Verteilte Systeme

Übung A2 - Einzelarbeit

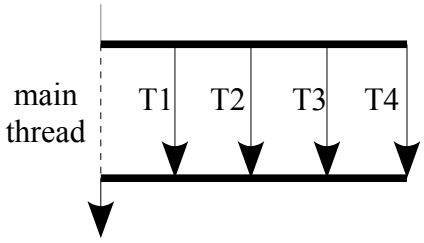
Bearbeitungszeit 2 Wochen

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

Aufgabe: Software-Simulation von Vektor-Prozessierung

Kopiert die Klasse *VectorMathLinear* nach *VectorMathDistributed*. 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 Prozessorkerne besitzt.

Vor der Rückgabe des Ergebnisses soll mittels verschuldeter Semaphore gewartet werden bis alle im Rahmen der Methodenausführung gestarteten Threads wieder beendet sind!



Beachtet dabei:

- Die Anzahl der CPU-Kerne ist in der statischen Variablen PROCESSOR_COUNT ersichtlich.
- Es wäre äußerst ungeschickt das Ergebnis stückchenweise in den Threads anzulegen, und am Ende zusammenzusetzen. Besser ist es das Ergebnis als Ganzes zu erzeugen, und in den Threads nur die Elemente zu setzen!
- 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. Hier kann es aber auf manchen Arten von CPUs zu zusätzlichen signifikanten Kollisionskosten durch Cache-Invalidierung kommen falls nicht nur die gelesenen, sondern auch dazu benachbarte Speicherbereiche von der CPU gecached werden! Daher ist dieser Lösungsansatz bei Multi-Plattform Software zu vermeiden.
- 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: Intel stattet seine Prozessoren meistens mit patent-geschützter Hyperthreading-Technologie aus; diese bewirkt eine virtuelle Verdopplung der Anzahl der CPU-Kerne, die jedoch dann jeweils signifikant langsamer sind als physische Kerne. Zudem Übertakten moderne High-End CPUs zurzeit einen Kern deutlich solange die anderen nicht allzu hoch belastet sind; dies erhöht die Leistung von Software im Single-Thread Design gegenüber Multi-Threading!