Praktische Informatik in der Wirtschaft
Westfälische Wilhelms-Universität Münster
Prof. Dr. Herbert Kuchen 12. Dezember 2016

9. Übung zu Informatik I im WS 16/17
Abgabe: 9. Januar 2017, 12 $\frac{00}{}$ Uhr.

Aufgabe 22 - Gruppenabgabe (10 + 10 + 10 =30 Punkte)
An einer Metallplatte liegen an allen vier Seiten unterschiedliche aber konstante Temperaturen an. Nun soll errechnet werden, wie sich jeder einzelne Punkt der Metallplatte erhitzt. Dazu wird die Metallplatte in einzelne Messpunkte unterteilt, z.B. 100x100. Anschließend wird in einem iterativen Verfahren die Temperatur jedes einzelnen Messpunktes immer wieder als Durchschnitt der Temperaturen seiner Nachbarpunkte errechnet. Die Formel, die zur Berechnung verwendet wird, ist die folgende:

$$a_{x,y}(t) = \frac{1}{4}(a_{x-1,y}(t-1) + a_{x,y-1}(t-1) + a_{x+1,y}(t-1) + a_{x,y+1}(t-1))$$

Dabei bezeichnet $a_{x,y}(t)$ die Temperatur an den Koordinaten (x,y) zum Zeitpunkt t. Falls x oder y außerhalb der Grenzen der Metallplatte liegen, werden die entsprechenden Temperaturen, welche an den Seiten der Metallplatte anliegen, verwendet. (**Hinweis:** Bei einer Größe von 100x100 ist der Punkt oben links (0,0) und der Punkt oben rechts (99,0)) Als Abbruchkriterium dienen die Temperaturveränderungen jedes einzelnen Punktes von einem Zeitpunkt t zu $t + 1$. Wenn von diesen keine größer ist als ein Schwellenwert ϵ, wird angenommen, dass das Ergebnis hinreichend exakt ist und die Berechnung wird abgebrochen (Dieses Verhalten ist ähnlich zu dem von Aufgabe 12).

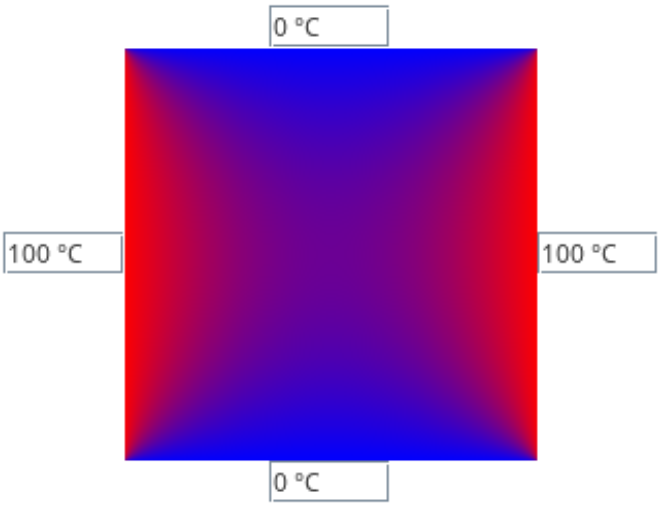

Abbildung: Darstellung des Ergebnisses der oben genannten Berechnung.

Im Learnweb finden Sie ein Projekt, welches die oben dargestellte GUI bereits zur Verfügung stellt. Außerdem ist die Datenstruktur (die Klasse `HitzePanel.DatenMatrix`), welche Sie zum Lösen des Problems verwenden sollen, bereits implementiert. Diese stellt vier Methoden zur Verfügung:

getValue(x,y) Gibt den Wert der letzten Iteration an der Stelle (x, y) zurück.
(Entspricht $a_{x,y}(t-1)$)

setValue(value,x,y) Setzt einen Wert der aktuellen Iteration an der Stelle (x, y).
(Entspricht $a_{x,y}(t) = value$)

nextIteration() Schließt die aktuelle Iteration ab, indem `getValue` und `setValue` nun auf die jeweils nächste Iteration zeigen.
(Entspricht $t = t + 1$)

getSize() Gibt die Größe der Metallplatte zurück, bzw. die Anzahl der Messpunkte pro Seite.

a) Implementieren Sie die Methode `solve()` innerhalb der Klasse `SequentialSolver`. Verwenden Sie hierfür das Prinzip der schrittweisen Verfeinerung (siehe b). Nachdem das Ergebnis fertig berechnet wurde, muss die Methode `this.finish()` aufgerufen werden. Diese sorgt dafür, dass das Bild aktualisiert wird und die Anwendung wieder reagiert.

b) Bereiten Sie die Zwischenstände der schrittweisen Verfeinerung auf und laden Sie diese als Pdf-Datei hoch. Dabei sollte klar werden, wie Sie bei der schrittweisen Verfeinerung vorgegangen sind.

c) Implementieren Sie die Methode `solve()` innerhalb der Klasse `ParallelSolver`. Greifen Sie hier auf das in der Vorlesung vorgestellte Prinzip der Threads zurück. Insgesamt sollten 4 Threads erstellt werden, welche jeweils einen möglichst gleich großen Teil der Werte der DatenMatrix berechnen. Um mit der Berechnung der nächsten Iteration zu beginnen, müssen die schnelleren Threads warten, bis alle mit der Berechnung fertig sind. Anschließend kann von einem der Threads die Methode `matrix.nextIteration()` aufgerufen werden, bevor alle Threads weiterarbeiten können. Sorgen Sie dabei für eine sorgfältige Anwendung der Methoden `wait()`, `notify()`, bzw. `notifyAll()` und der synchronisierten Methoden. Nachdem die Temperaturänderungen aller Threads kleiner als ϵ sind, muss auch hier die Methode `this.finish()` aufgerufen werden.

Hinweis: Über die Oberfläche können Sie die Größe der Platte und den Wert von Epsilon manipulieren. Die Checkbox gibt an, ob während der Ausführung der Methode solve() bereits eine Aktualisierung der Grafik stattfinden soll. Dadurch lässt sich schön nachvollziehen, wie sich die Temperaturverteilung verhält. Dies wird aber nur funktionieren, wenn Ihre `solve()` in einem neben-läufigen Prozess ausgeführt wird (bspw. einem Thread), da ansonsten die Anwendung bis zum AbschlieSSen der Berechnung nicht reagiert. Für Ihren parallelen Solver sollte dies also funktionieren. Für den sequentiellen eher nicht.

Anmerkung: *Versehen Sie ihre Abgaben bitte mit* ***Namen****,* ***Matrikelnummern****,* ***E-Mail-Adressen*** *und* ***Studiengängen*** *der beteiligten Bearbeiter/innen und laden Sie diese in der entsprechenden Aktivität im Learnweb hoch. Aufgaben, die mit dem Hinweis* ***Gruppenabgabe*** *versehen sind, dürfen mit maximal 3 Bearbeiter/innen gelöst werden. Aufgaben mit dem Hinweis* ***Einzelabgabe*** *müssen von jedem Studenten und jeder Studentin eigenständig gelöst und abgegeben werden. Viele der Aufgaben werden über das* ***EASy*** *(E-Assessment System) System im Learnweb eingereicht und automatisch vorausgewertet. Um Probleme bei der Abgabe wegen Überlastung des EASy-Servers zu vermeiden, würde ich Sie bitten, eine Abgabe auf den letzten Drücker nach Möglichkeit zu vermeiden. Bei Fragen zum Übungsbetrieb wenden Sie sich bitte an ihren/ihre Tutor/in oder an Tobias Reischmann.*