Paralelno programiranje
ak. god. 2023./24.

Velimir Kovačić 0036533917

Druga domaća zadaća

Igra Connect4

1. Implementacija

1.1.Priprema poslova

```
t = Tree(b, split_depth)        # izgradnja stabla do dubine paralelizacije
tasks = t.get_children()        # pribavljanje zadataka s najnize razine
s = len(tasks) // size          # kolicina posla po procesoru
```

U nekom trenutku imamo stanje ploče *b*, gradimo stablo stanja do dubine paralelizacije i skupljamo sve listove. Listovi su zadaci koji će se obavljati paralelno. Svaki processor dobije minimalno *s* zadataka.

1.2. Prijenos zadataka

```
# raspodjela poslova
split = [s for i in range(size)]
for i in range(1, len(tasks) - s * size + 1):
   split[i] += 1
accum = split[0]
for i in range(1, len(split)):
    prev = accum
    accum += split[i]
    comm.send((tasks[prev:accum], search_depth - split_depth), dest=i)

values = perform_tasks(tasks[:split[0]], search_depth - split_depth)
```

Prvo se odredi koliko će poslova koji procesor obavljati, ako nije moguće sasvim pravedno podijeliti, preferira se dati procesoru 0 manje posla da što prije krene prikupljati rezultate.

Procesor voditelj prenese sve zadatke prema toj podjeli na procesore radnike . Zadaci se prenose korištenje MPI funkcije *send*.

1.3. Obavljanje zadataka

```
# funkcija za obavljanje zadataka
def perform_tasks(tasks, depth):
    values = []
    for task in tasks:
        # gradi se stablo iz stanja i sprema vrijednost korijena
        t = Tree(task.board, depth, task.player)
        values.append(t.root.value)
```

```
    return values
```

Svaki procesor ima listu zadataka za koje gradi stablo i sprema vrijednosti.

1.4. Prikupljanje rezultata

```
# pribavljanje rezultata
for i in range(1, size):
    received_message = comm.recv(source=i)
    values.extend(received_message)

    # azuriranje vrijednosti stanja na dubini paralelizacije
    for i in range(len(tasks)):
        tasks[i].value = values[i]
    t.update_children()
```

Zadaci se prikupljaju sa svih procesora i ažuriraju se vrijednosti stabla.

1.5. Posljednja 2 koraka igre

```
Player move: 5
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  1  1  1  0
2  1  1  2  2  2  0

Computer move: 6
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  1  1  1  0
2  1  1  2  2  2  2

Winner =  computer
```

Igrač stavlja žeton u stupac 5 (indeksi počinju od 0) i računalo pobjeđuje stavivši žeton u stupac 6.

1.6. Razred Tree

Tree je razred koji služi za izgradnju i manipulaciju stablima.

```
class Tree:
    def __init__(self, board, depth, player=2):
        self.root = State(board, player, 0, depth, None)
        self.depth = depth

    def get_best_move(self)

    def get_children(self)

    def update_children(self)
```

Ima funkcije za pronalazak najboljeg poteza, dohvaćanje listova i ažuriranje svih vrijednosti u stablu nakon promjene vrijednosti listova.

1.7. Razred State

State je razred koji predstavlja čvor u stablu, nosi informacije o izgledu ploče i drugim svojstvima igre u tom trenutku.

```
class State:
    def __init__(self, board, player, depth, max_depth, col_played):
        self.board = board
        self.value = 0
        self.player = player
        self.children = []
        self.depth = depth
        self.max_depth = max_depth
        self.col_played = col_played
        self.winner = False


    def update_value(self)

    def add_children(self)
```

Ima funkcije za ažuriranje vrijednosti i dodavanja djece.

1.8. Razred Board

Razred *Board* služi za spremanje i predstavljanje stanja same ploče u nekom trenutku.

```
class Board:
    def __init__(self, rows, cols, board=None):
        self.rows = rows
        self. cols = cols
        if board == None:
            self.board = [[0 for i in range(cols)] for j in range(rows)]
        else:
            self.board = [row[:] for row in board]

    def move(self, col, player=1)

    def stalemate(self)

    def check_winner(self, col)

    def is_movable(self, col)
```

Ima funkcije za izvođenje poteza, provjeru kraja igre i provjeru mogućnosti poteza.

2. Kvantitativna analiza

2.1. Rezultati

Koristi se dubina pretraživanja od 7 i veličina ploče 6x7.

Aglomeracija na dubini 1

Broj procesora	1	2	3	4	5	6	7	8
vrijeme (s)	20,91	12,39	9,40	6,66	6,83	6,72	4,28	4,48
E_{rel}	1	0,844	0,741	0,785	0,612	0,519	0,698	0,583
s_{rel}	1	1,688	2,223	3,14	3,06	3,114	4,886	4,664

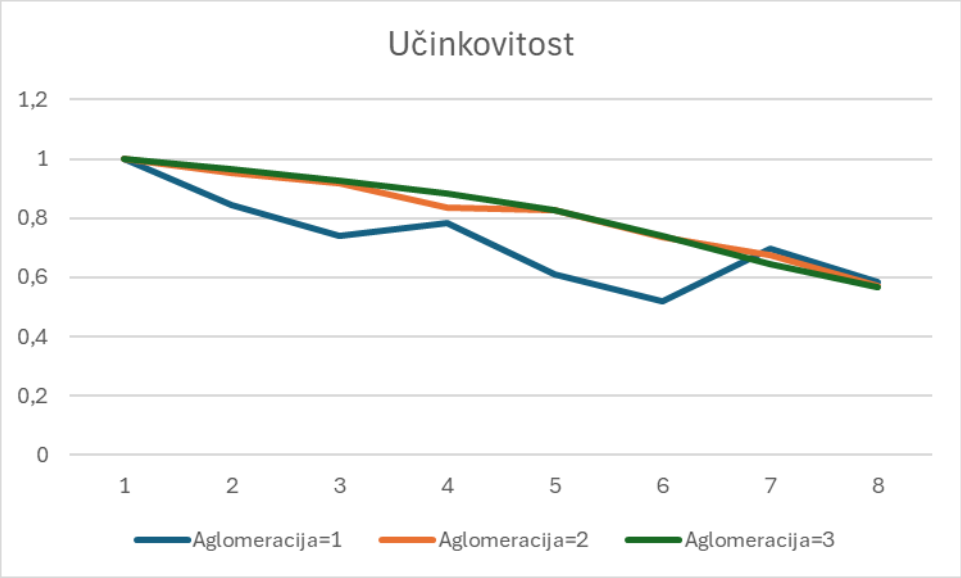
Aglomeracija na dubini 2

Broj procesora	1	2	3	4	5	6	7	8
vrijeme (s)	19,55	10,26	7,09	5,84	4,72	4,42	4,13	4,27
E_{rel}	1	0,953	0,919	0,837	0,828	0,737	0,676	0,572
s_{rel}	1	1,906	2,757	3,348	4,14	4,422	4,732	4,576

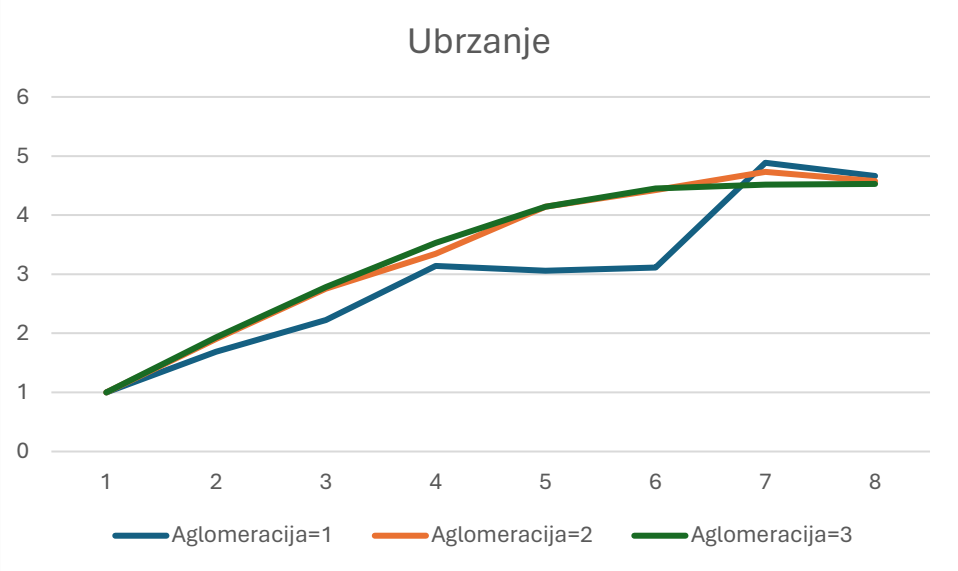
Aglomeracija na dubini 3

Broj procesora	1	2	3	4	5	6	7	8
vrijeme (s)	16,02	8,28	5,76	4,54	3,87	3,60	3,55	3,54
E_{rel}	1	0,967	0,927	0,882	0,828	0,742	0,645	0,566
s_{rel}	1	1,934	2,781	3,528	4,14	4,452	4,515	4,528

2.2. Učinkovitost



2.3. Ubrzanje



2.4. Analiza rezultata

Što je aglomeracija na većoj dubini, to je i broj zadataka veći i može se pravilnije razdijeliti po procesorima (sitnozrnatost). Dakle, možemo reći da je aglomeracija na dubini 1 krupnije zrnata od aglomeracije na dubini 3. Vidimo da pri krupnijoj zrnatosti učinkovitost mnogo više varira, dok je pri sitnijoj zrnatosti krivulja pravilnija. Uz to, učinkovitost je uglavnom veća kod sitnije zrnate podjele poslova. Pri većoj dubini aglomeracije, količina zadataka koja se mora prenijeti u komunikaciji postaje veća, a to može dovesti do smanjene učinkovitosti zbog trajanja prijenosa. Konačno, udio programa koji se ne može paralelizirati (dubina do koje pretražuje procesor 0 i dijeljenje zadataka) također dovodi do smanjene učinkovitosti.

Vidi se da u jednom trenutku aglomeracija na dubini 2 ima veću učinkovitost od one na dubini 3, a aglomeracija na dubini 1 ima čak veću učinkovitost od obje. To je zbog manjeg dijela kojeg nije moguće paralelizirati i manje količine zadataka koji se šalju u komunikaciji.

Na ubrzanju vidimo sličnu situaciju kao kod učinkovitosti, samo što je sada graf rastući. Aglomeracija na dubini 1 ima velike varijacija, a aglomeracija na dubini 3 ima najveće ubrzanje, sve dok nije prestignuta aglomeracijama na dubinama 1 i 2.