Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Computer Engineering, Fachbereich I

Fach: Parallele Architekturen und Prozesse

Dozent: Sebastian Bauer

Projektarbeit

„Wärmeleitungsgleichung“

vorgelegt von:

StudentIn:	Anastasija Loskutow
	Marcel Jödicke
Studiengang:	Computer Engineering
Fachsemester:	7
Matrikelnummer:	s0539898 s0540129
E-Mail:	s0539898@htw-berlin.de
	s0540129@htw-berlin.de

Berlin, 23.02.2016

Inhalt

Verzeichnis der Abbildungen

1 Ziel des Projektes

Ziel der Projektarbeit im Rahmen des Fachs „Parallele Architekturen und Prozesse“ war es, ein Programm zur Berechnung der einfachen Wärmeleitungsgleichung zu entwickeln und zu optimieren. Auch war es wichtig eine grafische Oberfläche mit Eingriffsmöglichkeiten zu implementieren.

2 Beschreibung der Wärmeleitungsgleichung

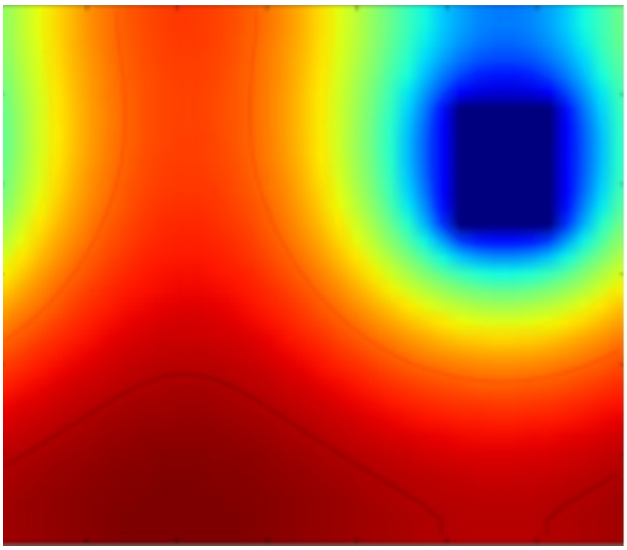

Abbildung 1: Wärmeleitung

Die Wärmeleitungsgleichung ist eine partielle Differentialgleichung, die den Zusammenhang zwischen der zeitlichen und der räumlichen Änderung der Temperatur an einem Ort in einem Körper beschreibt. Die zeitliche Temperaturänderung eines Körpers erfolgt ausgehend von einem Anfangszustand, also dem Zustand zum Zeitpunkt t=0, und den Grenzbedingungen. Sie eignet sich z.B. zur Berechnung instationärer (partieller) Temperaturfelder.

Die Wärmeleitungsgleichung lässt sich aus dem Energieerhaltungssatz und dem Fourierschen Gesetz der Wärmeleitung herleiten. Man kann sie auch als Diffusion von thermischer Energie bezeichnen.

2.1 Ausgewählte Methode für die Umsetzung

In unserem Beispiel geht es um ein Objekt, das die Form eines Quadrates mit Kantenlänge a hat. Dabei besteht der Quadrat aus vielen kleineren Subquadraten, die unterschiedliche Temperaturzustände repräsentieren (siehe Abbildung 2).

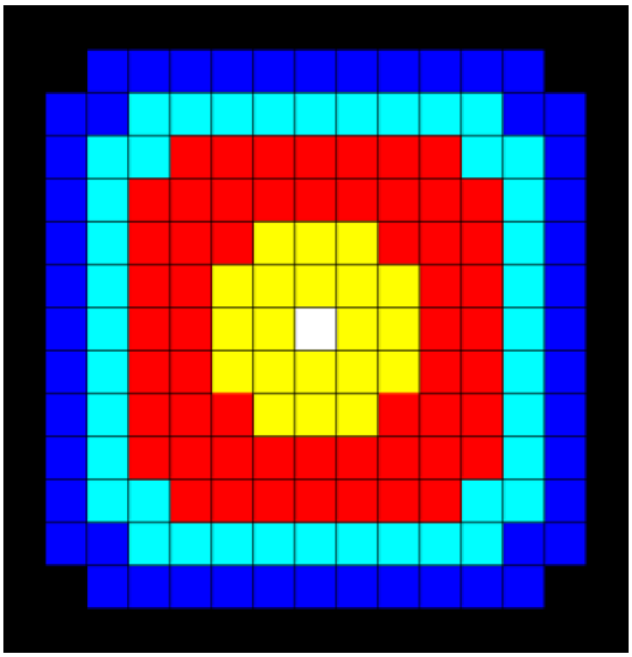

Abbildung 2: Wärmeleitungsgleichung

Zum Anfang beträgt das mittlere Subquadrat (siehe in Abbildung 2: weiße Subquadrat) die höchste Temperatur und die Temperatur der Ränder wird dauerhaft auf 0 gehalten (siehe im Abbildung 2: schwarz).

Für die weitere Bestimmung des zeitlichen Verlaufs der Temperaturverteilung, haben wir uns für die Finite-Differenzen-Methode entschieden. Diese ist ein nummerisches Verfahren zur Lösung partieller Differenzialgleichung. Diese ermöglicht es, dass wir die Temperatur der einzelnen Subquadrate bestimmen können. Dabei hängt der Zustand des Subquadrates (zum Zeitpunkt t+1) von den Zuständen des selben Elements und den Zuständen seiner Nachbarelemente ab. Die neuen Zeitpunkte werden solange berechnet, bis das Ergebnis konvergent bleibt.

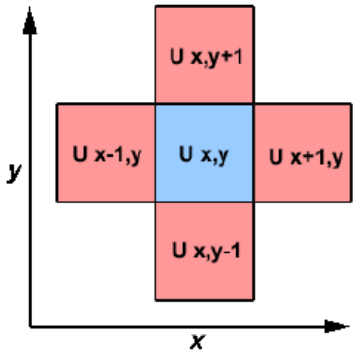

Abbildung 3: Grundidee der Gleichung

Die Formel:

$$u_{x,y}^{t+1} = u_{x,y}^{t} + c_x(u_{x+1,y}^{t} + u_{x-1,y}^{t} - 2u_{x,y}^{t}) + c_y(u_{x,y+1}^{t} + u_{x,y-1}^{t} - 2u_{x,y}^{t})$$

C_x, C_y ist eine Konstante, die die Geschwindigkeit der Wärmeausbreitung beeinflusst.

3 Umsetzung des Programms

Die Implementierung unseres Projektes besteht aus zwei Teile. Der Eine ist für die Umsetzung der Gleichung bzw. zur Berechnung der Temperatur verantwortlich und der andere für die grafische Präsentation. Nachdem die einzelnen Bereiche abgearbeitet wurden, haben wir diese zusammengeführt (Abbildung 4).

Abbildung 4: Umsetzung des Programms

3.1 Umsetzung der Gleichung

Zuerst wurde ein serielles Programm, dass für jeden Zeitpunkt den Zustand berechnet, geschrieben. Dabei können folgende Parameter in der main Datei vom Benutzer eingegeben werden:

- Kästchenanzahl
  → nur Eingabe von ungeraden Zahlen möglich, damit man das mittlere Kästchen identifizieren kann.
  Im Abbildung 5 sieht man, dass hier die Kästchenanzahl 3 entspricht (x und y Richtung). Man kann somit die Mitte (rot) sofort erkennen. Falls eine gerade Zahl eingegeben werden sollte, so wird diese sofort mit 1 addiert.

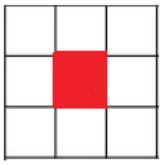

Abbildung 5: Beispiel Kästchenanzahl = 3

- Kästchenbreite
- Wärmeausbreitungskoeffizient (in der Formel c)
- Maximale Temperatur des mittleren Kästchen

Bei Start wird das Feld mit dem heißestem Punkt in der Mitte und den konstanten mit den Wert 0 an den Rändern initialisiert. Der Zustand der anderen Kästchen wird berechnet. Alle Werte werden nach und nach an SDL Kästchen-Koordinaten (siehe 3.2) zugewiesen und nach jeder Darstellung neu berechnet. Am Ende der Implementierung wurden auch zusätzliche Hitzequelle durch Mausklicks eingebaut.

Die Umsetzung findet sich in der Datei Heatfield.cpp und Heatfield.h.
Dabei werden erst einmal die Koordinaten für die Kästchen ermittelt und danach die Koordinaten für die Ränder. Dabei entspricht Breite + Breite/5 immer dem Abstand von einem Kästchen zum nächsten inklusive Rand. Insgesamt mussten wir dann noch +1 rechnen, da wir ebenfalls einen Außenrand haben wollten. Im Anschluss weisen wird die Temperaturwerte zu. Dadurch das wir die Werte absteigend zuweisen wollten, also z.B. bei 5 Kästchen der heißeste Punkt in der Mitte 255 und die Kästchen links daneben 127,5 und 0, hat sich dieses Problem als nicht trivial herausgestellt. Die einzige Lösung die wir dafür fanden, war die Werte oben links und unten rechts des Feldes zu berechnen, da wir für diese ein Schema festgestellt haben, und danach die Berechneten Werte jeweils für Oben rechts bzw. für unten links zu spiegeln. Danach fügen wir alle Werte zusammen und ordnen Sie der Reihe nach im Vektor.

3.2 Umsetzung der grafischer Oberfläche

Die grafische Oberfläche haben wir mithilfe von SDL2 dargestellt. Dies ist eine freie Multimedia-Bibliothek für verschiedene Plattformen, die in der Programmiersprache C/C++ realisiert wird. Sie stellt eine Schnittstelle (API) für Grafik-, Sound- und Eingabetasten bereit. Als Voraussetzung für die Verwendung von SDL haben wir erst einmal folgende SDL-Bibliotheken installiert

- libsdl2-dev
- libsdl2-image-dev
- libsdl2-ttf-dev

Für die Umrandung zwischen den Kästchen haben wir festgelegt, das es $\frac{1}{5}$ von eingegebener Breite entspricht. In Abbildung 6 sieht man ein Beispiel, wenn der Benutzer 11 Kästchen mit Breite von 30 haben möchte. Die grünen Linien entsprechen dabei dem Rand.

Nach der Berechnung von Temperaturwerten werden den Kästchen unterschiedliche Farben zugewiesen. Rot erhält das mittlere Kästchen mit der heißesten Temperatur und der Rand erhält dann die Farbe schwarz. Die Zwischenkästchen variieren dann zwischen hell- bis dunkelrot.

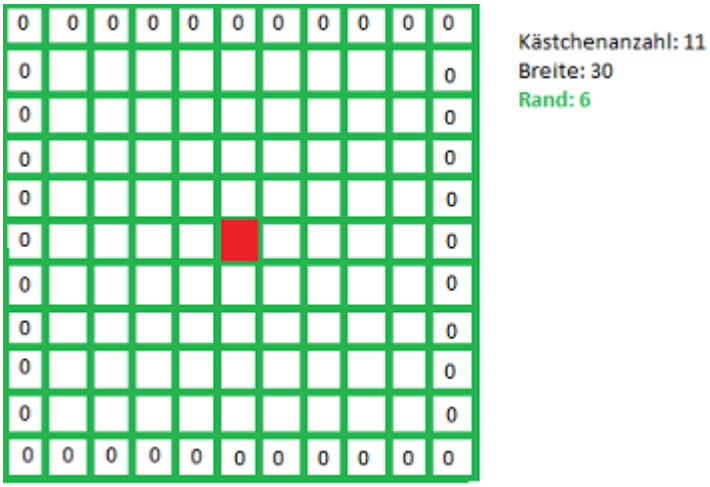

Abbildung 6: Beispiel mit 11 Kästchen

Für die Speicherung der Kästchen und Ränder Koordinaten im Vektor wird eine „for-Schleife“ verwendet. Die Rändergröße ist mit Kästchengröße/5 festgelegt.

Die Koordinaten werden vor den Zeichnen den struct „fillRect“ (x,y, Breite und Höhe) übergeben.
Das Rendern auf den Bildschirm erfolgt mittels der SDL-Funktion „SDL_SetRenderfillRect“.
Die Initialisierung und die Methoden befinden sich in den Dateien View.cpp und View.h

3.3 Parallelisierung

Das wichtigste Ziel des Projektes war es, den von uns geschriebenen Algorithmus mit einer API für Parallelität zu implementieren, um die gegebenen Ressourcen optimal auszunutzen (Threads, Prozessorkerne). Die Ergebnisse davon sollen dabei mit sequentieller Variante verglichen werden, um die Optimierung nachzuweisen.
Als Schnittstelle haben wir uns für OpenMP entschieden. Hierfür bietet sich der Befehl #pragma omp parallel for an, welcher voraussetzt, dass die folgende Code-Zeile eine for-Schleife ist. Die einzelnen Iterationen der folgenden for-Schleifen werden dann auf die zur Verfügung stehenden Prozessorkerne verteilt. Dies hat uns für die Berechnung der neuen Temperaturwerte sehr genutzt, da jeder Kästchenwert unabhängig von den anderen berechnet werden kann. Auch die Positionsüberprüfung der Maus lies sich gut parallelisieren. Die Initialisierung ist stark von der Reihenfolge abhängig, und somit nicht parallelisierbar.

4 Auswertung

Die sequentielle und die parallelisierte Version wurde auf einem Laptop getestet und die Ergebnisse wurden dann ausgewertet.

Notebook:

- Prozessor: Intel Core i3-2350m
- CPU 2,3 GHz * 2 Kerne
- Anzahl der Threads: 4
- Typ des BS: 64 Bit, , Ubuntu 15.04
- RAM: 4 GB

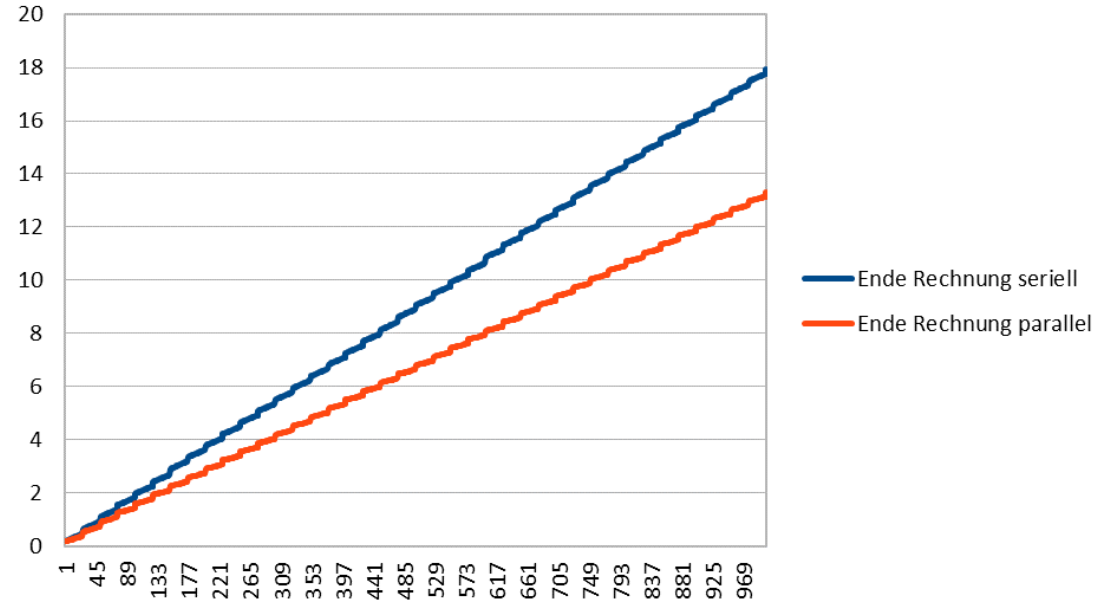

Abbildung 7: Diagramm für 203 Kästchen, Breite = 6

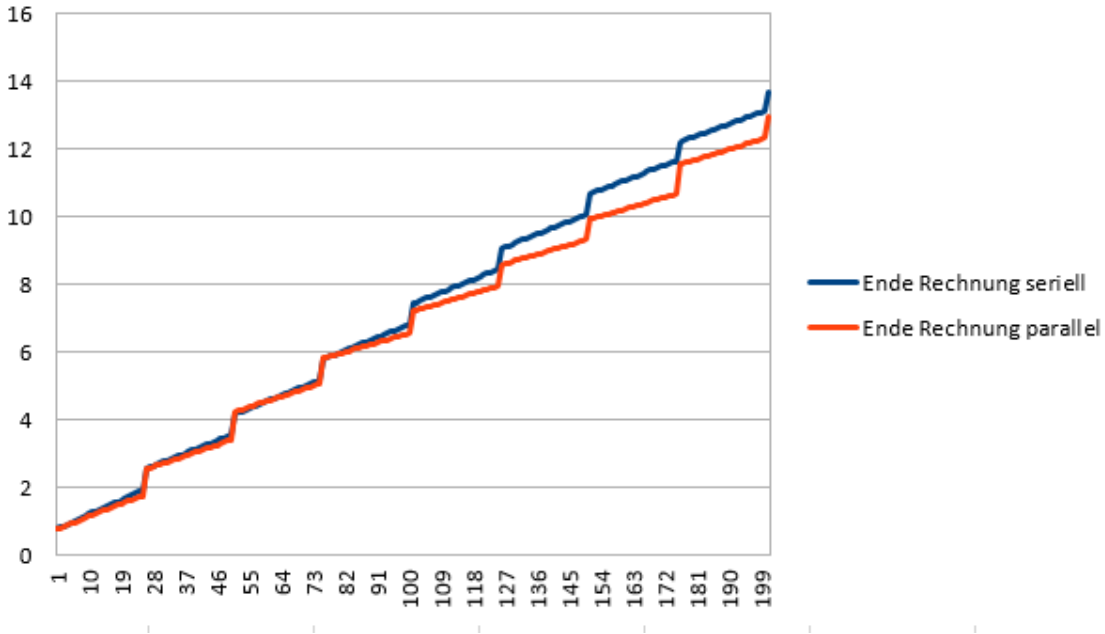

Abbildung 8: Diagramm für 351 Kästchen , Breite = 6

Durch die Parallelisierung ist ein Anstieg der Performance zu verzeichnen, welcher von uns durch DIpS (Durchschnittliche Iterationen pro Sekunde) verglichen wurde.
Jedoch haben wir eine bessere Parallelität erwartet.
Wir vermuten, dass es nicht immer lohnt jeden Abschnitt vom Programm zu parallelisieren. In unserem Fall gab es Probleme beim parallelisieren der SDL Darstellung und

und die Parallelisierung der Initialisierung hat so viel Overhead aufgebaut, das dass Programm dadurch langsamer lief.

5 Quellen

- http://www3.math.tu-berlin.de/ppm/skripte/fdm1.0.pdf, 21.02.2016
- https://de.wikipedia.org/wiki/W%C3%A4rmeleitungsgleichung, 21.02.2016
- http://www.math.uni-hamburg.de/home/oberle/skripte/diffgln/dgl2-d-07.pdf, 21.02.2016
- http://www.mathematik.uni-wuerzburg.de/~borzi/proj_parabolic.pdf, 21.02.2016
- https://de.wikipedia.org/wiki/Finite-Differenzen-Methode, 21.02.2016
- http://www3.math.tu-berlin.de/ppm/skripte/fdm1.0.pdf, 21.02.2016