Bierfass-Simulation

Mark Geiger Rico Fritzsche

4. Mai 2015

Zusammenfassung

Im Kurs Parallele Programmierung soll eine Simulation implementiert werden, welche das Abkühlen eines Fasses darstellt. Das Fass soll in einem Eimer mit Null Grad Celsius kaltem Wasser so tief eingetaucht werden, das die Durchschnittstemperatur nach genügend langer Zeit Zehn Grad Celsius beträgt. Eine Grafik soll das Ergebnis darstellen.

Umsetzung

Für die Simulation wird eine Matrix definiert, welche in drei Bereiche geteilt wird. Einen Rand, welcher an jeder Seite aus einer Reihe besteht. Hier werden die Außentemperaturen festgelegt. Der äußere Bereich wird dazu nochmals horizontal geteilt, was die Eintauchtiefe des Fasses umsetzt. Dem oberen Teil des Randes wird die Außentemperatur zugewiesen, in unserem Fall 30 Grad Celsius. Der untere Teil erhält den Wert Null Grad Celsius, welches die Temperatur des Wasser in dem Eimer entspricht. Das innere wird mit einer beliebigen Ausgangstemperatur belegt.
Damit die Threads bei der Berechnung sich nicht gegenseitig Behindern und falsche Werte berechnen, wird eine Schreib- und eine Lesematrix initialisiert. Anschließend wird die gewünschte Anzahl an Threads gestartet, in unserem Beispiel 200. Diese führen die Berechnung nach folgendem Muster aus. Jeder Thread wählt einen zufälligen Punkt in der Matrix den er im Zyklus bearbeitet. Ist der gewählte Punkt eine Randpunkt, startet er einen neuen Zyklus. Trifft er einen Punkt innerhalb des Randes (im Fass), holt er sich die Werte der umliegenden Punkte aus der Lesematrix. Aus diesen bildet er den Durchschnitt und schreibt das Ergebnis an die Stelle des zu Beginn gewählten Punktes in die Schreibmatrix. Anschließend beginnt ein neuer Zyklus. Der beschriebene Algorithmus wird in Listing 1 abgebildet.

```
1  void step(){
2      int size = SIZE;
3
4      while(finished == FALSE){
5          int c,d;
6          c = rand() % (size-2)+1;
7          d = rand() % (size-2)+1;
8
9          if (isPointWarm(c,d)) {
10             writeMat[c + d * SIZE] =  WARM_TEMP;
11         } else if(isPointCold(c,d)) {
12             writeMat[c + d * SIZE] = COLD_TEMP;
13         } else {
14             double z1 = readMat[c + 1 + d * size];
15             double z2 = readMat[c - 1 + d * size];
16             double z3 = readMat[c + (d - 1) * size];
17             double z4 = readMat[c + (d + 1) * size];
18             double z5 = readMat[c + d * size];
19             double z6 = readMat[c + 1 + (d - 1) * size];
20             double z7 = readMat[c - 1 + (d + 1) * size];
21             double z8 = readMat[c - 1 + (d - 1) * size];
22             double z9 = readMat[c + 1 + (d + 1) * size];
23             writeMat[c + d * SIZE] = (z1 + z2 + z3 + z4 + z5 +
24              z6 + z7 + z8 + z9) / 9.0;
25         }
26     }
27 }
```

Im Mainthread werden nach kurzer Wartezeit die Ergebnisse der Schreibmatrix in die Lesematrix kopiert. Damit rechnen die Threads mit etwas Verzug mit den aktuellen Temperaturwerten.
Zur Visualisierung wird die Schreibmatrix beim kopieren in die Lesematrix zusätzlich in eine Textdatei kopiert. Nach Beendigung des Programms liest ein Python-Skript die Dateien ein und erstellt eine Colour-Map für jede gespeicherte Matrix. Abschließend können die erstellten Colour-Maps zu einer Gif-Datei zusammengefügt. In der Abbildung 1 ist das Endergbnis in einer Clour-Map aufgetragen.

Ausführung

make (baut erste Implementation -¿ die bessere)
make main2 (um 2. Implementation zu bauen)
make plot (nur möglich wenn in main2 matrizen geschrieben werden, siehe const WRITE_RESULTS)

Visualisierung

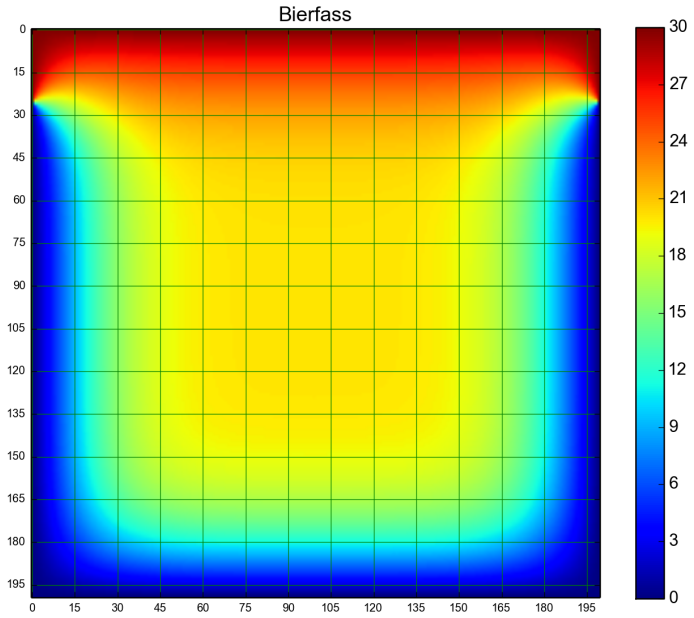

Abbildung 1: Ergebnis der Testimplementation, dargestellt mittels Python-Scripts