

Kazalo vsebine

[Uvod 3](#_Toc2118025112)

[Specifikacije našega procesorja 3](#_Toc1311633873)

[Sl 4](#_Toc1088884018)

[Ck 6](#_Toc1908920355)

[PSL 7](#_Toc1770273847)

[MF 9](#_Toc2013641569)

[Dokaz PSL in MF 11](#_Toc224501087)

[Meritev hitrosti 13](#_Toc378195254)

[Obupna optimizacija 16](#_Toc90856575)

[Zaključek 26](#_Toc631157143)

[Vaja 4 OpenMP 26](#_Toc930664648)

[Uvod 27](#_Toc1836349797)

[Implementacija 27](#_Toc1590239893)

[Zaključek 27](#_Toc846429976)

# Uvod

Pri tej prvi vaji smo se prvič spoznali z koncepti paralelnega in porazdeljenega računanja na praktičnem primeru. Cilji naloge so bili, da pohitrimo delovanje psl IN inMF algoritma, da več procesorskih jeder. Oz. niti porazdeljeno rešuje problem hkrati. Problemi, ki smo jih paralelno in porazdeljeno rešili so:

* Ck
* MF
* PSL

# Specifikacije našega procesorja

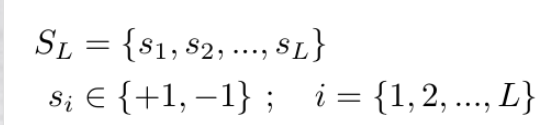
Specifikacije našega procesorja so naslednje:

Graphical user interface, text, application, email

Description automatically generated

Kot vidimo z zgornje slike ima naš procesor 4 glavna jedra in 4 dodatna navidezna jedra (hyperhtreading) oz. niti. Skupaj ima 8 logičnih niti. Torej bo zanimivo pogledat kaj se dogaja z časom računanja pri računanju do 4-ih jeder, od 4 do 8 in od 8 naprej.

# Sl



Sl je zaporedje +1 in -1 dolžine L, ki ga prejememo na vhodu v program. Sl v lagoritmu najprej zgenberiramo naključno potem pa nareidmo več iteracij cleotnega lagoritma v katergea podamo Sl vsakič najboljšega soseda PSL prejšnje iteracije. Z tem hitrje določimo boljše vrednosti MF in PSL.

Text

Description automatically generated

Naključno generiranje SL glede na podano dolžino L.

Text

Description automatically generated

Izbira naslednjega Sl glede na najboljšega soseda

Klic parallelnega izračunaPSL

# Ck

Ck je funkcija, ki se uporablja za račun tako PSL kot tudi MF

Text

Description automatically generated with medium confidence

Text

Description automatically generated

# PSL

Text

Description automatically generated

Po zgornji enačbi smo implementirali anslednjo sekvenčno kodo:

Text

Description automatically generated

Klic funckije Ck

Ko smo boljše razumeli delovanje funkcije PSL preko sekvenčne implementacije smo se lotili implementacije paralelne različice:

Text

Description automatically generated

Klic funckije Ck

# MF

Text

Description automatically generated

Po zgornji enačbi smo implementirali anslednjo sekvenčno kodo:

Text

Description automatically generated

Klic funckije Ck

Ko smo boljše razumeli delovanje funkcije MF preko sekvenčne implementacije smo se lotili implementacije paralelne različice:

Text

Description automatically generated

Klic funckije Ck

# Dokaz PSL in MF

Da dokažemo, da sta MF in PSL pravilno implementirani preverimo njuno delovanje na primeru iz eštudija:

Table, calendar

Description automatically generated with medium confidence

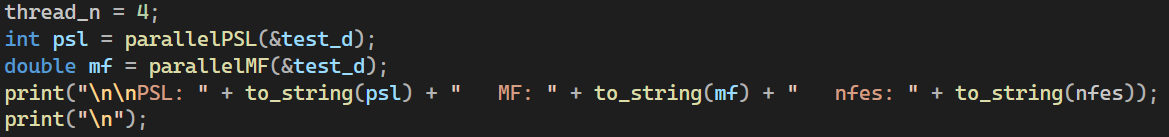
Preverili bomo za primer 8 Sequence **0x89** PSl: **2** MF **2,00**

Graphical user interface, application

Description automatically generated

Pretvorba iz hexdecimalne v binarno obliko







Kot vidimo funkciji delujeta pravilno, saj smo dobili rezultat, ki se ujema z tem na priemru iz estudija

# Meritev hitrosti

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Kot vidimo stvar sploh nedela in sem zel orazočaran nad sabo.

Nekje v paralizaciji smo mogli anredit napako, saj stvar sploh ne dela ne hitrje na 4 nitih in nasploh dela grozno počasi na 1 niti.

Razlog, da deluje tako počasi na 1 niti je verjetno uporaba vektorja. Razlog, da pa nedeluje paralelno pa nevem i nne razumem, saj je tip pralizacije neka kot smo ga imeli pri paralizaciji seštevanja vektorja pri prejšnji vaji.

# Obupna optimizacija



Text

Description automatically generated

Spremenilo smo z polja vektorjev v dinamicno oalocirano polje booleanov

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Z tem smo dosegli 10-100x pohitritev delovanja

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Z zgornjo kodo smo optimizirali, da vsaka nit opravi enako količino dela.

Paralizacija še vedno nedeluje čist, ampak tukaj so rezultati za nfes = 1000000

Text

Description automatically generated

Text

Description automatically generated

Na večjem problemu za L je paralizacija začela boljše delovat

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

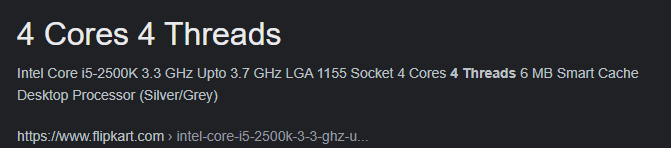
Za večje niti pa še vedno moaj implementacija ne deluje pravilno

# Zaključek

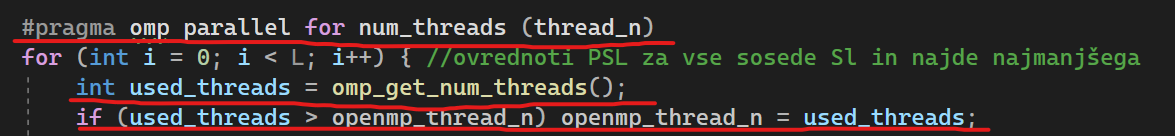
Pri tej vaji sem še boolj podrobno spoznal uporabo paralerizacije na bolj praktičnem primeru paralerizacije MF in PSL algoritma. Kljub temu, da na koncu moja implementacije ni čisto delujoča sem se veliko naučil in bom pri naslednjih vajih poskusil prej omenjene napake odpravit.

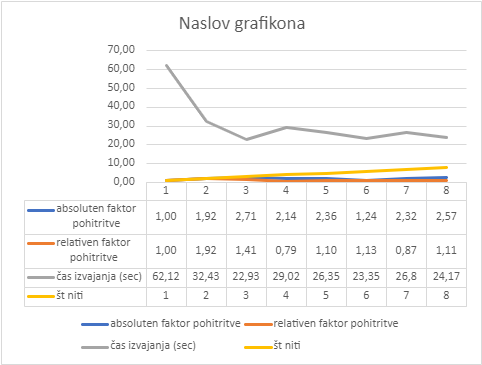
# Vaja 4 OpenMP

## Uvod



## Implementacija





## Zaključek

OpenMp je odlična knjižnica za hitro prototipiranje paralizacije, morda celo več kot prototipiranje. Deluje dokaj dobro z rezultati naše lastne implementacije paralelizacije in zelo dobro ohrani pohitritve pri večjem številu uporabljenih niti kot jih ima naš dejanski računalnik.