Temperaturna plošča

Rok Grmek, Matej Klemen

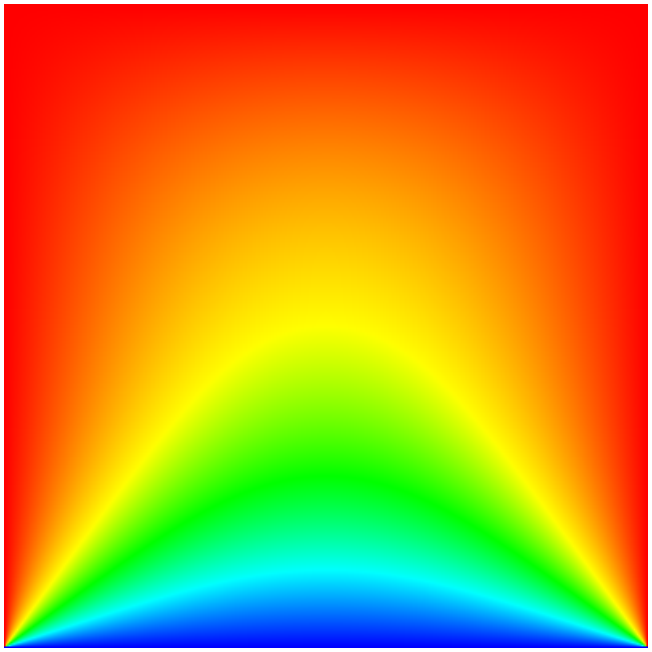
3. november 2017

Kazalo

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).



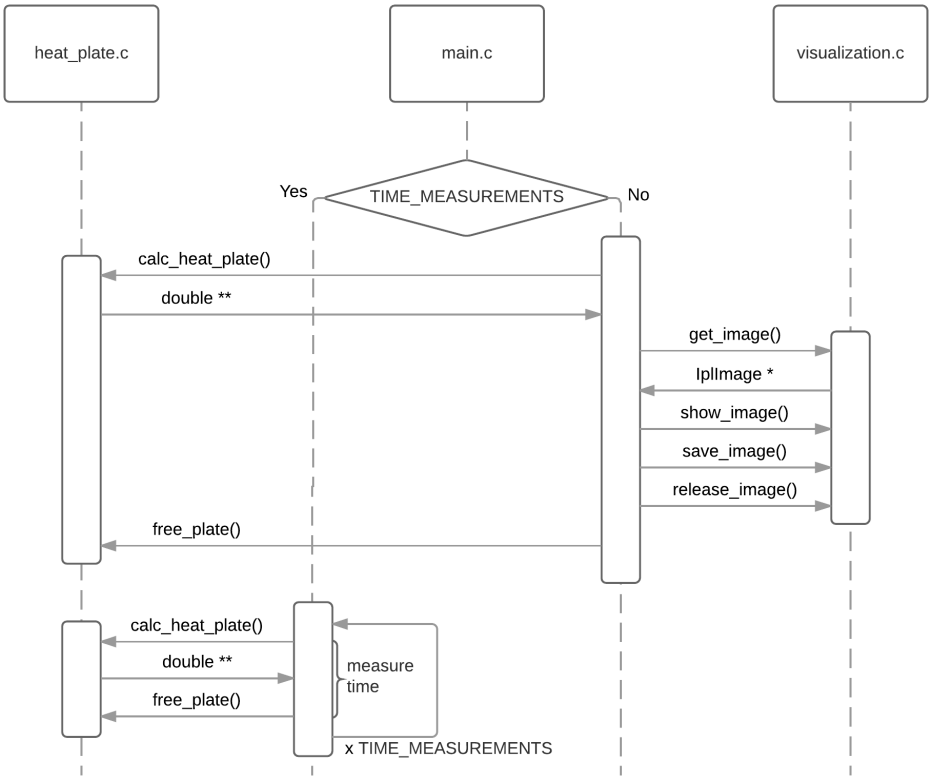
Slika 1: Primer temperaturne plošče, kjer rdeča barva prikazuje najvišjo, temno modra barva pa najnižjo temperaturo.

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 ε), 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:

$$višina := višina + 2 \tag{1}$$

$$širina := širina + 2 \tag{2}$$

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:

$$c[i][j] := \frac{c'[i-1][j] + c'[i][j-1] + c'[i][j+1] + c'[i+1][j]}{4}, \tag{3}$$

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 ε (v takem primeru se računanje zaključi).

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

1.2 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.

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.

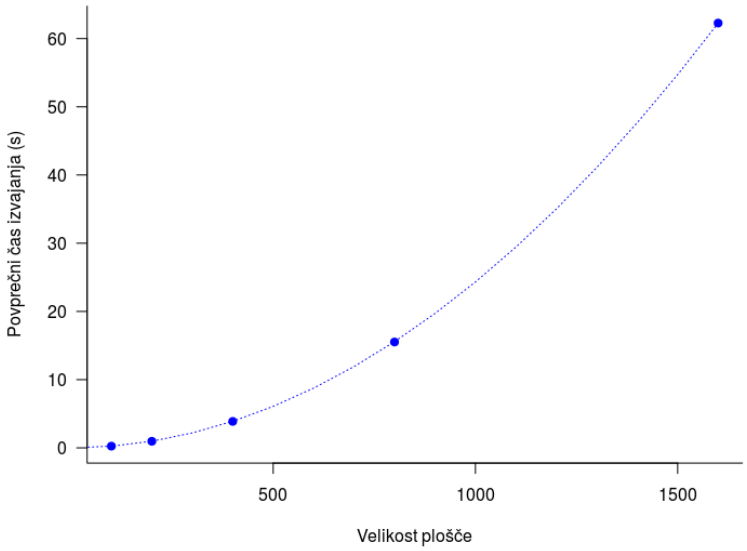
Tabela 1: Specifikacije testnega sistema.

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 vrednost $\varepsilon = 0,01$ 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]
100	0,250	0,000
200	0,965	0,001
400	3,877	0,003
800	15,516	0,006
1600	62,272	0,062



Slika 3: Graf, ki prikazuje povprečni čas izvajanja programa v odvisnosti od velikosti stranice.

Iz rezultatov je vidno, da je povprečni čas izvajanja kvadratno odvisen od velikosti stranice plošče, kar ustreza tudi teoretični časovni zahtevnosti $O(k \cdot h \cdot w)$, kjer je k število iteracij, h višina, w pa širina plošče. V našem primeru je namreč število iteracij konstantno (določeno z ε) in velja zveza $h = w$. Točke na grafu se zelo lepo prilegajo funkciji $t(x) = 2,432 \cdot 10^{-5} \cdot x^2$, iz te pa lahko razberemo, da algoritem na testnem sistemu potrebuje približno $2,432 \cdot 10^{-5}$ s za izračun posamezne točke (v vseh iteracijah skupaj).

2 Paralelizacija s knjižnico "Pthreads"

2.1 Ideja paralelizacije

Do vključno inicializacije plošč poteka vse enako kot v serijskem algoritmu, za tem pa se delo razdeli med $NUM\_THREADS$ niti. Vsaka nit v posamezni iteraciji izračuna svoj delež vrstic. Pred računanjem se meje vrstic določijo na podlagi spodnjih enačb (*index* predstavlja zaporedno številko niti iz intervala $[0, NUM\_THREADS - 1]$, ki jo dobi posamezna nit podano kot argument).

$$spodnja\ meja := 1 + \left\lfloor \frac{index \cdot (h-2)}{NUM\_THREADS} \right\rfloor \tag{4}$$

$$zgornja\ meja := 1 + \left\lfloor \frac{(index+1) \cdot (h-2)}{NUM\_THREADS} \right\rfloor \tag{5}$$

V zgornjih enačbah $(h-2)$ predstavlja število vrstic, katerih temperatura ni fiksno nastavljena. Te razdeliva kar se da enakomerno med niti, na koncu pa mejam še prištejeva 1, saj je temperatura prve vrstice fiksno nastavljena na 100 °C in se računanje začne v drugi vrstici. V takem primeu delitve se število dodeljenih vrstic med posamenzimi nitmi razlikuje za kvečjemu eno vrstico.

Ko niti poračunajo svoj del plošče, morajo počakati na preostale niti, saj je nadaljnje izvajanje programa odvisno od izračunov vseh niti v trenutni iteraciji. To je doseženo s postavitvijo t.i. prepreke (angl. *barrier*), ki nitim ne pusti opravljati nadaljnjega dela, dokler vse niti ne pridejo do mesta, kjer je postavljena.

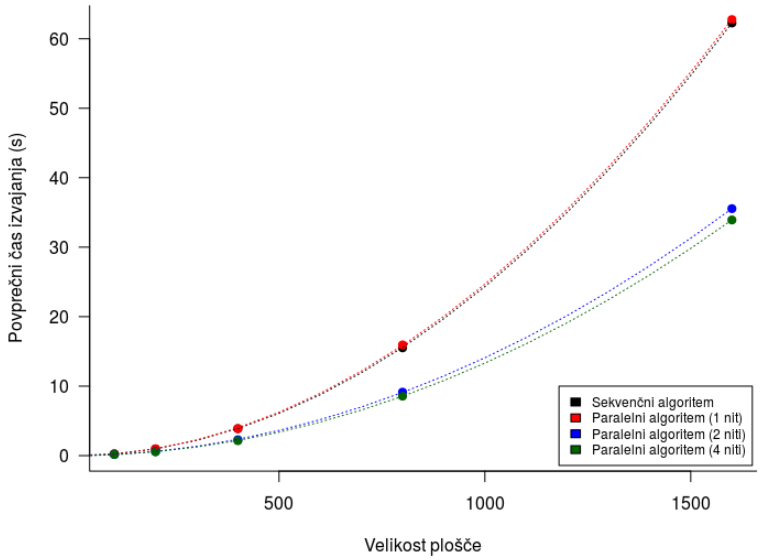
Sledi postopek sinhronizacije največje izračunane absolutne razlike temperature v trenutni iteraciji. Vsaka nit je ta podatek izračunala na svojem delu plošče in ga shranila globalno, v tem delu pa posamezna nit preveri, če so vsi izračunani lokalni maksimumi pod mejo ε. V takem primeru se računanje zaključi, niti se združijo in program poteka od tu naprej enako kot pri serijskem algoritmu. V nasprotnem primeru se niti počakajo (še ena prepreka) in nato hkrati začnejo novo iteracijo.

2.2 Rezultati

Program je bil testiran na istem sistemu (specifikacije so navedene v tabeli 1) in pri enakih pogojih kot serijski algoritem. Za fiksno vrednost $\varepsilon = 0,01$ sva izmerila čase pri različnih velikostih plošče in za različno število niti. Rezultati so vidni v tabeli 3 in na grafu 4.

Tabela 3: Povprečni čas izvajanja paralelnega programa in standardna napaka glede na velikost stranice in število uporabljenih niti z uporabo knjižnice Pthreads.

Število uporabljenih niti	Velikost stranice [št. točk]	Povprečni čas izvajanja [s]	Standardna napaka [s]
1	100	0,253	0,000
	200	0,974	0,000
	400	3,890	0,002
	800	15,935	0,071
	1600	62,760	0,052
2	100	0,167	0,002
	200	0,578	0,006
	400	2,282	0,020
	800	9,130	0,044
	1600	35,545	0,030
4	100	0,163	0,001
	200	0,558	0,001
	400	2,149	0,000
	800	8,557	0,001
	1600	33,917	0,005

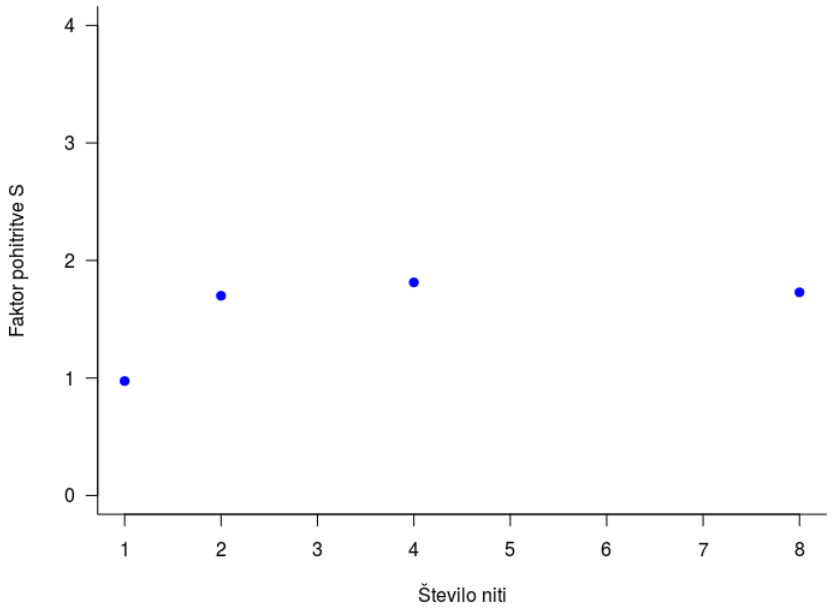


Slika 4: Povprečni čas izvajanja paralelnega programa glede na velikost plošče ob uporabi različnega števila niti.

Za lažjo analizo meritev definiramo **faktor pohitritve** S kot:

$$S = \frac{T_s}{T_p}, \tag{6}$$

kjer je T_s čas izvajanja sekvenčnega programa, T_p pa čas izvajanja paralelnega programa. Največja dosežena pohitritev je bila 1,813-krat in sicer v primeru štirih niti (toliko je tudi logičnih jeder na testnem sistemu). Ta pohitritev je sicer le malo boljša od tiste v primeru dveh niti, ampak je tak rezultat pričakovan, saj imamo le dve fizični jedri. "Hyper-threading" nam ne prinese bistvene pohitritve, saj je naše računanje zahtevno tudi iz vidika pomnilniških dostopov, tukaj pa si po dve niti delita isti predpomnilnik. V primeru še večjega števila niti nam faktor pohitritve celo pada, saj čas izvajanja še narašča zaradi režije in preklapljanja med nitmi, hkrati pa že izkoriščamo vse razpoložljive vire sistema. To je vidno tudi na grafu 5.

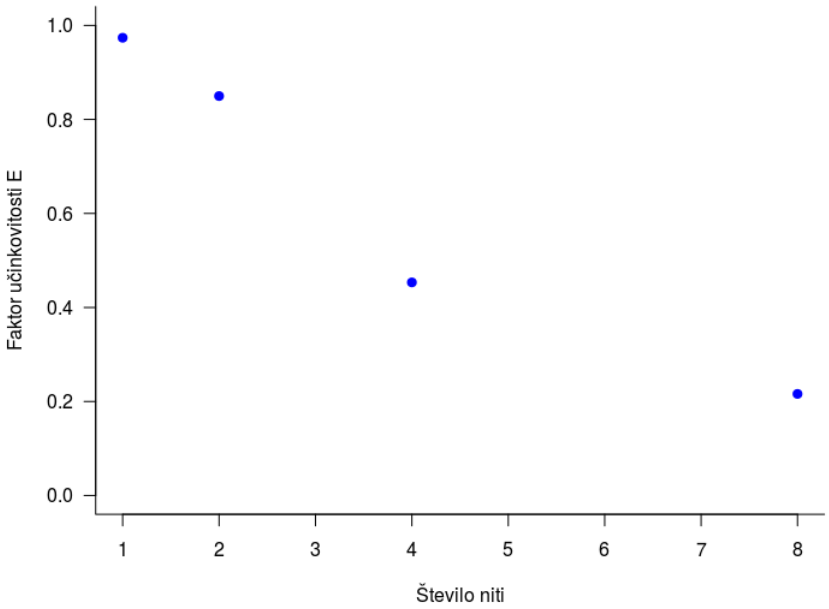


Slika 5: Faktor pohitritve v odvisnosti od števila uporabljenih niti za fiksno velikost plošče 800×800.

Smiselno je preveriti še **učinkovitost** oziroma izkoristek uporabljenih niti, E, ki jo izračunamo kot:

$$E = \frac{S_N}{N}, \tag{7}$$

kjer je S_N faktor pohitritve in N število niti. Učinkovitost je najboljša v primeru ene same niti (0,974). Rahlo slabšo a še vedno zadovoljivo učinkovitost dosežemo tudi za dve niti (0,849), pri večjem številu niti pa ta precej pade, saj smo omejeni na zgolj dve fizični jedri. Učinkovitost v odvisnosti od števila niti je vidna na grafu 6.



Slika 6: Učinkovitost v odvisnosti od števila uporabljenih niti za fiksno velikost plošče 800×800.

3 Paralelizacija s knjižnico "OpenMP"

3.1 Ideja paralelizacije

Paralelizacija poteka na podoben način kot paralelizacija s knjižnico Pthreads (za katero je ideja paralelizacije opisana v poglavju 2.1). Na začetku vsake iteracije se delo statično razdeli med vse niti - vsaka nit dobi $\frac{\check{s}tevilo\ vrstic\ -\ 2}{\check{s}tevilo\ niti}$ vrstic. Če $\check{s}tevilo\ vrstic - 2$ ni deljivo s $\check{s}tevilo\ niti$ (ostane $0 < r < \check{s}tevilo\ niti$), potem se prvim r nitim dodeli še po 1 vrstico.

Zatem vsaka nit poračuna svoj del plošče, na koncu pa niti sinhronizirajo maksimalne absolutne razlike med starimi in novimi temperaturami, ki so jih izračunale na svojem delu plošče v trenutni iteraciji. Če je največja izmed teh temperatur manjša od meje konvergence ε, se izračun konča, sicer pa se začne nova iteracija in ponovi postopek.

V primerjavi s paralelizacijo s knjižnico Pthreads je bilo tukaj potrebno manj dela. Če zanemarimo vključevanje knjižnice OpenMP in nastavitev števila niti, ki jih program uporablja, je bila paralelizacija dosežena zgolj z naslednjo vrstico:

```
#pragma omp parallel for reduction(max: max_diff).
```

To navodilo prevajalniku pove, naj poskrbi, da se delo porazdeli tako, kot je bilo navedeno v prvem odstavku tega poglavja ter da se niti na koncu počakajo in sinhronizirajo svoje lokalne podatke.

3.2 Rezultati

Program je bil testiran na sistemu, opisanem v tabeli 1.
Izmerjeni časi so zelo podobni časom, izmerjenim na programu, ki je bil paraleliziran s knjižnico Pthreads. Največjo pohitritev program doseže z uporabo štirih niti, saj ima sistem štiri fizična jedra. A ta pohitritev še zdaleč ni štirikratna, kot bi bilo pričakovati v idealnem primeru (tudi tokrat tehnologija "Hyper-threading"ni prinesla bistvene pohitritve).
Podobni rezultati so bili pričakovani, saj sta ideji paralelizacije pri obeh knjižnicah skoraj enaki - minimalna razlika se pojavi zgolj pri delitvi dela med nitmi. Poleg tega je lahko minimalni časovni pribitek ali pa odbitek posledica različno učinkovitih implementacij paralelnih konstruktov v posamezni knjižnici.

TODO (dopolni) komentar rezultatov, grafi rezultatov, primerjava s Pthreads

Tabela 4: Povprečni čas izvajanja paralelnega programa in standardna napaka glede na velikost stranice in število uporabljenih niti z uporabo knjižnice OpenMP.

Število uporabljenih niti	Velikost stranice [št. točk]	Povprečni čas izvajanja [s]	Standardna napaka [s]
1	100	0,253	0,000
	200	0,973	0,000
	400	3,983	0,007
	800	15,808	0,021
	1600	62,756	0,038
2	100	0,146	0,000
	200	0,554	0,000
	400	2,508	0,025
	800	8,798	0,001
	1600	35,274	0,002
3	100	0,184	0,000
	200	0,690	0,000
	400	2,913	0,001
	800	10,982	0,018
	1600	43,008	0,069
4	100	0,140	0,000
	200	0,539	0,000
	400	2,148	0,001
	800	8,734	0,006
	1600	34,699	0,006
8	100	0,193	0,001
	200	0,643	0,001
	400	2,284	0,002
	800	9,025	0,002
	1600	35,343	0,105