Nebenläufige Programmierung SoSe 2015
Universität des Saarlandes

Studenten: Jens Heinen (2542182)
Lukas Schaal (2539218)
Christoph Rosenhauer (2549220)
Milestone 1

Test test test

- Anzahl der Threads geregelt über von außern steuerbaren Parameter
- Graphen in unterschiedliche Klassen unterteilen
- Objekte werden in Oberklasse zusammengefasst
- MP oder SM ? noch zu klären
- Notiz: Max Wert von Initialisierung wird nie überschritten!
- Unterteilung der Graphstrukur in gröbere "Spalten", die aus mehreren Einzelspalten bestehen
- Jede Spalte wird von einem Thread bearbeitet

1 Allgemeines & Überblick

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

Klassenstruktur: Ein Graph wird dynamisch aufgebaut. Dafür gibt es eine Klasse *klasse*. Ein Knoten entsteht, sobald sein Wert positiv wird. Die Knoten werden in einer Oberklasse *Oberklasse* zusammengesetzt. Zur spaltenweise Abarbeitung wird es eine Klasse *Column?* geben. Jede Column bekommt nach Erstellung einen Thread zugewiesen, der die Berechnungen in jeder Iteration übernimmt.

2 Nebenläufige Probleme

Hier gehen wir folgenden Weg:

- Die Synchronisation soll erfolgen, sobald zwei Spalten in den Fall des Propagierens gelangen *Bedingung hierfür ist was ?*. Falls diese Bedingung erfüllt ist, werden mitels eines Message Passing / Shared Memory Konzepts die Daten der Akkumulatoren ausgetauscht. Zyklisches Verhalten bei seltenem Propagieren? Wie handhaben ?
- Wie wird die Synchronisationsbedingung i = k (Iterationsstadien unterschiedlicher Spalten) sichergestellt?
- Globale / Lokale Konvergenz: alle Threads erreichen ein Maximum an Iterationen ? Alle Spalten bekommen eine eigene Approximationsgrenze ?

3 Verwendete Datenstrukur

Als Datenstruktur implementieren wir eine *ArrayList?* deren Elemente einzelne Knoten sind. Jeder Knoten speichert linke und rechte Akkumulatorenfelder.