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

# 1 Allgemeines & Überblick

Die Graphenstruktur wird insgesamt in m gröbere Spalten unterteilt, wobei m als steuerbarer Parameter implementiert werden kann. Zusätzlich werden sog. Synchronisationsgrenzen angegeben, nach wie vielen Iterationsschritten die Akkumulatoren an den Spaltenübergängen synchronisiert werden sollen. Dieser Parameter soll ebenfalls von außen steuerbar sein.

# 2 Nebenläufige Probleme

Hier gehen wir folgenden Weg:

- Die Synchronisation soll anhand eines Parameters erfolgen. Falls diese Bedingung erfüllt ist, werden mitels Shared Memory die Daten der Akkumulatoren ausgetauscht. Zwei oder mehrere
- Wie wird die Synchronisationsbedingung i = k (Iterationsstadien unterschiedlicher Spalten) sichergestellt? Synchronisiert wird auf folgende Weise: Nach x festgelegten Iterationstasks wird ein Synchronisationstask an alle Threads übergeben. Dabei nimmt sich ein Thread die Referenzen auf die Nachbarspalten und verrechnet Akkumulatordaten und aktuelle Werte der Knoten.

# 3 Verwendete Datenstrukur

Der gegebene Graph ist ein $n \times m$ Graph. Dabei wird das Objekt intern auch in m Spalten unterteilt. Die Anzahl der operierenden Threads wird über einen Parameter gegeben. Ein Thread bekommt dabei einen Task übergeben und bearbeitet nach diesem Task eine oder mehrere Spalten des Graphen. Die Threads werden mittels eines Threadpools / Scheduler den Spalten zugewiesen. Die Tasks beinhalten lokalen Austausch, Synchronisation zwischen mehreren Spalten und globale Konvergenz. 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 nur seinen aktuellen Wert kennt. Die Übergangsrate ist in der GraphInfo Klasse enthalten.

# 4 Konvergenz

## 4.1 Lokale Konvergenz

Die lokale Konvergenz innerhalb einer Spalte wird nach festgelegten Iterationsschritten (abhängig von Parameter) überprüft. Im Falle einer erkannten lokalen Konvergenz innerhalb einer Spalte wird mit den benachbarten Spalten synchronisiert. Dafür werden der Iterationszustand i von Spalte j und Zustand k von Spalte j+1 angepasst, so dass i = k gilt. Dazu wird die Spalte mit der lokalen Konvergenz so lange iteriert, bis sie auf dem gleichen Stand ist wie die benachbarte Spalte. Falls nicht, wird nach x Iterationen definitiv synchronisiert, wobei x von einem Parameter abgeleitet wird. Inwiefern x davon abhängt, wird durch Tests ermittelt und im Laufe der Implementierung optimiert.

## 4.2 Globale Konvergenz

Globale Konvergenz erfolgt implizit durch lokale Konvergenz. Dazu existieren 2 Synchronisationsarten: Eine festgelegt durch den Parameter, eine bei lokaler Konvergenz einer Spalte. Wenn lokale Konvergenz vorliegt, wird das Verhältnis zwischen Inflow und Outflow der betreffenden Spalten

Studenten: Christoph Rosenhauer (2549220)
Lukas Schaal (2539218)
Jens Heinen (2542182)


überprüft. Wenn der Unterschied zwischen Inflow & Outflow zu groß wird , muss in kleineren Iterationsdistanzen überprüft werden.