

# Temperaturna plošča

Rok Grmek, Matej Klemen

3. november 2017

# Kazalo

<b>1</b>	<b>Serijski algoritem</b>	<b>3</b>
1.1	Opis implementacije . . . . .	3
1.2	Uporabljene knjižnice . . . . .	4
1.3	Rezultati . . . . .	4
<b>2</b>	<b>Paralelizacija s knjižnico “Pthreads”</b>	<b>7</b>
2.1	Ideja paralelizacije . . . . .	7
2.2	Rezultati . . . . .	7

## Uvod

V okviru predmeta Porazdeljeni sistemi rešujeva problem temperaturne plošče. Na tem problemu bova tekom semestra spoznavala različne pristope za paralelno programiranje.

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).



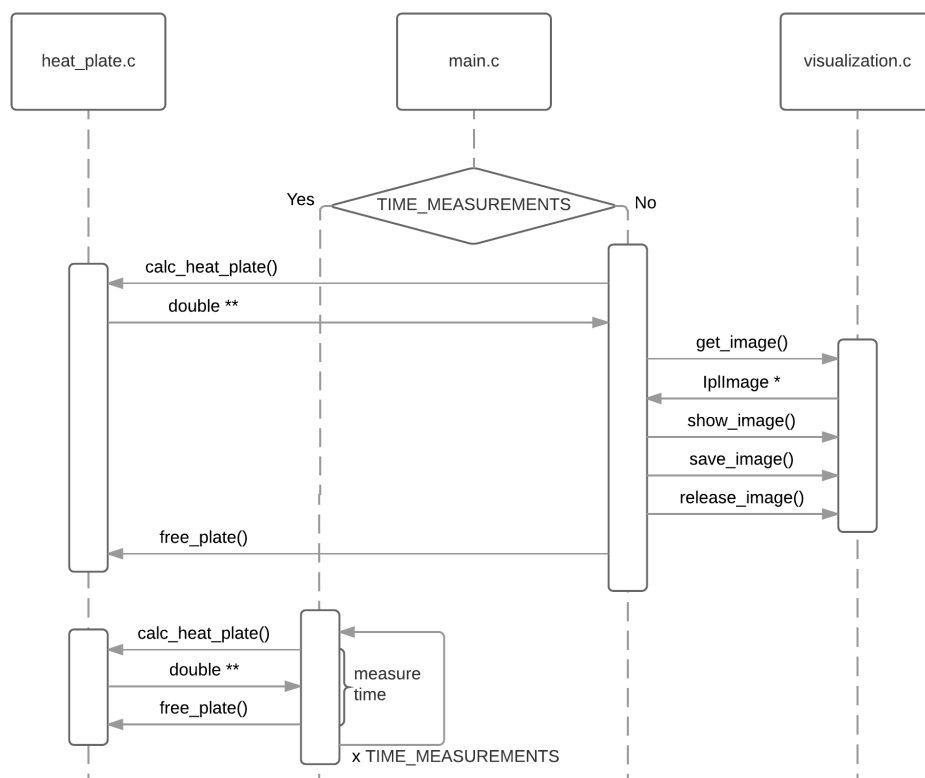
Slika 1: Primer temperaturne plošče, kjer rdeča barva prikazuje najvišjo, temno modra barva pa najnižjo temperaturo.

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.

# 1 Serijski algoritem

## 1.1 Opis implementacije

Program ima dva načina delovanja. Če je definirana vrednost *TIME\_MEASUREMENTS*, se bo program za podane argumente večkrat izvedel in izračunal nekaj uporabnih statistik. Ta način je namenjen predvsem opazovanju časa 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.



Slika 2: Diagram zaporedja, ki prikazuje delovanje programa.

Algoritem sprejme 3 argumente: višino plošče, širino plošče in mejno razliko med dvema iteracijama (v nadaljevanju označeno z  $\varepsilon$ ), na podlagi katere se zaključi računanje. 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. 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), pred prehodom v naslednjo iteracijo pa preveri, če je temperaturna razlika v tej iteraciji že manjša od  $\varepsilon$  (v takem primeru se računanje zaključí).

Na koncu algoritem izpiše število izvedenih iteracij in zažene vizualizacijo končnega stanja temperature plošče. O vizualizaciji temperature plošče pa je več napisano v naslednjem poglavju.

## 1.2 Uporabljene knjižnice

Za vizualizacijo temperature plošče uporablja 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 prepíše na sliko. Če slika z največjo stranico presega *MAX\_SIZE*, potem se jo še pomanjša (*cvResize*). Pripravljen sliki 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.

## 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.

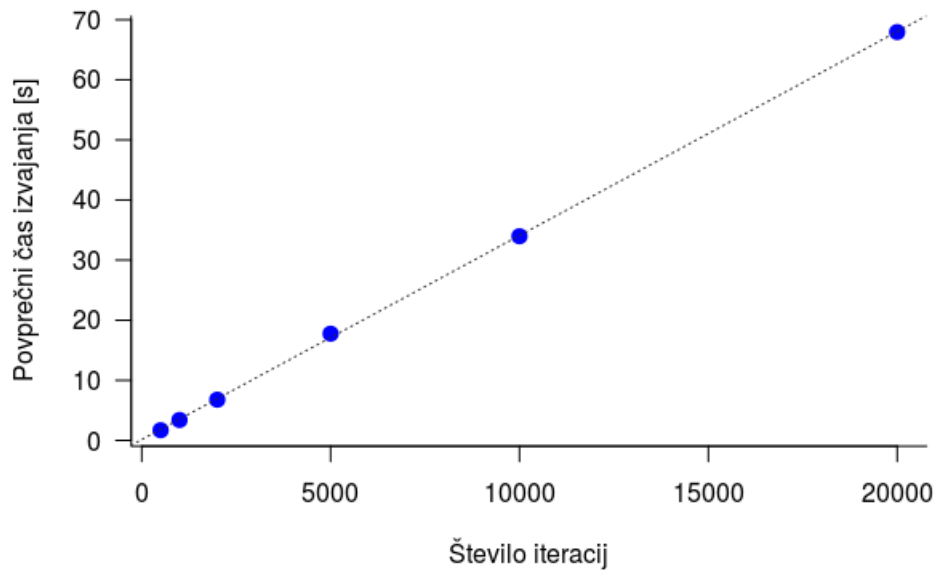
Procesor:	Intel Core i5-4210U
Frekvenca procesorja:	1.70GHz
Število jeder:	2
Maksimalno število niti:	4
Velikost predpomnilnika:	3MB
Velikost in tip glavnega pomnilnika:	16GB DDR3
Grafična kartica:	NVIDIA GeForce 820M 2GB DDR3
Operacijski sistem:	Ubuntu 16.04

TODO: NAPIŠI EPSILON, ZAMENJAJ TABELO, ZAMENJAJ GRAF

Pri testiranju sva se omejila na fiksno vrednost  $\varepsilon = 0,00??$  in na plošče kvadratne oblike, spreminjala pa sva zgolj velikost stranice. Za vsako izbrano velikost sva algoritem 100-krat zagnala in vsakič izmerila čas izvajanja. Iz meritev sva nato izračunala povprečni čas izvajanja in standardno napako (ta predstavlja razpršenost meritev okrog povprečnega časa). Rezultati so navedeni tabelarično v tabeli 2 in z grafom, prikazanim na sliki 3.

Tabela 2: Povprečni čas izvajanja programa in standardna napaka v odvisnosti od velikosti stranice.

Velikost stranice [št. točk]	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 velikosti stranice.

#### TODO: POPRAVI CELOTEN POVZETEK REZULTATOV

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  $t(k) = 0,003 \cdot k + 0,151$ , iz te pa lahko razberemo, da algoritem na testnem sistemu potrebuje približno 0,003 s za posamezno iteracijo (za  $h = w = 500$ ), 0,151 s pa se porabi za ostale operacije, ki niso odvisne od števila iteracij.

## 2 Paralelizacija s knjižnico “Pthreads”

### 2.1 Ideja paralelizacije

Shema delitve dela med niti.

Opis komunikacije, ki poteka med nitmi (lahko shematsko).

### 2.2 Rezultati

Meritve časa izvajanja za različne velikosti problema (tukaj pomoje le za najboljše število niti). Omenit je treba, da so specifikacije enake (referenca na tabelo specifikacij) in da so bile meritve opravljene na enak način kot pri serijskem algoritmu.

Tabela in graf za meritve.

Pohitritev (časi za nek primer plošče v odvisnosti od števila niti). Ali dosežete pohitritev? Kdaj je pohitritev največja in zakaj? Je to pri največjem številu niti? Ali pohitritev limitira k nekemu številu?

Učinkovitost (verjetno na enakih podatkih kot pohitritev). Komentar na spreminjanje učinkovitosti.

Tabela in graf za meritve.