Studenten: Christoph Rosenhauer (2549220)
Nebenläufige Programmierung SoSe 2015 Lukas Schaal (2539218)
Universität des Saarlandes Jens Heinen (2542182)
Milestone 1

Osmose-Prozess: Meilenstein 1

July 21, 2015

1 Allgemeines & Überblick

Die Graphstruktur $n \times m$ soll durch eine variable Anzahl von Threads bearbeitet werden, was von 1 Thread (optional sequentielles Bearbeiten) bis zu m Threads reichen kann, so dass jede Spalte des Graphen in jeder Iteration von genau einem Thread bearbeitet wird. Zusätzlich werden sog. Synchronisationsgrenzen angegeben, nach wie vielen Iterationsschritten die Akkumulatoren an den Spaltenübergängen synchronisiert werden sollen. Dieser Parameter soll von außen steuerbar sein. Zur verwendeten Datenstruktur siehe 3.

2 Nebenläufige Probleme

Der Zeitpunkt der Synchronisation soll anhand eines Parameters erfolgen. Falls diese Bedingung erfüllt ist, werden mittels Shared Memory die Daten der Akkumulatoren ausgetauscht. Im Detail wie folgt:

Wir starten für jede Spalte einen Iterationstask, der x Iterationen ausführt. Dieser wird an den Threadpool übergeben. Danach rufen wir *shutdown* auf dem Pool auf (er nimmt danach keine neuen Tasks an, führt die übergebenen jedoch weiter aus. Hierbei ist die Reihenfolge der Ausführung prinzipiell egal.) Danach wartet der ThreadPool mit *awaitTermination* als Bedingung einer while-Schleife, bis alle noch laufenden Threads ihren Task beendet haben. Dann wird ein Synchronisationstask für je 2 benachbarte Spalten an den Threadpool übergeben. An dieser Stelle ist wichtig, dass zum Beginn der Synchronisation zweier Spalten kein Thread mehr einen Iterationstask auf einer der Spalten ausführt. Ein Thread nimmt sich nun die Referenzen auf die Nachbarspalten und verrechnet Akkumulatordaten und aktuelle Werte der Knoten, in Abhängigkeit von vorhandener lokaler Konvergenz, siehe 4.1. Hierbei muss beachtet werden, ob die propagierenden Spalten von einem oder mehreren Threads bearbeitet werden.

3 Verwendete Datenstrukur

Der gegebene Graph ist ein $n \times m$ Graph. Dabei wird das Objekt intern auch in m Spalten unterteilt.

Die Zahl der erstellten Threads hängt vom verwendeten Threadpool ab. Ein Thread bekommt dabei einen Task übergeben und bearbeitet nach diesem Task eine oder mehrere Spalten des Graphen. Ein Task steht für x Iterationsschritte; für jede Spalte wird ein eigener Task gebaut. Die Threads werden mittels eines Threadpools den Spalten zugewiesen und bei Bedarf erstellt. Die Tasks beinhalten lokalen Austausch (vertikale Berechnung und Erstellen der Akkumulatoren) und Synchronisation zwischen mehreren Spalten.

Jede Spalte besteht aus ihren Knoten und Vektoren der links- und rechtsbenachbarten Akkumulatoren. Durch Call-by-Reference kann ein Thread, der gerade Spalte j bearbeitet, auch Daten von Spalte $j-1$ & $j+1$ bekommen. An dieser Stelle sollte durch genaues Locking die Gefahr von Data Races verhindert werden.

Ein Knoten ist ein Objekt, das seinen aktuellen Wert und den alten Wert zur Überprüfung der lokalen Konvergenz kennt. Die Übergangsrate ist in der GraphInfo Klasse enthalten. Kann als Vektor implementiert werden.

4 Konvergenz

4.1 Lokale Konvergenz

Die lokale Konvergenz hängt nicht nur vom vertikalen flow, sondern auch vom horizontalen inflow/outflow der beteiligten Spalten ab. Diese Konvergenz wird nicht explizit nach x Iterationsschritten geprüft, sondern implizit nach einer erfolgreichen Synchronisation. Wenn in einer Spalte nach x Iterationen keine Änderung der Knotenwerte erfolgte und $|inflow - outflow| \leq \epsilon$ ist, so ist diese Spalte lokal konvergent. Dies kann sich durch Änderung des inflows ändern.

Aus dieser Definition von lokaler Konvergenz ergeben sich drei Synchronisationsfälle:

- Zwei betrachtete Spalten sind **nicht** lokal konvergent. Dann werden die Werte der Akkumulatoren mit den der Knoten verrechnet. Danach überprüfe lokale Konvergenz in beiden Spalten.
- Eine Spalte ist lokal konvergent. Dann wird deren Outflow mit den Werten der Knoten der benachbarten Spalte verrechnet. Wenn $|inflow - outflow| > \epsilon$, wird der inflow mit den aktuellen Werten der Knoten verrechnet und die Spalte wird ab diesem Zeitpunkt nicht mehr als lokal konvergent betrachtet.
- Beide Spalten sind lokal konvergent. Es wird keine Synchronisation durchgeführt.

Inwiefern die Anwesenheit von lokaler Konvergenz die Synchronisationsintervalle beeinflusst, muss noch entschieden werden.

4.2 Globale Konvergenz

Globale Konvergenz erfolgt implizit durch lokale Konvergenz. Lokale Konvergenz in allen Spalten impliziert globale Konvergenz des Graphen.