Temperaturna plošča: serijski algoritem

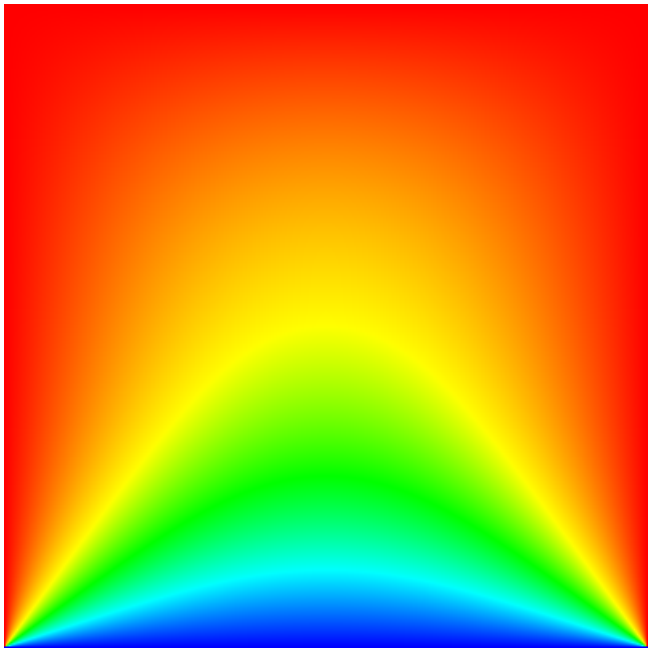
Rok Grmek, Matej Klemen

3. november 2017

1 Opis problema

V okviru predmeta Porazdeljeni sistemi rešujeva problem temperaturne plošče. Na tem problemu bova tekom semestra spoznavala različne pristope za paralelno programiranje, ampak najprej potrebujeva osnovo - serijski algoritem.

Problem temperaturne plošče je predstavljen s ploščo, ki je na treh straneh (zgoraj, levo in desno) segreta na 100 °C, na spodnji stranici pa ohlajena na 0 °C. Zanima nas, kako se v takem primeru toplota porazdeli po plošči (primer porazdelitve temperature je viden na sliki 1).



Slika 1: Primer temperaturne plošče, kjer rdeča barva prikazuje najvišjo, temno modra barva pa najnižjo temperaturo.

Iskala bova stacionarno rešitev enačbe $\nabla^2 W = f(x, y)$, pomagala pa si bova z metodo končnih diferenc. Ploščo bova najprej razdelila na mrežo točk in posameznim točkam določila

začetne vrednosti. Nato bova v vsaki točki izračunala novo temperaturo na podlagi sosednjih točk. Postopek računanja novih točk se bo ponavljal, dokler rešitev ne skonvergira.

2 Opis uporabljene metode

2.1 Algoritem

Program ima dva načina delovanja. Če je nastavljena zastavica *TIME_MEASUREMENT*, se bo program za podane argumente izvedel 100-krat in izračunal nekaj uporabnih statistik. Ta način je namenjen predvsem opazovanju hitrosti izvajanja programa. Sicer pa se bo program izvedel 1-krat in na koncu vizualiziral porazdelitev temperature na plošči. V nadaljevanju je opisan slednji način izvajanja, celotno delovanje programa pa je predstavljeno tudi z diagramom zaporedja, vidnim na sliki 2.

Algoritem sprejme 3 argumente: višino plošče, širino plošče in število iteracij (v nadaljevanju označeno s k), ki jih bo algoritem izvedel. Plošči, ki jo določata vnešena višina in širina, algoritem na vseh štirih stranicah doda pas širine 1, ki služi za nastavljanje robnega pogoja temperature. Novi dimenziji sta torej:

$$višina := višina + 2 \quad in \quad širina := širina + 2.$$

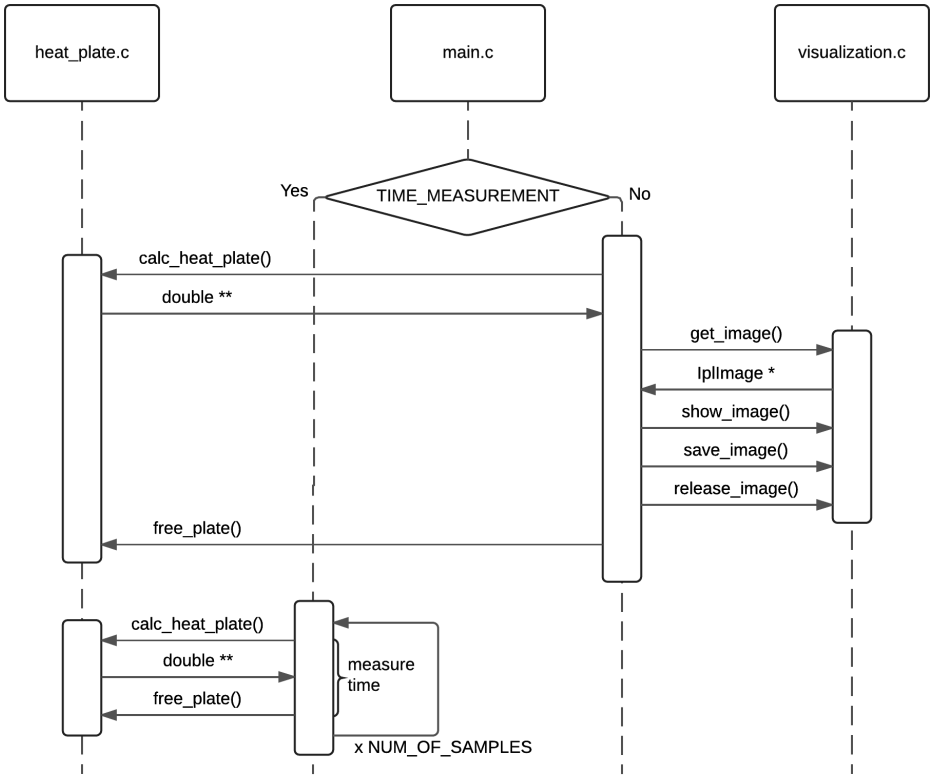
Na začetku se alocirata dve tabeli dimenzij $višina \times širina$. Ena predstavlja trenutno stanje plošče, druga pa stanje plošče v prejšnji iteraciji. Alokaciji sledi inicializacija plošče - trem stranicam (levi, desni, zgornji) algoritem nastavi temperaturo na 100 °C, eni (spodnji) pa na 0 °C. Vsem ostalim celicam plošče se zaporedno (*od zgoraj navzdol, od leve proti desni*) dodeli povprečje leve sosednje, zgornje sosednje, povsem desne ter povsem spodnje celice plošče. Eno izmed plošč algoritem izbere kot ploščo trenutnega, drugo pa kot ploščo prejšnjega stanja.

Nato sledi glavna zanka, ki se ponovi k-krat. V vsaki iteraciji gre algoritem skozi "dinamične" celice plošče (celice, ki niso del katerega izmed robov plošče) in za vsako tako celico c[i][j] v i-ti vrstici in j-tem stolpcu po naslednji formuli izračuna novo temperaturo:

$$c[i][j] = \frac{c'[i-1][j] + c'[i][j-1] + c'[i][j+1] + c'[i+1][j]}{4},$$

kjer $c'[i][j]$ predstavlja temperaturo celice v i-ti vrstici in j-tem stolpcu v prejšnji iteraciji. Ob vsakem izračunu nove temperature algoritem izračuna še absolutno razliko med trenutno (novo) in prejšnjo temperaturo ter jo v primeru, da je to v trenutni iteraciji največja izračunana absolutna razlika temperatur, shrani. Ob koncu iteracije algoritem zamenja vlogi plošč - tista, ki je do sedaj predstavljala prejšnje stanje plošče, bo v naslednji iteraciji vsebovala novo stanje plošče in obratno.

Na koncu algoritem izpiše največjo absolutno razliko temperatur in zažene vizualizacijo končnega stanja temperaturne plošče. O vizualizaciji temperaturne plošče pa je več napisano v naslednjem poglavju.



Slika 2: Diagram zaporedja, ki prikazuje delovanje programa.

2.2 Uporabljene knjižnice

Za vizualizacijo temperaturne plošče uporabljava knjižnico *OpenCV*. Najprej se pripravi prazna slika (*cvCreateImage*) z dimenzijami naše plošče, nato pa se za vsako točko na plošči pretvori temperaturo v 3 8-bitne kanale (rdeča, zelena, modra) in vrednosti prepiše na sliko. Če slika z največjo stranico presega *MAX_SIZE*, potem se jo še pomanjša (*cvResize*). Pripravljeno sliko se prikaže (*cvShowImage*) v oknu (*cvNamedWindow*, *cvMoveWindow*, *cvResizeWindow*) in shrani na disk (*cvSaveImage*). Na koncu se le še sprosti zaseden pomnilnik (*cvReleaseImage*). Primer vizualizacije je viden na sliki 1.

3 Rezultati

Program je bil testiran na sistemu, katerega specifikacije so navedene v tabeli 1. Da bi k izmerjenemu času čim manj pripomogli stroški režije operacijskega sistema, je bil sistem med testiranjem minimalno obremenjen z drugimi procesi.

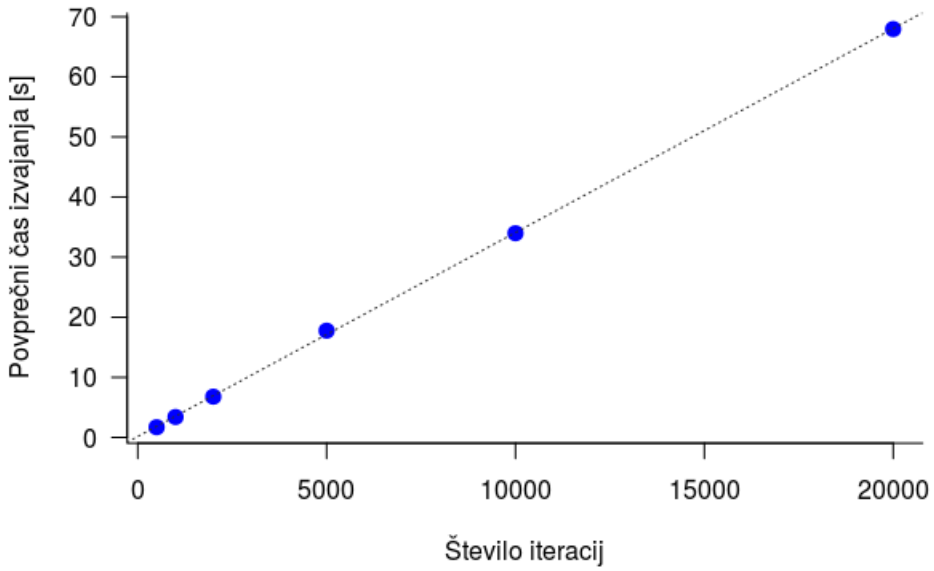
Tabela 1: Specifikacije testnega sistema.

Procesor:	Intel Core i5-4210U
Frekvenca procesorja:	1.70GHz
Število jeder:	2
Maksimalno število niti:	4
Velikost predpomnilnika:	3MB
Velikost in tip glavnega pomnilnika:	16GB DDR3
Grafična kartica:	NVIDIA GeForce 820M 2GB DDR3
Operacijski sistem:	Ubuntu 16.04

Pri testiranju sva se omejila na fiksno velikost temperaturne plošče (500×500) in spreminjala zgolj število iteracij. Za vsako izbrano število iteracij sva program 100-krat zagnala in vsakič izmerila čas izvajanja. Iz meritev sva nato izračunala povprečni čas izvajanja in standardno napako meritve, ki predstavlja razpršenost meritev okoli povprečnega časa izvajanja. Rezultati so navedeni tabelarično v tabeli 2 in z grafom, prikazanim na sliki 3.

Tabela 2: Povprečni čas izvajanja programa in standardna napaka meritev v odvisnosti od števila iteracij.

Število iteracij	Povprečni čas izvajanja [s]	Standardna napaka [s]
500	1,698	0,001
1000	3,400	0,003
2000	6,780	0,003
5000	17,757	0,103
10000	33,972	0,026
20000	67,940	0,024



Slika 3: Graf, ki prikazuje povprečni čas izvajanja programa v odvisnosti od števila iteracij.

Iz rezultatov je vidno, da je povprečni čas izvajanja linearno odvisen od števila iteracij, kar ustreza tudi teoretični časovni zahtevnosti $O(k \cdot h \cdot w)$, kjer je k število iteracij, h višina, w pa širina plošče. Točke na grafu se zelo lepo prilegajo funkciji $t(k) = 0.003393 \cdot k + 0.150600$, iz te pa lahko razberemo, da algoritem na testnem sistemu potrebuje približno 0.003393 s za posamezno iteracijo (za $h = w = 500$), 0.150600 s pa se porabi za ostale operacije, ki niso odvisne od števila iteracij.