen und geeigneter Hardware.
- Beachtet zudem dass Eure Messungen von zwei technischen Eigenschaften moderner CPUs maßgeblich beeinflusst sein können: CPU-Hersteller statten ihre Prozessoren zum Teil mit *Hyperthreading* Technologie aus; diese bewirkt eine Verdopplung der Anzahl der Prozessoren, die jedoch dann jeweils signifikant langsamer sind als Prozessoren mit jeweils eigener Ausführungseinheit (Core). Zudem Übertakten moderne High-End CPUs einen Kern deutlich solange die anderen nicht allzu hoch belastet sind; dies erhöht die Leistung von Software im Single-Thread Design gegenüber Multi-Threading!