Paralelno programiranje za srednjoškolce – je li došlo vrijeme?

Predrag Brođanac
V. gimnazija, Zagreb
predrag.brodanac@skole.hr

Sažetak – Danas gotovo da i ne postoji računalo koje nema više procesora ili višejezgreni procesor. I ne samo da višejezgrene procesore imaju stolna računala, imaju ih i prijenosna računala, tableti pa čak i pametni telefoni. Višeprocesorske i višejezgrene arhitekture zahtijevaju i promjene u pristupu programiranju. Iako je paralelno programiranje postojalo i prije pojave višeproceosrskih računala, danas je ono daleko izraženije. Nakon što je paralelno programiranje integrirano kroz kolegije na višim godinama fakulteta orijentiranih na računalnu znanost, danas je ono sve češće tema uvodnih kolegija o programiranju na nižim godinama. Postavlja se pitanje je li, obzirom na dostupnost paralelnih arhitektura, a imajući u vidu buduće trendova u programiranju, došlo vrijeme da paralelno programiranje postane sastavni dio nastave informatike i u srednjim školama?

Ključne riječi: *paralelno programiranje, proces, thread*.

I. UVOD

Od 1986. do 2002. godine brzina procesora povećavala se je u prosjeku 50% godišnje (Herlihy & Shavit, 2008). Od 2002. trend povećanja brzine procesora smanjuje se u prosjeku 20% godišnje (Pachero, 2011). Tijekom godina naviknuli smo na povećanje procesorske snage te smo računala počeli upotrebljavati u raznim sferama života. Za svakodnevnu uporabu računala snaga procesora nije esencijalna. Međutim, računala se danas uvelike koriste u znanstvenim istraživanjima: istraživanja na području lijekova, dekodiranje genoma, analiza velikog seta podataka te za razne simulacije kao primjerice modeliranje u okviru biokemije, klime, nuklearnih reakcija, potresa i sl. U takvim situacijama performanse itekako dolaze do izražaja (Hochstein, Basili, Vishkin, & Gilbert, 2008), (Bohmann, 2002).

Do zastoja u rastu radnog takta procesora dolazi iz razloga što je procesor sastavljen od tranzistora. Razvojem tehnologije veličina tranzistora dogodilo se je da su se tranzistori smanjivati uz istovremeno povećanje brzine. Međutim, posljedica toga je povećanje potrošnje energije. Velika količina energije troši se na zagrijavanje tranzistora. Početkom 21. stoljeća moć hlađenja tranzistora na procesoru dosegla je svoj limit (Hennessy & Patterson , 2011). Iz navedenog razloga postalo je nemoguće povećavati brzinu i smanjivati veličine tranzistora te je snaga ovakvih procesora došla do maksimuma, a put prema povećanju performansi leži u ugradnji više jezgri unutar istog procesora (Ziwisky, Persohn, & Brylow, 2013).

Od 2005. godine većina proizvođača procesora odlučila je usmjeriti se prema proizvodnji višejezgrenih procesora odnosno višeprocesorskih računala (Computer processor history, 2019). Danas je to najčešće 4 ili 8 jezgri no pretpostavlja se da će taj broj ići i do 100 jezgri u narednim godinama (Rivoire, 2010).

Ova promjena u smjeru proizvodnje procesora uvelike je utjecala na softversku industriju. Ugradnja više jezgri u procesor odnosno više procesora u računalo neće automatski povećati brzinu izvođenja postojećeg serijskog programa. Potrebni su posebni programi koji će iskorištavati mogućnosti takvih procesora (Hochstein, Basili, Vishkin, & Gilbert, 2008). Isto tako trenutno ne postoji jednostavan način koji bi serijski program prilagodio efikasnom izvođenju na višejezgrenim računalima. Upravo stoga se je razvilo jedno novo područje programiranja: *paralelno programiranje*. Paralelno programiranje možemo definirati kao model programiranja kojemu je cilj kreirati programe kompatibilne s hardverskim okruženjem koje je u mogućnosti izvoditi programske instrukcije istovremeno (Palach, 2014). Radi se o programiranju više računala ili više procesora istog računala da simultano rješavaju zadatak (Wilkinson & Allen, 1999). Kod pisanja paralelnih programa naići ćemo na neke probleme koje nismo susretali inače kod programiranja. Ponekad ćemo trebati pribjeći nekim potpuno drugačijim algoritmima nego što smo koristili kod serijskih programa i slično. Kako dvije ili više jezgri istovremeno rade na istom programu one će morati na neki način komunicirati te se na taj način usklađivati.

Višeprocesorska/višejezgrena računala postala su stvarnost, (Niño, 2011) postavlja su pitanje do kuda smo mi kao edukatori programiranja došli? Iako je paralelno programiranje zadnjih godina napredovalo što se tiče alata i dalje je nužno da programer ima odgovarajuća znanja i iskustvo kako bi mogao kreirati aplikacije za više procesora. Kada se je pojavilo paralelno programiranje bilo je ograničeno samo na malu, usko specijaliziranu zajednicu koja je imala pristup višeprocesorskim sustavima (Ernst & Stevenson, 2008). Kao i s većinom novih saznanja kada se pojave ona se s vremenom spuste do razine sveučilišta. Tako je to bilo primjerice i s objektno-usmjerenim programiranjem 70-tih godina prošlog stoljeća ali i s paralelnim programiranjem sredinom 90-tih godina prošlog stoljeća (Berry, 1995). Svjedoci smo da se neki od tih koncepata s vremenom spuste i na niže razine obrazovanja, primjerice upravo objektno-usmjereno programiranje danas je u kurikulumima nekih srednjoškolskih programa (Kurikulum nastavnog predmeta informatika za osnovne i srednje škole, 2018), (K–12 Computer Science Framework, 2016). Jedan od zaključaka konferencije Revitalizing Computer Architecture Research, koja je

održana 2005. godine bio je da paralelno programiranje postane dostupno budućim programerima (Bruce, Danyluk, & Murtagh, 2010). Par godina kasnije (2013.) ACM/IEEE i NSF/IEEE-TCP preporučuju da se paralelno i distribuirano programiranje treba uvesti u svaki dodiplomski studij informatike (John & Thomas, Parallel and Distributed Computing across the Computer Science Curriculum, 2014). Većina istraživača smatra da paralelno programiranje treba poučavati na nižim godinama studija (Manogaran, 2013), (Bunde, Mache, & Drake, 1, 2014), (Feldman & Bachus, 1997) i mnogi drugi. Vrlo često je paralelno programiranje implementirano kao dio uvodnih kolegija u programiranje tzv. CS1 odnosno CS2 (Valentine, 2014), (Adams J. C., Injecting parallel computing into CS2, 2014). Veliki broj sveučilišta izmijenio je svoje kurikulume i/ili napravio promjene u postojećim kolegijima kako bi paralelno programiranje dobilo svoje mjesto već na samom početku studiranja (Saraswat & Bruce, 2010).

Nekolicina istraživača opisuje i svoja iskustva s paralelnim programiranjem u srednjoj školi (Ben-David Kolikant, 2003), (Feldhausen, Bell, & Andresen, 2014), (Rifkin, 1994), (Kolikant, Ben-Ari, & Pollack, 2000), (Ben-Ari & Kolikant, Thinking parallel: the process of learning concurrency, 1999), (Torbert, Vishkin, Tzur, & Ellison, 2010).

II. METODOLOGIJA PRETRAŽIVANJA RADOVA

Pretraživanje radova započeto je na ACM bazi (https://dl.acm.org). Ključne riječi za pretraživanje koje su upisane u osnovnu tražilicu bile su: *parallel programming education*. Rezultat pretraživanja bio je 166 237 rezultata koji su poslagani prema relevantnosti pretrage. Prvih nekoliko radova koji su se pojavili su redom:

- *Teaching parallel programming using Java,*
- *A breadth-first course in multicore and manycore programming,*
- *A Three-Semester, Interdisciplinary Approach to Parallel Programming in a Liberal Arts University Setting,*
- *A Three-Semester, Interdisciplinary Approach to Parallel Programming in a Liberal Arts University Setting*, itd.

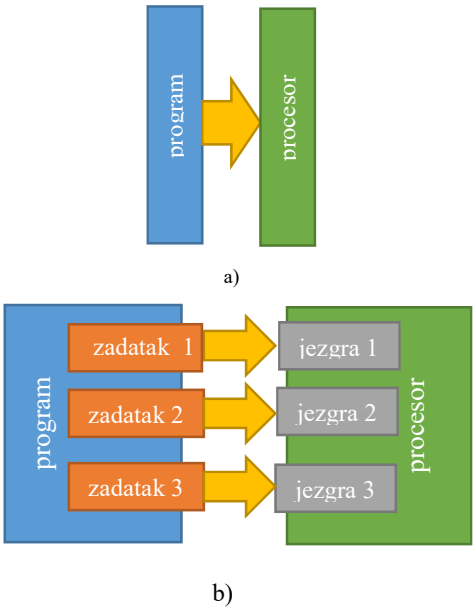
U obzir su uzimani samo oni radovi koji su u naslovu imali pojmove koji su se mogli povezati s paralelnim programiranjem, odnosno s paralelnim programiranjem u edukaciji i to s naglaskom na pisanje paralelnih programa. Za svaki od takvih radova pročitan je sažetak rada te ako se je činio relevantnim rad je preuzet, zapisan u jednu MS Excel tablicu (samo naslov) te je zajedno s relevantnim podacima upisan u MS Word u bazu referenci.

Na ovaj način pronađeno je prvih 30tak radova. Za svaki od tih radova pogledan je popis radova na koje se pripadni rad referencira te popis radova koji se referenciraju na pripadni rad. Isti postupak je primijenjen i na svaki, na ovaj način dobiveni rad. Za svaki tako pronađeni rad također je pročitan sažetak te je obrađen na način kako su obrađeni radovi pronađeni direktno u tražilici (njih 30tak).

Ovako je dobiveno ukupno 203 rada. Tijekom čitanja za svaki rad su napravljene anotacije (u MS Excelu) te je napisan kratak sažetak koji se je odnosio na temu, primjerice radi li se o radu koji se odnosi na poučavanje paralelnog programiranja u srednjoj školi, na fakultetu, radi li se o radu koji ima provedeno istraživanje i sl. Od preuzetih radova nakon čitanja odbačeno je 27 radova jer nisu bili relevantni za temu.

III. O PARALELNOM PROGRAMIRANJU

Kod standardnog serijskog programa naredbe redom izvodi jedna jezgra procesora. Kod paralelnog programa se različiti dijelovi programa tzv. zadatci mogu izvoditi na različitim jezgrama istog procesora ili na različitim procesorima.



Slika 1: Izvođenje a) serijskog programa na jednoj jezgri b) paralelnog programa na više jezgri

Grubo govoreći na paralelizaciju možemo gledati s dva aspekta:

- paraleliziranje poslova - svaki proces obavlja svoju vrstu posla. Primjerice na ispitu iz informatike bilo je pet zadataka te je ispitu pristupilo 100 studenata. Ispite ispravlja pet nastavnika te svaki nastavnik ispravlja jedan zadatak. Nakon što ispravi svoj zadatak uzme drugi ispit itd. U tom slučaju svaki nastavnik uvijek radi sličan posao, a nastavnici međusobno rade "različite" poslove.
- paraleliziranje podataka – podaci koje treba obraditi podijele se u podskupove te svi procesori obrađuju iste operacije ali svaki nad svojim podskupom podataka. U analogiji s ispravljanjem ispita to bi značilo da svaki nastavnik uzme primjerice 20 ispita i ispravi ih u potpunosti.

Temelje suvremenih računala postavio je John von Neumann. Prema von Neumannovoj arhitekturi računalo se sastoji od glavnog spremnika, centralnog procesora (CPU) te veza između spremnika i centralnog procesora. Glavni spremnik sastoji se od niza memorijskih lokacija u koje se pohranjuju podaci odnosno instrukcije koje centralni procesor treba izvoditi. Ovakav princip osnova je svakog modernog računala. U novije vrijeme računala se mogu sastojati od više centralnih procesora koji mogu istovremeno izvoditi operacije te na taj način ubrzati izvođenje programa. Osim toga danas se računala mogu povezivati i u računalne mreže te više računala unutar neke mreže može istovremeno izvoditi neki zadatak.

Obzirom na broj instrukcija koje se mogu istovremeno izvoditi te izvore podataka koje treba obraditi nizom instrukcija, četiri su osnovne vrste arhitekture računala.

Ove arhitekture poznate su i kao Flynnova taksonomija (Flynn, 1972):

- **Single-Instruction, Single-Data (SISD)** – standardno računalo prema von Neumannovom modelu. Računalo s jednom procesorskom jedinicom koje ima pristup jednom spremniku. U nekom trenutku izvodi se samo jedan program. U svakom koraku računalo izvodi jednu instrukciju programa nad odgovarajućim podacima. Kod ovakvih računala nema paralelizma,
- **Multiple-Instruction, Single-Data (MISD)** – računalo s više procesorskih jedinica od kojih svaka može izvoditi različite programe, koji su smješteni u njihovim lokalnim memorijama ali podatke dobivaju iz jednog centralnog spremnika. U svakom koraku svaki procesor učitava isti set podataka iz zajedničkog spremnika te programsku instrukciju iz vlastitog spremnika te izvodi instrukciju nad učitanim setom podataka. Radi se o vrlo rijetkoj arhitekturi koja ima izuzetno rijetku primjenu,
- **Single-Instruction, Multiple-Data (SIMD)** – računalo koje ima više procesora od koji svaki ima vlastiti pristup centralnom spremniku ali je samo jedan programski spremnik. U svakom koraku svaki procesor dobavlja različite podatke iz centralnog spremnika ali istu programsku instrukciju iz programskog spremnika
- **Multiple-Instruction, Multi-Data (MIMD)** - više procesora od kojih svaki ima svoj programski spremnik i vlastiti pristup centralnom spremniku s podacima. U svakom koraku svaki procesor može izvoditi svoju instrukciju nad svojim setom podataka.

Današnja višeprocesorska računala bazirana su na MIMD arhitekturi (Rauber & Runger, 2013).

Mnoštvo je različitih hardverskih arhitektura ali i pristupa paralelnom programiranju unutar jedne vrste paralelne arhitekture (Hyde, 1989). Shodno tome trenutno postoji mnoštvo modela za poučavanje paralelnog programiranja i nije vidljivo koji od modela će prevladati (Rivoire, 2010).

Kao i kod poučavanja ostalih koncepata programiranja programsko okruženje je važno ali ono ne smije biti osnova, a osnovni razlog za to su (Gross, 2011) česte promjene u programskim okruženjima. (Ivanov, 2012) ide i korak dalje te smatra da se radi o jednom od najdinamičnijih područja računalne znanosti. (Carro, Herranz, & Mariño, 2013) smatraju da je bolje poučavati koncepte paralelnog programiranja nego se vezati uz programsko okruženje.

(Brown, Adams, Bunde, Mache, & Shoop, A stratified view of programming language parallelism for undergraduate CS education, 2012) govore o 4 razine modula za poučavanje paralelnog programiranja:

- biblioteke na vrlo niskoj razini bliskoj hardveru i operacijskom sustavu,
- biblioteke na višoj razini koje imaju sloj za apstrakciju i upravljanje,
- programski jezici koji podržavaju paralelno programiranje,
- okruženja za produktivno paralelno programiranje (visoka razina).

Tri su osnovne kategorije paralelnih sustava na hardverskoj razini (Adams J. C., Injecting parallel computing into CS2, 2014):

- **sustavi s dijeljenom memorijom** (*eng. shared-memory systems*)
- **sustavi s distribuiranom memorijom** (*eng. distributed-memory systems*)
- **mješoviti sustavi** (*eng. hybrid systems*)

III.I. SUSTAVI S DIJELJENOM MEMORIJOM

Kod sustava s dijeljenom memorijom imamo jedno računalo koje se sastoji od dva ili više procesora. Osim toga svaki procesor može se sastojati od dviju ili više jezgri. Većina današnjih računala pa čak i tableta i mobitela raspolaže procesorima koji imaju dvije ili više jezgri, tzv. *višejezgreni procesori*. Radi se o jednom procesoru koji unutar sebe imaju više jedinica koje mogu istovremeno izvoditi operacije – *jezgri*. Nekoliko je različitih arhitektura višejezgrenih procesora no većina njih ima vlastitu L1 priručnu memoriju dok su ostale (L2 i L3) memoriji zajedničke te je zajednička i sabirnica na razini procesora ali i sve ostalo (centralni spremnik, sabirnica računala) (Wolffe & Trefftz, 2009).

U višeprocesorskim sustavima procesori imaju u potpunosti neovisnu priručne memorije te zajednički centralni spremnik.

Bilo da se radi o višeprocesorskim ili višejezgrenim arhitekturama na istom računalu nazivamo ih arhitekturama s dijeljenom memorijom (*eng. shared memory*). Kod takvih arhitektura imamo jedan centralni spremnik, koji ponekad zovemo i globalni spremnik, na kojega je direktno spojeno više procesora ili više jezgri. Komunikacija među procesorima odvija se putem zajedničkih dijeljenih varijabli na način da svi procesi imaju pristup istim memorijskim lokacijama.

S aspekta programiranja nekoliko je programskih sustava koji omogućavaju programiranje na ovakvim arhitekturama:

- multithreading – program se sastoji od više threadova koji međusobno komuniciraju dijeljenom memorijom. Threadove podržavaju C++11, Java, Python i sl.
- multiprocessing – program se sastoji od više procesora koji komuniciraju slanjem/primanjem poruka. Multiprocessing podržavaju C (kao MPI), Erlang, Scala, Python i dr.

Danas se, kada govorimo o paralelnom programiranju s dijeljenom memorijom možda najčešće koristi C u kombinaciji s OpenMP.

Za paralelno programiranje s dijeljenom memorijom može se upotrebljavati i grafički procesor, a neki od programskih sustava koji omogućavaju uporabu grafičkog procesora za paralelno programiranje su: CUDA, OpenMP, OpenACC i sl. (Adams J. C., Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns, 2015).

.

III.II. SUSTAVI S DISTRIBUIRANOM MEMORIJOM

Višeračunalne arhitekture nazivamo još i arhitekture s distribuiranim spremnikom (*eng. distributed memory*). Ovakve arhitekture sastoje se od niza procesorskih jedinica koje nazivamo čvorovi (*eng. nodes*). Čvorovi su međusobno povezani mrežnim sustavom koji omogućava

razmjenu podataka među čvorovima. Svaki čvor je samostalna cjelina koja ima svoj procesor, centralni spremnik te ostale potrebne elemente. Podaci su u takvim arhitekturama pohranjenu u lokalni spremnik svakog čvora. Razmjena podataka među čvorovima odvija se slanjem poruka (*eng. message passing*).

Jedna od poznatijih višeračunalnih arhitektura je klaster (*eng. cluster*).

Jezici koji eksplicitno imaju podršku za sustave s distribuiranom memorijom su Earlang i Scala. Tradicionalni jezik koji omogućava komunikaciju razmjenom poruka je C s MPI te jezici koji eksplicitno podržavaju MapReduce/Hadoop (primjerice Java, Python i slično). U praski su najčešće korišteni C s MPI te za velike količine podataka MapReduce/Haddop (Adams J. C., Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns, 2015).

III.III. MJEŠOVITI SUSTAVI

Kod mješovitih arhitektura imamo više multiptocesorskih računala koja su spojena računalnom mrežom.

Programska podrška za ovakve sustave temelju se najčešće na MPI + neki sustav, pri čemu je MPI zadužen za distribuciju procesa među čvorovima te omogućava komunikaciju, a onda se na razini čvora koristi proizvoljan softver. Neke od kombinacija su: MPI + OpenMP, MPI + CUDA i sl. (Adams J. C., Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns, 2015).

Malo drugačiju sistematizaciju pristupa paralelnom programiranju opisuju (Wolffe & Trefftz, 2009). Osnovna podjela bila bi na:

- single instruction multiple data (SIMD) - pogodan je za grafičke procesore,
- sustavi s dijeljenom memorijom i multithreading – procesori s više jezgri. Većina današnjih programskih jezika podržava multithreading,
- razmjena poruka (*eng. message passing*) – sustavi s više čvorova od kojih svaki ima svoj adresni prostor, a komunikacija se odvija putem poruka na razini lokalne mreže,
- distribuirani sustavi – računala koja su geografski udaljena (grid ili cloud). Klijent/poslužitelj ili per-to-per aplikacije, a danas se najčešće koristi MapReduce.

IV. ISKUSTVA POUČAVANJA PARALELNOG PROGRAMIRANJA

Većina današnjih stolnih računala ima dvije ili više jezgri odnosno dva ili više procesora (Niño, 2011). Višejezgreni procesori ugrađuju se danas i u tablete, pametne telefone i sl. (Bunde, Mache, & Drake, 1, 2014). Isto tako vidjeli smo da postoji i niz programskih okruženja za paralelno programiranje (threads, MPI, MapReduce, OpenMP) u raznim suvremenim programskim jezicima (Python, C/C++, Java, C#, Erlang, Scala,…). Međutim prije tridesetak godina situacija nije bila niti približno takva. Paralelna hardverska infrastruktura bila je skupa i teško je bilo do nje doći, a niti s programskim okruženjima za razvoj paralelnih programa situacija nije bila bolja. Problem je naravno bio i s metodikom nastave, nedostatkom materijala i sl.

IV.I. MUKE PO HARDVERU

(Schaller & Kitchen, Experiences in teaching parallel computing—five years later, 1995) opisuje iskustva uvođenja paralelnog programiranja na Rochester institutu za tehnologiju. Prvi pokušaj je instalacija mreže PC računala s Inmos T800 trasputers mikroprocesorima. Nakon prve godine korištenja ustanovljeno je da su transputeri dobar odabir međutim PC računala nemaju adekvatnu podršku za postavljene zahtjeve. Nakon dulje analize kao najpogodniju opciju odabrali su RIT Transtech mrežu koja se sastoji od 4 MCP1000 ploče koje su povezani cross bar switchevima. Na ovaj način su dobili ukupno 88 T800 TRAMova (svaki od po jedan ili četiri MB memorije), TTG3 grafički TRAM i jedan Intel i860 TRAM. Sve je organizirano na ukupno 16 mjesta (*sites*) te je omogućavalo istovremeni rad 16 korisnika, svaki korisnik na svom mjestu. Ovakav sustav autori opisuju kao dobar za poučavanje paralelnog programiranja.

Ideju kako uvesti paralelno programiranje bez bilo kakvih ulaganja u hardver opisuju (Sanders & Hartman, 1990). Autori su naime koristili resurse National Center for Supercomputer Applications (NCSA). Programi se pripremaju lokalno, a onda kada sigurno nema grešaka u programima oni se uploadaju na računala NCSA te se tamo paralelno izvode na paralelnim superračunalima. Rezultate izvođenja programa potom je moguće pohraniti u datoteku ili ih preuzeti putem mreže radi daljnje analize. Mogućnost da jedno sveučilište "pomogne" drugom u kontekstu opreme i znanja opisuje (Kurtz, Kim, & Alsabbagh, 1998). Iskustva simbioze manjeg sveučilišta s obližnjim biološkim institutom na način da se paralelno programiranje poučava na problemima iz biologije (simulacija rasta šuma, DNA analizu i sl.), a za uzvrat biološki odjel daje na raspolaganje svoja superračunala opisuje (Liu, 2008).

(Wilkinson & Allen, 1999) opisuju svoja iskustva s clusterom radnih stanica. Kao osnovnu prednost navode činjenicu da su radne stanice uglavnom dostupne na sveučilištima, a i pripadni softver je dostupan besplatno. Primjer spajanja četiri računala u cluster pod operacijskim sustavom RedHat Linux opisuje (Bohmann, 2002). Svrha je bila primjena u kemiji za kemijske proračune te su dobivena ubrzanja od tri puta. (Peck, 2010) opisuje LittleFe – kompletan cluster koji se sastoji od više čvorova koji se može koristiti za paralelno programiranja, a pri tome je još i prenosiv. karakterizira ga da se lako instalira te je jednostavan za korištenje. Omogućava tri najčešća modela paralelnog programiranja: dijeljena memorija, distribuirano programiranje te mješovito programiranje.

(Bunde, Karavanic, Mache, & Mitchell, 2013) smatraju da paralelno programiranje koje koristi grafičke procesore (GPU) te korištenje CUDA-e je komplicirano ali studentima zanimljivo jer znatno utječe na performanse programa. S druge strane većina današnjih računala, uključujući i prijenosna računala ima mogućnost programiranja GPU. Kao najvažniji čimbenik koji će studente zainteresirati za paralelno programiranje (Ernst, Wittman, Harvey, Murphy, & Wrinn, 2009) navode

uočavanje značajne razlike u performansama programa te autori smatraju da je za to najpogodnije programiranje grafičkih procesora te CUDA. Paralelno programiranje na grafičkim procesorima zagovaraju i (Mache & Karavanic, 2012).

(Touriño, Martín, Tarrio, & Arenaz, 2005) opisuju web portal na koji studenti mogu uploadati svoja paralelna rješenja te ih izvoditi na različitim paralelnim arhitekturama te na taj način mjeriti ubrzanja, vrijeme izvođenja i sl. Sličan sustav, pod nazivom OnRamp opisuju i (Foley, Koepke, Ragatz, Brehm, & Regina, 2017).

Svoja iskustva poučavanja paralelnog programiranja na više umreženih Apple uređaja (iPhone, iPod, iPad) opisuje (Rogers, 2010). Za komunikaciju među uređajima koristi se Wifi. Poslužiteljski dio aplikacije izvodi se na Apple računalu dok se klijentski dijelovi aplikacije izvode na drugim Apple računalima ili iUređajima. (Matthews, Teaching with parallella: a first look in an undergraduate parallel computing course, 2016) opisuje mogućnost paralelnog programiranja na Parallelli - uređaju nalik Raspberry Pi – jednostavno računalo s 18 jezgri. "Računalo" napravljeno od međusobno povezanih Raspberry Pi računala koji su spojeni na servo motore te se na taj način omogućava dinamična vizualizacija paralelnog računanja opisuju (Li, i dr., 2017). Računalo je između ostaloga pokazano na nekoliko izložbi te je ustanovljeno da 69% posjetitelja nakon iskustva rada s "računalom" bolje razumije paralelno programiranje.

Iskustva poučavanje paralelnog programiranja na robotima opisuju (Jacobsen & Jadud, 2005) i (Jadud, Simpson, & Jacobsen, 2008). Roboti naime istovremeno rade 3 vrste radnji: čitaju podatke sa senzora, izvode operacije nad dobivenim podacima te kontroliraju motore.

Primjer paralelnog računanja u oblaku s klasterima računala opisuje (Radenski, 2012). Autor smatra da je arhitektura za paralelno programiranje skupa, a često puta nije iskorištena cijelo vrijeme stoga je praktično za situacije kada je to potrebno unajmiti računala u oblaku. Slična iskustva opisuju i (Ceraj, Riley, & Shubert, 2009). Ovdje se radi o Amazonovom cloudu – StarHPC. Kreira se image cijelog sustava koji se instalira oblak, a koji isto tako studenti mogu instalirati na svoja računala – operacijski sustav Linux s odgovarajućim programima.

Kako napraviti vlastiti klaster računala pomoću virtualnih mašina koristeći KVM, VirtualBox ili VMWare opisuju (Shoop, i dr., 2012). Kreiranje vlastitog Beowulf clustera računala na inicijativu studenata te razvoj kroz godine opisuje (Prins, 2004).

IV.II. IZAZOVI SOFTVERA

(Higginbotham & Morelli, 1991) opisuju svoju biblioteku za pomoć pri učenju paralelnog programiranja. Biblioteka se temelji na semaforima, a osnovni cilj je da olakša uporabu "šume" funkcija operacijskog sustava (UNIX) te na taj način paralelno programiranje učini jednostavnijim. Neke od karakteristika ove biblioteke su: jednostavno okruženje za pisanje paralelnih programa, neovisno o programskom jeziku; omogućava vizualizaciju, debugiranje te praćenje izvođenja programa. Sličan pristup paralelnom programiranju na razini operacijskog sustava UNIX opisuju i (Robbins, Wagner, & Wenzel, 1989). (Schaller & Kitchen, Experiences in teaching parallel computing—five years later, 1995) naglašavaju da je paralelni hardver osnova za paralelno programiranje ali za njegovo uspješno korištenje treba i odgovarajući softver. Budući da su autori odabrali hardver RIT Transtech s njim inicijalno dolazi paralelni programski jezik Occam te jezici C i FORTRAN. Occam jezik podržan je preko Inmos Occam Toolseta koji omogućava razvoj softvera za transputere, uređivanje programa, kompiliranje te istovremeno izvođenje na više računala. Isto tako C i FORTRAN dolaze s bibliotekama koje podržavaju rad s procesima i komunikaciju. Kao razloge odabira ovih jezika autori naglašavaju da je Occam pravi paralelni jezik koji omogućava programeru da izrazi i kontrolira paralelizam na prirodan način. S druge strane C je osnovni jezik s kojim su se studenti susretali u prijašnjem obrazovanju, dok je FORTRAN važan zbog programiranja za znanstvene potrebe.

Biblioteku za pomoć pri paralelnom programiranju razvio je i (Kotz, 1995). Cilj je bio razviti sustav koji će studentima omogućiti eksperimentiranje s paralelnim konceptima bez da se puno opterećuju "mehanikom" paralelnog programiranja. Biblioteka DAPPLE omogućava paralelno programiranje u programskom jeziku C++ te omogućava programeru da manipulira kolekcijama podataka (vektorima i matricama). (Carr, Fang, Jozwowski, Mayo, & Shene, ConcurrentMentor: A visualization system for distributed programming education, 2003) razvili su ThreadMentor – C++ biblioteka koja sadrži mnoge funkcije za rad s threadovima. Osnova biblioteke je klasa Thread s pripadnim metodama za definiranje, pokretanje, zaustavljanje, dodavanje threadova. Izvođenje programa automatski šalje podatke sustavu za vizualizaciju izvođenja.

Mogućnosti paralelnog programiranja u Pascalu opisuju (Burns & Davies, 1988). Pascal-FC (Functionally Concurent Pascal) omogućava različite pristupe paralelnom programiranju: sve globalne varijable dostupne su svim procesima kao dijeljene varijable, podržava semafore i monitore te omogućava razmjenu poruka. (Davies, 1990) opisuje iskustva paralelnog programiranja u Pascalu-FC. Svoja iskustva učenja paralelnog programiranja u programskom jeziku Pascal opisuju i (Jipping, Toppen, & Weeber, 1990). Autori su implementirali unit za Pascal koji omogućava paralelno i distribuirano programiranje (Concurent Distributed Pascal – CDP). Još jedan primjer softvera za učenje paralelnog programiranja na bazi programskog jezika Pascal (Ben-Ari & Silverman, DPLab: an environment for distributed programming, 1999). Radi se o paketu DPLab koji ima okruženju s grafičkim korisničkim sučeljem za pisanje distribuiranih programa. Jezik za programiranje je Pascal koji je za ove potrebe proširen naredbama i konstruktima za paralelno programiranje. Sam softver izrađen je u Programskom jeziku Java, a kao osnova za kreiranje grafičkog korisničkog sučelja korišten je Swing. DPLab sastoji se od editora za pisanje teksta, kompilatora te sustava za komunikaciju među računalima

(Yue, 1994) smatra da je Ada dobar jezik za učenje paralelnog programiranja, a kao argumente za to navodi da se radi o popularnom (u to vrijeme) jeziku koji je

portabilan, a raspolaže i s puno paralelnih konstrukata. Ada je odabrana i iz razloga što se radi o sveučilištu u Houstonu, koje je blizu NASE, a koja je u to vrijeme koristila Adu te svi studenti u početnik kolegijima programiranja rade upravo taj programski jezik.

(Bohmann, 2002) za kemijske proračune kao paralelno okruženje koristi sustav Linda. Joyce/Linda je jezik koji je preuzeo najbolje od programskog jezika Linda te programskog jezika Joyce, a način njegove primjene u nastavi opisuje (McDonald, Teaching concurrency with Joyce and Linda, 1992). Da je Linda jezik pogodan za paralelno programiranje smatraju i (Dukielska & Sroka, 2010). S Lindom u pozadini autori su razvili okruženje JavaSpace NetBeans (JDN) čiji je osnovni cilj bio omogućiti paralelno programiranje bez da se opterećuje detaljima koji su u pozadini paralelnog programiranja. Alat ima i paralelni debuger koji može upravljati s više paralelnih procesa koji međusobno komuniciraju te može pohranjivati podatke od debugiranju.

(Bunde, Mache, & Drake, 1, 2014) smatraju da je dobar odabir za poučavanje paralelnog programiranja programski jezik Haskel. Kao argument tomu autori navode da se radi funkcionalnom jeziku koji je oslobođen nižih razina programiranja te njegove karakteristike da zaključava varijable u trenutku čitanja i pisanja vrijednosti tzv. *muttable variable*. Uporaba funkcija *par()* i *pseq()* omogućava jednostavnu paralelizaciju sekvencijalnih programa.

(Ortiz, 2011) smatra da standardni „serijski“ jezici nisu prirodni za paralelno programiranje. Kao jezik pogodan za paralelno programiranje autor navodi funkcionalni programski jezik Erlang. Kao osnovne probleme paralelnog programiranja autor navodi sinkronizacija te pristup dijeljenim varijablama i strukturama. Ukoliko ne bi bilo ovih problema paralelno programiranje bilo bi jednostavno, a Erlang nema varijable koje mogu mijenjati vrijednosti, nema dijeljene memorije te ne postoji zaključavanje varijabli. (Adl-Tabatabai, Kozyrakis, & Saha, 2006-2007) opisuju Transactional memory – oblik kontrole paralelnog programa kako bi se izbjegli nedostaci zaključavanja varijabli (*eng. lock*). Postoje dvije vrste memorijskih operacija: one koje se izvode u cijelosti (*eng. commit*) i one koje nemaju utjecaja na vrijednosti varijabli (*eng. abort*). Sve transakcije se izvode u izolaciji što znači da se nad jednim blokom memorije izvodi samo jedan skup operacija te ono što jedan proces radi s podacima biti će vidljivo tek kada se napravi commit podataka. Osnova ovakvog sustava je verzioniranje podataka i otkrivanje konflikata.

(Finlayson, Mueller, Rajapakse, & Easterling, 2015) argumentiraju prednosti programskog jezika Tetra za poučavanje paralelnog programiranja. Tetra je programski jezik sličan Python ali je namijenjen paralelnom programiranju te ima debuger za paralelno programiranje. Paralelno izvođenje programskih funkcija vrlo je jednostavno – potrebno je samo navesti direktivu. Osim toga Tetra ima paralelnu for petlju te se mogu pokrenuti procesi koji će se izvoditi u pozadini.

(McGuire, 2010) uspoređuje dva softverska sustava za paralelno programiranje: Java threadove te OpenMP. Kao osnovne prednosti threadova navodi se da su oni ugrađeni u programski jezik te se može koristiti za paralelno programiranje na višejezgrenim računalima. Kao problem threadova navodi se činjenica da nisu deterministički. S druge strane OpenMP omogućava višeprocesorsko programiranje s dijeljenom memorijom. Radi se o skupu direktiva kompilatoru kojima se naglašava da dio koda treba izvoditi paralelno. OpenMP je podržan u programskom jeziku FORTRAN i C/C++ te je na višoj razini od threadova.

Biblioteku koja daje prijedlog paralelizacije sekvencijalnog programa te upućuje ne kritične odjeljke unutar programa opisuju (Carr & Shene, A portable class library for teaching multithreaded programming, 2000). Vlastitu biblioteku za paralelno programiranje opisuju (Carr, Chen, Jozwowski, Mayo, & Shene, 2002). Biblioteka ima dvije razine apstrakcije: topologiju i kanale te podržava threadove i procese na istom ili različitim računalima putem jedinstvenog sučelja (*eng. interface*). Isto tako biblioteka se može povezati na dodatni alat kojim je moguće vizualizirati komunikaciju. DAJ (Distributed Algorithms in Java), alat za izradu i vizualizaciju paralelnih programa opisuje (Schreiner, 2002). Radi se o jednostavnom Java sučelju (*eng. interface*) koji omogućava pisanje paralelnih programa temeljenih na modelu slanja poruka. Alat omogućava da se više različitih čvorova povežu kanalima te svaki čvor neovisno obavlja svoj zadatak. Komunikacija se odvija porukama među čvorovima te je moguće postaviti vremenski limit za čekanje na određenom čvoru. Isti alat, ali s već implementiranim standardnim algoritmima opisuje (Ben-Ari, Interactive execution of distributed algorithms, 2001). Alat je namijenjen srednjoškolcima a s ciljem da razumiju paralelne algoritme te mogu kreirati i analizirati scenarije izvođenja algoritma korak po korak. Java omotač (*eng. wrapper*) za pojednostavljenje paralelnog programiranja koji od programera skriva neke detalje paralelnog programiranja opisuju (Graf & Bunde, 2014). Od paralelnih elemenata postoji samo paralelni red u koji je moguće dodavati i s njega uzimati zadatke.

(Jaswanth, Herhut, Hudson, & Shpeisman, 2012) opisuju nadogradnju JavaScripta koja omogućava paralelno programiranje – River Trail. Ovaj dodatak dostupan je kao Add-on u Mozilla Firefoxu. Kao osnovne prednosti River Triala navode mogućnost pisanja jednostavnih aplikacija bez poznavanja detalja operacija koje se izvode na razini podataka, determinističko izvođenje programa – nema istovremenog pristupa varijablama jer su sve varijable nakon postavljanja vrijednosti dostupne samo za čitanje, nema potrebe za dodatnim instalacijama i sl. ThreadMentor – alat za pomoć pri učenju paralelnog programiranja detaljno opisuju (Carr, Mayo, & Shene, Teaching multithreaded programming made easy, 2001).

Danas se za paralelno programiranje mogu koristiti velike količine podataka koje postoje u oblaku, a koje je moguće dobiti i obrađivati. Kao alat za obradu velikih skupova podataka može se koristiti Googleov MapReduce odnosno Apaceova implementacija MapReducea – Hadoop (Radenski, 2012). Phoenix++ kao implementaciju MapReduce koja se može izvoditi na lokalnim clusterima računala opisuje (Matthews, Using Phoenix++ MapReduce to introduce undergraduate students to parallel computing, 2017). Kao osnovne koristi Phoenix++ autorica navodi

svakako izvođenje na lokalnim računalima te uočavanje ubrzanja na manjim skupovima podataka (za razliku od Hadoop).

Iskustva poučavanja paralelnog programiranju za djevojčice više osnovne i srednje škole u Scratchu opisuju (Feldhausen, Bell, & Andresen, 2014). Scratch naime također omogućava istovremeno pokretanje više različitih threadova te će pri tome koristiti više jezgri procesora, ukoliko raspolažemo takvim procesorom.

Mogućnosti poučavanje paralelnog programiranja na LEGO robotima u programskom jeziku Occam-PI opisuju (Jacobsen & Jadud, 2005) i (Jadud, Simpson, & Jacobsen, 2008).

(Ben-Ari, A suite of tools for teaching concurrency, 2004) opisuje niz alata za pomoć pri paralelnom programiranju, što uključuje simulatore (Pascal i C), pakete za kreiranje scenarija paralelnih algoritama, verifikatore paralelnih programa.

Zanimljivo istraživanje proveli su (Hochstein, Basili, Vishkin, & Gilbert, 2008), a cilj je bio usporedba dva pristupa paralelnom programiranju: MPI i XMTC. Studenti su rješavali isti problem (množenje matrice vektorom) u MPI i XMTC tehnologiji te kod brzine rješavanja problema nije uočena značajna razlika kod ovih dviju tehnologija ali je kod točnosti uočena značajna razlika u korist XMTC. Isto tako MPI rješenja bila su u prosjeku 7 puta duža od XMTC rješenja.

(Asanovic, i dr., 2009) izražavaju svoju težnju za kompilatorima koji će standardni serijski kôd pretvoriti u odgovarajući paralelni, bez ikakve intervencije programera. Autori smatraju da takvi kompilatori vjerojatno neće moći dobro skalirati program između primjerice 32, 64 ili 128 jezgri. Isto tako razmišljaju o programskim jezicima gdje bi programer dao samo kostur programa, a onda bi programski jezik sam odradio detalje oko implementacije.

IV.III. ŠTO KAKO I KADA POUČAVATI?

Iako se možda na prvi pogled čini da se paralelno programiranje ne razlikuje puno od serijskog pokazuje se da studenti imaju problema s paralelnim programiranjem (Carr & Shene, A portable class library for teaching multithreaded programming, 2000). Kao osnovne razloge tim problemima autori između ostaloga navode:

- paralelno programiranje zahtijeva drugačije razmišljanje,
- ponašanje paralelnih programa je dinamičko, što između ostaloga otežava ispravljanje grešaka u programima,
- pravilna sinkronizacija programa je teška i sl.

Kod paralelnog programiranja osim standardnih grešaka koje postoje i kod serijskog programiranja pojavljuju se i neke greške koje ne postoje kod serijskog programiranja. Neke od takvih grešaka su (Choi & Lewis, 2000):

- *istovremeni pristup* (*eng. data race*) – događa se kad dva ili više threada istovremeno čitaju ili pišu na istu memorijsku lokaciju,
- *potpuni zastoj* (*eng. deadlock*) – događa se kada jedan thread čeka da se dogodi uvjet koji se nikada neće dogoditi,
- bespotrebno korištenje zaključavanja varijabli (*eng. miscellaneous*).

U istraživanju provedenom na 180 studentskih rješenja uočeno je da 23 rješenja ima grešku zbog istovremenog pristupa, a da pri tome nije upotrjebljen mehanizam zaštite varijable, 11 rješenja imalo je u svojim programima potpuni zastoj dok je 11 studenata bespotrebno u programima koristilo zaključavanje varijabli.

Mnogi dokumenti, među kojima su ACM CS2013 curriculum guidelines te NSF/IEEE-TCPP (Bogaerts, 2017) govore da je paralelno programiranje potrebno uvesti za sve studente računalnih znanosti. ACM CS2013 preporuča da se paralelno programiranje disperzira u više kolegija tijekom studija. To ima svoje administrativne izazove (Burtscher, i dr., 2015), od kojih su neki:

- identifikacija kolegija za uključivanje paralelnog programiranja – serijsko programiranje je specijalan slučaj paralelnog programiranja pa bi se paralelno programiranje moglo uključiti u početne kolegije programiranja,
- prilagodba sadržaja kolegija – uvođenje novih koncepata znači da bi neki postojeći trebali biti izbačeni,
- edukacija nastavnika i dr.

U svom istraživanju (Feldman & Bachus, 1997) pokazuju da je paralelno programiranja moguće uvesti na niže godine studija. (Plouzeau & Raynal, 1992) opisuju što bi kolegij paralelnog programiranja trebao sadržavati te smatraju da je osnova kolegija paralelnog programiranja poznavanje serijskog programiranja.

(Organick, 1985) navodi kako se često smatra da za paralelno programiranje nema dovoljno zanimljivih primjera. Autor to demantira i kao zgodne primjere za paralelno programiranje navodi i detaljno opisuje paralelno rješenje Laplaceovih jednadžbi u dvije dimenzije te paralelno množenje matrica.

(Schaller & Kitchen, Experiences in teaching parallel computing—five years later, 1995) smatraju da studentima treba ukazati na osnove paralelnog hardvera, pripadnog softvera, teorije paralelnog programiranja, a potom se treba fokusirati na izgradnju paralelnih programa. Autori su uvidjeli da za to nije dovoljan jedan kolegij već su gradivo podijelili u dva kolegija. Prvi kolegij predstavlja osnovne izazove paralelnog programiranja: paralelne arhitekture, algoritme, jezike, topologije mreže i granularnost. Isto tako unutar ovog kolegija studenti i eksperimentiraju s paralelnim programiranjem. Drugi kolegij predstavlja trendove u izgradnji paralelnih algoritama. Neke od teme iz ovog kolegija su: utjecaj modela paralelnog programiranja na dizajn programa; metrika i analiza paralelnog algoritma, složenost algoritama, sortiranje i pretraživanje, matrične operacije, kombinatorički problemi i algoritmi nad grafovima, grafičko procesiranje i sl.

(Wilkinson & Allen, 1999) za poučavanje paralelnog programiranja koriste kombinaciju PVM i MPI no za objašnjavanje algoritama koriste pseudokod. Navedeni pseudokod temelji se na programskom jeziku C, a koristi pojednostavljenu notaciju za slanje i primanja poruka. Program napisan u pseudokodu lako se može pretvoriti u PVM ili MPI. Prvi dio kolegija su osnovne tehnike paralelnog programiranja, koje uključuju: hardversku infrastrukturu, mogućnosti ubrzanja, programiranje slanjem poruka, sinkronizaciju, programiranje s dijeljenom

memorijom i sl. Drugi dio kolegija bavi se različitim algoritmima, a neki od njih su: sortiranja, matrične operacije, procesiranje slike, pretraživanje i sl.

Među nastavnicima na sveučilištu velika je dvojba: uvesti paralelno programiranje kao samostalni kolegij ili ga implementirati u različite kolegije. Strategiju integracije pojedinih tema paralelnog programiranja u pojedine postojeće kolegije opisuju (Brown, Adams, Bunde, Mache, & Shoop, Strategies for adding the emerging PDC curriculum recommendations into CS courses, 2013) te (Brown, i dr., 2010). (Prasad, i dr., 2011) smatraju da paralelno programiranje treba integrirati u postojeće kolegije iz područja: programiranja, algoritama te računalnih arhitektura. Autori smatraju da iz kolegija ništa nije potrebno izbaciti već da postojeće sadržaje treba poučavati na moderan, paralelni način. (Berk, 1996) opisuje integraciju paralelnog programiranja u kolegij Operacijski sustav. (Monismith, 2015) smatra da postoji niz kolegija u koje bi se moglo implementirati teme iz paralelnog programiranja:

- operacijski sustavi – teme vezane uz procesore, threadove, sinkronizaciju i sl.,
- kolegiji koji se bave assemblerskim jezicima te digitalnom logikom,
- kolegijima vezanim uz strukture podataka i softversko inženjerstvo.

Slična razmišljanja opisuje i niz drugih autora, od kojih su neki: (Toll, 1995), (Nevison, 1995), (Graham, 2007), (Shene, 1998), (Fekete, Teaching about threading: where and what?, 2009), (Brown, Lu, & Midkiff, 2013), (Burtscher, i dr., 2015), (Newhall, Danner, & Webb, 2017). Čitav proces traženja adekvatnih kolegija za integraciju paralelnog programiranja opisuje (Meredith, 1992). Niz detaljno razrađenih modula koje je moguće integrirati u postojeće kolegije opisuju (Brown & Shoop, Modules in community: injecting more parallelism into computer science curricula, 2011). Sličan opis modula s planom integracije opisuju i (John & Thomas, Parallel and Distributed Computing across the Computer Science Curriculum, 2014).

Učenje paralelnog programiranja u ranijoj fazi edukacije omogućava integraciju paralelnog programiranja u ostale kolegije na nižim godinama studija (Pollock & Jochen, 2001). Autori opisuju tečaj paralelnog programiranja za neprogramere. Nastava se je temeljila uglavnom na proučavanju gotovih programa. (Bogaerts, 2017) opisuje iskustva uvođenja tema iz paralelnog programiranja u početni kolegij programiranja (CS1). Kroz svoja iskustva autor opisuje metodiku poučavanja paralelnog programiranja te navodi vrlo zanimljive analogije, primjerice: ako jednoj osobi treba da iskopa rupu 100 sekundi, koliko će vremena trebati da se iskopa ista rupa ako ju kopa 10 ljudi? Ova analogija je vrlo zanimljiva jer vrijeme ovisi o veličini rupe. Ukoliko je rupa dovoljno velika da ju istovremeno kopa 10 osoba vrijeme će biti u prosjeku 10 puta kraće, međutim ako je rupa takva da na njoj istovremeno ne mogu raditi sve osobe vrijeme se možda i neće značajno smanjiti obzirom na vrijeme potrebno da rupu iskopa jedna osoba. Integraciju paralelnog programiranja u kolegij CS1 opisuju i (Bruce, Danyluk, & Murtagh, 2010). Paralelno programiranje uvodi se u kontekstu event-driven programiranja kroz animacije. Kada započne neka animacija ona se izvodi unutar posebnog threada. Isto tako moguće je imati dva elementa unutar iste animacije koji se paralelno animiraju, svaki u svom threadu (primjerice igra u kojoj žaba želi prijeći cestu s automobilima, pri čemu je svaki automobil zasebni thread). (Rague B. , 2011) smatra da nakon trotjedne intervencije paralelnim programiranjem u kolegij CS1 studenti kasnije puno bolje razumiju paralelno programiranje. (Adams J. C., Injecting parallel computing into CS2, 2014) smatra da je pravi trenutak za uvođenje paralelnog programiranja kolegij CS2. Tada naime programi koji se rješavaju postaju vremenski složeniji što je pogodno za ilustraciju ubrzanja (npr. množenje matrica). (Massung & Heeren, 2013) opisuju dodavanja paralelnog programiranja u kolegij CS2 i to na primjerima obrade slike, gdje je dobro vizualizirano paralelno izvođenje programa. Obrađivani su neki standardni postupci nad slikom: brisanje komponente zelene boje sa slike, kreiranje histograma boja, pomicanje svakog piksela slike za jedno mjesto u desno. (Grossman & Anderson, 2012) opisuju integraciju paralelnog programiranja u kolegij o strukturama podataka i algoritma te smatraju da ako studenti mogu razumjeti primjerice B-Trees strukturu ili Dijkstrin algoritam za traženje najkraćih putova u grafu, onda ne bi trebali imati problema niti s paralelnim programiranjem.

Poučavanje osnova paralelnog programiranja u okviru dva uvodna kolegija programiranja (CS1 i CS2) opisuju i (Ernst & Stevenson, 2008). Paralelno programiranje u okviru CS1 i CS2 kolegija opisuje i (Valentine, 2014) te navodi kako 100% studenata smatra da studenti informatike nakon treće godine studija trebaju znati osnove paralelnog programiranja. Iskustva integracije osnova paralelnog programiranja u dva postojeća kolegija dodiplomskog studija opisuju (Bunde, Karavanic, Mache, & Mitchell, 2013). U prvom kolegiju je to ukupno 2.5 sata (1.5 sat predavanja te jedan sat vježbi za računalima) gdje su studenti upoznati s razmjenom podataka između CPU i GPU što je potkrepljeno primjerom zbrajanja dvaju vektora, dok su u drugom kolegiju tema bila paralelna igra života. (Niño, 2011) smatra da bi paralelno programiranje trebalo integrirati u razne kolegije, počevši od kolegija CS2. Paralelno programiranje kao sastavni dio nekoliko kolegija na studiju opisuje (Kumar, 2017). Autor smatra da su za poučavanje dobri modeli:

- research-tutored – studenti rade u grupama, proučavaju dobivenu literaturu, razgovaraju o pročitanim materijalima i sl. te
- research-oriented – većinu nastavnog gradiva studenti istražuju samostalno.

Istraživanje koje je autor proveo između ostaloga ukazuje da je ovakav način integracije paralelnog programiranja u kombinaciji s modelom poučavanja od studenata dobro prihvaćen te je uočen značajan napredak u razumijevanju sadržaja paralelnog programiranja.

Promjenu kolegija Strukture podataka na način da se osnovni algoritmi prikazuju i u paralelnoj izvedbi zagovaraju (Johnson, Kotz, & Makedon, 2009).

(Hyde, 1989) smatra da bi paralelno programiranje trebalo biti poseban kolegij unutar dodiplomskog studija. Smatra da je paralelno programiranje prirodno jer je svijet koji nas okružuje paralelan, a razlog problema je možda

činjenica odabira pogrešnog programskog jezika za poučavanje paralelnog programiranja. Isto tako autor smatra da je za paralelno programiranje nužno imati vježbe za računalom te navodi niz problema koji su pogodni za poučavanje paralelnog programiranja. Ideju poučavanja paralelnog programiranja u okviru kolegija *Programski jezici*, u vrijeme kada su jezici koji podržavaju paralelno programiranje bili rijetki opisuje (Yeager, 1991). Paralelno programiranje svodi se na pisanje korutina (*eng. corutines*). (Gross, 2011) opisuje iskustva poučavanja paralelnog programiranja kao posebnog kolegija u sklopu drugog semestra. Ideja kolegija je studente poučiti tehnici paraleliziranja nekih računalnih postupaka, verifikacija paralelizacije, ukazati na važnost komunikacije, koja može biti: implicitna – pristup istim memorijskim lokacijama ili eksplicitna – slanje i primanje poruka. Iako se radi o složenim konceptima za studente, nisu uočene razlike u uspjehu na ispitu iz ovog i drugih kolegija na prvoj godini fakulteta. Kolegij paralelnog programiranja, na višim godinama studija, gdje se koriste High-Level (OpenMP) i Low-Level (PVM i MPI) jezici opisuje (Pan, Teaching Parallel Programming Using Both High-Level and Low-Level Languages, 2002). (Lin & Tatar, 2011) opisuju kolegij paralelnog programiranja koji se temelji na modelu koordinacije – više entiteta suradnički dolaze do zajedničkog rezultata pri čemu entiteti međusobno komuniciraju porukama. Kolegij paralelnog programa koji se može podijeliti i u module te integrirati u ostale kolegije opisuje (Ko, Burgstaller, & Scholz, 2013). Sličan mini kolegij u ukupnom trajanju od tri tjedna, koji također može biti i modul nekog postojećeg kolegija opisuje (Ernst D. J., 2011). Kolegij koji bi mogao biti prije svih uvodnih kolegija (CS0), odnosno neki oblik Advanced Placement predmeta za srednjoškolce, a koji obuhvaća osnove rada računala te programiranja ali i teme paralelnog programiranja opisuje (Lu, Conley, & Klein, 2014). Zanimljivo je da kolegij počinje programskim jezikom Scratch, a autori zagovaraju pristup učenju iskustvom. (Marowka, 2008) smatra da bi svaki kolegij trebao imati prefiks paralelno te da bi trebao između ostaloga poučavati i o paralelnom programiranju. Stoga autor predlaže uvodni kolegij paralelnog programiranja. Test-Driven pristup paralelnom programiranju u Javi opisuju (Ricken & Cartwright, 2010).

Detaljnu razradu kolegija paralelnog programiranja opisuju i: (Yue, 1994), (Berry, 1995), (Moore, 2000), (Gross, 2011), (von Praun, 2011), (Strazdins, 2012), (Neelima, 2017). Moderan kolegij paralelnog programiranja koji se temelji na MapReduce, BigTable database te Hadoopu opisuje (Albrecht, 2009). Kolegij se može podijeliti u dva dijela: u okviru prvog dijela studenti proučavaju literaturu, dok u sklopu drugog dijela rješavaju programske projektne zadatke. Moderan kolegij koji uključuje Java thredove, OpenMP ali i MapReduce opisuju (Shafi, Akhtar, Javed, & Carpenter, 2014).

Poučavanje paralelnog programiranja kroz izborni kolegij na dodiplomskom studiju opisuje (Rivoire, 2010). Cilj je bio dati studentima iskustvo paralelnog programiranja, no bez ograničavanja na samo jedan model. U kolegiju su se bazirali na tri modela: OpenMP, TBB (Threding Building Blocks) i CUDA. Studenti su smatrali da je najkorisniji dio kolegija bio rješavanje problema na računalu, a najzanimljivija platforma bila je CUDA. (Jackson, 1991) opisuje iskustva poučavanja paralelnog programiranja u sklopu mini-kolegija u trajanju od 5 tjedana gdje se uče koncepti paralelnog programiranja koji se ne rade u drugim kolegijima: kreiranje i uništavanje procesa, sinkronizacija, komunikacija, potpuni zastoj i sl. (Morris & Frinkle, 2014) opisuju 3 kolegija paralelnog programiranja koji se nadovezuju jedan na drugi, pri čemu je posljednji napravljen u suradnji s matematičarima te se bazira na rješavanju topoloških problema. (Bunde & Mache, Teaching concurrency beyond HPC, 2009) smatraju da su klasični problemi na kojima se poučava paralelno programiranje studentima nezanimljivi te bi trebalo tražiti primjere izvan matematike, kao primjere autori navode: umjetnu inteligenciju, renderiranje animacija, obradu velike količine podataka, kreiranje Mandelbrotovog skupa i sl.

(Pan, An Innovative Course in Parallel Computing, 2003) opisuje iskustva paralelnog programiranja kroz platforme OpenMP i MPI.

Svoja zavidna iskustva u poučavanju paralelnog programiranja kao izbornog predmeta koji se predaje u srednjoj školi od 90-tih godina prošlog stoljeća opisuju (Torbert, Vishkin, Tzur, & Ellison, 2010). Kao programski jezik koriste C, a kao alat za paralelno programiranje MPI. U drugom dijelu predmeta za paralelno programiranje koriste XMT, pthreads, OpenMP, sockete te programiranje za grafičke procesore i CUDA. Primjeri na kojima poučavaju paralelno programiranje vezani su uz pretraživanja, generiranje fraktala, stanične automate, obradu slike, matrične operacije i sl. Valja naglasiti da je ovo predmet za najbolje učenike iz programiranja. Paralelno programiranje kao izborni predmet u srednjoj školi (12. razred) u programskom jeziku Pascal opisuju (Ben-David Kolikant, 2003). U provedenom istraživanju cilj je bio ustanoviti razumijevanje sinkronizacije mjesec dana nakon završetka predmeta. Ustanovljeno je da učenici dobro razumiju većinu problema sa sinkronizacijom ali da veliki dio njih zaključuje prema predlošku. Isto tako ustanovljeno je da postoji problem u vezi između paralelnog programiranja i od prije naučenog gradiva. Poučavanje paralelnog programiranja na trodnevnom kampu za odabrane učenike opisuju (Chesebrough & Turner, 2010). Na kampu su se uz kombinaciju igranja uloga i programiranja u C/C++ i OpenMP obrađivali standardni paralelni problemi od kojih su neki: paralelna obrada elemenata liste, množenje matrica, računanje vrijednosti broja π i sl. Zanimljivo je da se učenicima više svidjelo programiranje u OpenMP nego igranje uloga, za koje je dio učenika smatrao da je djetinjasto. Slična iskustva poučavanja paralelnog programiranja za djecu uzrasta 9-10 godina bez programerskog iskustva opisuju (Gregg, Tychonievich, Cohoon, & Hazelwood, 2012). Osnovni ciljevi radionice bila su usvojiti osnove paralelnog programiranja na više procesora, razumijevanje pojma programiranje i računalo te učiti učenike paralelnom razmišljanju. Za razvoj paralelnih programa napravljen je alat EcoSim koji je blizak govornom jeziku. Iskustva poučavanja paralelnog programiranja u programskom jeziku Scratch za djevojčice viših razreda osnovne škole te srednje škole opisuju (Feldhausen, Bell, & Andresen, 2014). Preko 80% polaznica ovog kampa smatralo je da bi

bilo u stanju primijeniti znanja na neke druge paralelne probleme u Scratchu te da je razumjelo što je paralelno programiranje. Opis radionice u trajanju od 50 minuta za učenike srednjih škola, početnike u programiranju, čiji je cilj bio pokazati da učenici mogu razumjeti koncepte paralelnog programiranja opisuje (Rifkin, 1994).

(Broll, i dr., 2017) smatraju da uz dobro osmišljene nastavne materijale učenici u K-12 obrazovanju mogu svladati osnove paralelnog programiranja. Kao pomoć u tome autori su kreirali NetsBlox – program sličan Scratchu koji ima mogućnosti kreiranja paralelnih aplikacija. NetsBlox omogućava izradu igara gdje se paralelizam realizira s dva aspekta: dva lika istovremeno rade nešto ili jedan lik istovremeno radi više radnji (npr. kreće se i pri tome skuplja neke elemente na ploči).

Kao i kod većine koncepata u informatici i kod paralelnog programiranja su česte promjene: procesori se brzo razvijaju, mijenjaju se programska okruženja te je problem biti u trendu (Petit, Sahuquillo, Gmez, & Selfa, 2017). Stoga autori opisuju Advanced Multicore Architecture (AMA) metodologiju poučavanja, koja je bazirana na istraživanju te kritičkom pristupu problemu. Pri tome autori govore o 5 ključnih aktivnosti:

- teorijska predavanja,
- pregled radova i diskusija,
- istraživački orijentiran praktični ispit,
- vježbe na računalima s nekim višejezgrenim simulatorom,
- prezentacija istraživačkog rada.

(Manogaran, 2013) pokazuju da studenti koji su paralelno programi učili na interaktivniji način, što je uključivalo standardna predavanja, aktivnosti za aktivno učenje, grupne aktivnosti te rad na projektima postižu na ispitima bolje rezultate od studenata kojima je paralelno programiranje prezentirano na standardni način.

Igru koja bi trebala pridonijeti boljem razumijevanju koncepata paralelnog programiranja opisuju (Finlayson, Mueller, Rajapakse, & Easterling, 2015). Osnova igre je ploča na kojoj roboti prenose pakete, pri čemu robota može biti više. Kroz ovu igru uče se osnovni koncepti paralelnog programiranja: proces, sinkronizacija, komunikacija, semafori, isključivanje, potpuni zastoj, nedeterminističko ponašanje i sl. Poučavanje nekih koncepata paralelnog programiranja kroz igru opisuju i (Hunt & Willison, 2011). Da je igra važna u edukaciji smatraju i (Wein, i dr., 2009). Neki od argumenata koje autori za to navode su: razvoj strategija, analitičkog razmišljanja, rješavanje problema, donošenje odluka, planiranje i sl.; igre se mogu koristiti za vježbanje praktičnih vještina koje se rijetko koriste; današnji studenti vole se igrati. Cilj igre je studentima dati iskustvo rada sa složenim distribuiranim sustavima te upoznati studente sa skalabilnošću kako bi sustav maksimalno iskoristili. Igra se temelji se na generiranju velike količinu prometa prema društvenim mrežama kojom trebaju upravljati igrači. (Kitchen, Schaller, & Tymann, 1992) opisuju različite igre za usvajanje određenih koncepata paralelnog programiranja, a koje su pogodne kao aktivnosti na satu. Neke od igara su:

- SIMD – bacanje novčića: ako padne glava student digne ruku, te na taj način nastavnik može lako prebrojati koliko je palo glava. Postoji i verzija u kojoj student više puta baca novčić te o tome izvješćuje nastavnika,
- MIMD (MPI) – svaki student ima komad papira (memorija) na kojem pišu vrijednosti lokalnih varijabli te međusobno šalju papire. Cilj je uvidjeti da neki student ne može nastaviti posao dok nije dobio odgovarajući, traženi papir.

(Tikvati, Ben-Ari, & Kolikant, 2004) smatraju da je problem Bizantskih generala koristan za poučavanje paralelnog programiranja. Nekoliko analogija koje su korisne za poučavanje paralelnog programiranja opisuju (Neeman, Lee, Mullen, & Newman, 2006), a jedna od njih je slaganje Jigsaw puzli. Na ovom primjeru može se ilustrirati razlika između zajedničke memorije i distribuiranog paralelizma.

Uvođenje paralelnog programiranja kroz niz primjera igranja uloga opisuje i (Maxim, Bachelis, James, & Stout, 1990). Učenje osnovnih koncepata paralelnog programiranja putem igre OpenTTP, gdje je cilj sagraditi infrastrukturu za prijevoz dobara od proizvođača do konzumenata opisuju (Marmorstein, 2015).

Zanimljiv i aktualan pristup paralelnom programiranju kroz primjer izgradnje paralelnih aplikacija za Android platformu opisuju (Matos & Grasser, 2014).

Roboti imaju više senzora koji istovremeno može prikupljati podatke, koji se paralelno s tim obrađuju te kao rezultat te obrade slijede odgovarajuće aktivnosti. Možemo razlikovati nekoliko osnovnih tipova naredbi:

- sinkrone naredbe – odmah su gotove
- asinkrone naredbe – započinju s izvođenjem odmah, a vrijeme izvođenja nije unaprijed poznato,
- asinkroni događaji – događaju se neovisno o kontroli tijeka izvođenja programa, tzv. prekidi (*eng. interupts*).

Ovo je još jedna od mogućnosti kako uvesti paralelno programiranje (Bordignon, Mikkelsen, & Schultz, 2008). (Jacobsen & Jadud, 2005) također govore o uvođenju paralelnog programiranja korištenjem robota te smatraju da na taj način studenti dublje razumiju paralelno programiranje, a i zanimljivije je.

(Lupo & Wood, 2012) opisuju iskustva spajanja studenata s dvaju kolegija viših godina studija: Applied Parallel Computing i Advanced Rendering Techniques. Studenti su u grupama rješavali probleme renderiranja paralelnim algoritmima. Iskustva poučavanja paralelnog programiranja na način da se kreiraju heterogeni timovi studenata s različitih godina i kolegija opisuje (Cota de Freitas, 2013).

Kolegij s poslijediplomskom studija, gdje je cilj konfigurirati klaster računala te na toj arhitekturi poučavanje paralelnog programiranja opisuje (López & Baydal, 2017).

Većina autora bavi se paralelnim programiranjem, (Cunha & Lourenço, 1998) opisuju kolegij koji se temelji na distribuiranom programiranju.

(Paprzycki, 2006) smatra da je paralelno programiranje prirodno te je serijsko programiranje samo specijalan slučaj paralelnog programiranja. Stoga autor smatra da poučavanje programiranja treba započeti paralelnim programiranjem.

Iskustva poučavanju paralelnog programiranja studenata koji nisu informatičari (matematičari, fizičari, kemičari i sl.) opisuju (Cesar, i dr., 2017).

Online tečaj za učenje paralelnog programiranja opisuju (Mullen, i dr., 2017). Kao osnovne prednosti ovakvog načina učenja autori navode: moguće je učiti bilo kada i bilo gdje, besplatno ili uz malu nadoknadu, na istom mjestu sadržaj i zadaci, učenje modula umjesto lekcija, automatsko testiranje programskih rješenja, forumi i dr. Online tečaj na Courseri opisuju (Sarkar, Grossman, Budimlic, & Imam, 2017). Kolegij je podijeljen u 3 modula: paralelizam, konkurentnost i distribuirano programiranje, a programski jezik je Java. Materijale za poučavanje paralelnog programiranja te alat za testiranje paralelnih programa - ALPACA opisuje (Sadowski, i dr., 2011).

Poučavanje studenata paralelnom programiranju kroz niz predložaka (*eng. patterns*) opisuje (Adams J. C., Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns, 2015) te (Adams, Brown, & Shoop, 2013). (Capel, Tomeu, & Salguero, 2017) zagovaraju učenje kod kojega studenti aktivno sudjeluju u nastavi. Nakon što im se na primjeru ilustrira neki koncept studenti su samostalno, istražujući rješavaju slične probleme. Sličan pristup paralelnom programiranju temeljen na problemskoj nastavi ali i kurikulum paralelnog programiranja zajedno s nastavnim materijalima opisuje (Tran, 2010). Iskustva poučavanja paralelnog programiranja konstruktivističkim pristupom s primjerom reda koji je ilustriran kupcima koji dolaze i odlaze te pripadne implementacije opisuju (Dolgopolovas, Dagienė, Minkevičius, & Sakalauskas, 2015).

Mentalni model pronalaska grešaka u paralelnom programu opisuje (Sadowski, Mental models and parallel program maintenance, 2011). Istraživanje je načinjeno na način da bi student koji je napisao program nacrtao sliku koja predstavlja način razmišljanja, koja je potom analizirana.

Tareador – alat koji pomaže studentu da serijski program pretvori u paralelni opisuju (Ayguadé, i dr., 2015). Nakon što student podijeli serijski program u zadatke alat mu daje između ostaloga sljedeće informacije:

- ovisnosti zadataka te dijelovi o kojima treba voditi računa u paralelnoj izvedbi,
- prijedlog paralelizacije,
- simulacija paralelnog izvođenja na idealnoj arhitekturi.

Veliki broj radova bavi se vizualizacijom paralelnog programiranja. Kao osnovne razloge vizualizacije algoritama općenito (Naps & Chan, 1999) navode konceptualno razumijevanje algoritma te pomoć u kodiranju algoritma. (Bedy, Carr, Huang, & Shene, 2000) govore o dva tipa vizualizacije: vizualizacija algoritma i vizualizacija izvođenja programa, a vizualizirati se može u realnom vremenu ili po završetku izvođenja programa, pri čemu se za vrijeme izvođenja programa prikupljaju podaci za vizualizaciju. (Stasko & Kraemer, 1993) govore o nekoliko područja na koja se može primijeniti vizualizacija kod paralelnog programiranja, a to su: dizajn programa, evaluacija performansi programa te otkrivanje grešaka u programu. Slično razmišljaju i (Miller B. P., 1993) koji smatraju da je vizualizacija korisna jedino kod ispravljanja grešaka u programu i kod poboljšanja performansi programa. Autori također daju smjernice za kreiranje vizualizacije, a neke od njih su: vizualizacija treba biti interaktivna, davati važne poruke, a osobitu važnost treba voditi kod odabira boja jer one povećavaju važnost informacija koje sustav daje, naglašavaju važne slučajeve i sl. (Kraemer & Stasko, 1993) navode četiri osnovna koraka kod vizualizacije paralelnog programa:

- prikupljanje podataka – podatke treba prikupljati na način da se postupak može lako isključiti, a sam proces može biti softverski (unutar programa koji se izvodi) ili hardverski, putem posebnom chipa,
- analiza podataka – računanje statistike, redoslijed događanja, prikupljanje podataka u sustavima za debugiranje,
- pohranjivanje podataka – najčešće posebna datoteka ili baza podataka,
- prikaz – zabilježeni podaci kronološki se čitaju, a vizualiziraju se najčešće samo segmenti izvođenja programa npr. pristup dijeljenoj memoriji, komunikacija među procesima ili redoslijed pozivanja potprograma. Vizualizirati se mogu i performanse programa, primjerice iskorištenost procesora ili opterećenje komunikacije.

(Lönnberg, Malmi, & Berglund, Helping students debug concurrent programs, 2008) smatraju da je veliki problem paralelnog programiranja u tome da studenti na izvođenje programa gledaju s aspekta korisnika, ne vide ograničenja u programu (problemi s memorijom, mrežom i sl.). Često studenti ne razumiju što točno njihov program radi i koje slučajeve pokriva te je stoga korisno vizualizirati izvođenje programa. Autori smatraju da bi sustav za vizualizaciju, za programe koji ne rade, trebao ukazati na situacije u kojima program neće ispravno raditi, ukazati na mjesto u programu gdje se je greška dogodila te ustanoviti što student nije dobro razumio. O razlici između onoga što bi student htio da njegov program radi i onoga što stvarno program radi govore i (Cai, Milne, & Turner, 1993) te također smatraju da pri rješavanju tog problema može pomoći vizualizacija. Hardversko prikupljanje podataka putem TKM (Test Machine Kernel) chipa opisuju (Zimmermann, Perrenoud, & Schiper, 1988). Prednost ovakvog prikupljanja podataka jest činjenica da za prikupljanje podataka nema nikakvih intervencija u programskom kôdu.

(Exton & Kölling, 2000) opisuju sustav za vizualizaciju objektnog i usmjerenog programiranja te smatraju da vizualizacije s interakcijom mogu poboljšati razumijevanje gradiva. (Carr, Fang, Jozwowski, Mayo, & Shene, ConcurrentMentor: A visualization system for distributed programming education, 2003) opisuju sustav za vizualizaciju ConcurrentMentor kod kojega za prikupljanje podataka za vizualizaciju nije potrebno raditi intervencije u programskom kodu, a vizualizacija se odvija istovremeno s izvođenjem programa. (Robbins S. , 2003) opisuju poslužiteljsku verziju sustava za vizualizaciju. Alat ima dva dijela:

- dio koji prikuplja podatke o izvođenju programa od svih threadova te ih pohranjuje na poslužitelju,

- dio namijenjen vizualizaciji.

(Trümper, Bohnet, & Döllner, 2010) opisuju još jedan sustav za vizualizaciju te smatraju da je ona izuzetno važna kod ispravljanja grešaka u programu. Programeri u prosjeku 85-90% vremena troše na ispravljanje grešaka u programu, pri čemu je ispravljanje grešaka kod paralelnog programiranja još zahtjevnije jer ne postoje standardni debugeri kao kod sekvencijalnog programiranja. Kao najveći problem kod otkrivanju grešaka u paralelnom programu autori navode: otkrivanje potpunog zastoja, zajedničke memorije te istovremenog pristupa (*eng. race condition*).

Niz primjera kako vlastitim kodom vizualizirati paralelno izvođenje programa opisuju (Adams, i dr., 2016). Radi se o grafičkoj C++ biblioteci (TSGL) koja podržava više threadnu grafiku. Dobro osmišljenim programima u ovakvoj grafici dobro se može vizualizirati paralelno izvođenje programa. Neki od primjera na kojima su to autori načinili su: računanje površine ispod krivulje trapeznom formulom – svaki proces računa površinu jednog dijela te je svaki dio obojen drugačijom bojom; invertiranje slike – svaka komponenta boje svakog piksela zamjenjuje se s *255 – trenutna_komponenta* i sl. Na Provedenom istraživanju, grupa koja je učila paralelno programiranje ovim načinom pokazivala je značajnije bolje rezultate na programskim zadacima od grupe koja je paralelno programiranje radila standardnim načinom (zbrajanje, rad s matricama i sl.) te autori zaključuju da će problemi kod kojih studenti "vide" paralelizam značajno utjecati na razumijevanje paralelnog programiranja.

Evaluator – sustav za automatsko testiranje paralelnih rješenja (MDAT) opisuju (Larson & Palting, 2013). Kao veliki izazov evaluatora za paralelne programe autori navode problem testiranja istovremenog pristupa (*eng. race condition*) te potpunog zastoja i to iz razloga stohastičkog ponašanja programa. Stoga MDAT radi tako da pri izvođenju programa u nekom trenutku se izvodi samo jedan thread te se na taj način onemogućava da operacijski sustav sam radi raspored izvođenja threadova. Još jedan evaluator opisuju (Hundt, Schlarb, & Schmidt, 2017). Ovaj evaluator osim što validira programe ima i mogućnost provjere sadrži li program zadanu implementaciju rješenja te je moguće tražiti plagijate među rješenjima. Moguće je testirati različite modele paralelizacije: C++ threadove, OpenPM, MPI i CUDA.

ConEE simulator – alat koji analizira paralelni programski kôd te traži potpune zastoje i istovremene pristupe opisuju (Offenwanger & Lucet, 2014). Cilj je bio da se kroz vizualizaciju studenti intenzivnije angažiraju oko pisanja programa te eksperimentiranja, dok bi dio koji služi automatskom testiranju programskih rješenja kod studenata trebao razvijati samopouzdanje pisanja vlastitih programa. Još jedan sustav za testiranje paralelnih rješenja za razne paralelne modele – RAI opisuju (Dakkak, Pearson, Li, & Hwu, 2017). Alat ima niz opcija za konfiguriranje okruženja za izvođenje programa, a izuzetno je koristan za one studente koji ne raspolažu paralelnim arhitekturama. Sličan sustav – OnRamp opisuju i (Foley, Koepke, Ragatz, Brehm, & Regina, 2017). PAT (Parallel Analysis Tool) je sustav razvijen u Javi čiji je cilj studentima pomoći pri pretvaranju serijskog programa u paralelni (Rague B. W., 2012). Program identificira mjesta u programu koja je moguće paralelizirati te prikazuje benefite paraleliziranja programa.

U istraživanju gdje je studentima na ispitu iz paralelnog programiranja dan sustav za vizualizaciju rješenja Atropos te paralelni program koji ima grešku (Lönnberg, Malmi, & Ben-Ari, Evaluating a visualisation of the execution of a concurrent program, 2011) uočeno je da studenti iako im je rečeno da koriste, vizualizaciju često uopće nisu koristili. U radu (Lönnberg, Ben-Ari, & Malmi, Java replay for dependence-based debugging, 2011) smatraju da bi se korisnost alata mogla povećati ako bi studente eksplicitno poučavalo ispravljanje grešaka korištenjem ovog alata. Slično istraživanje, ovaj puta na alatu JThreadSpy opisuju i (Malnati, Maria Cuva, & Barberis, 2008). Autori su podijelili studente u dvije grupe od kojih je jedna grupa rješavala probleme koristeći JThreadSpy dok je druga grupa rješavala probleme bez ovog alata. Autori zaključuju da su studenti koji su koristili alat bolje detektirali problem potpunog zastoja i istovremenog pristupa. Zahvaljujući grafičkom prikazu studenti su lakše uočili odnose među threadovima. Isto tako studenti koji su mogli koristiti JThreadSpy brže su implementirali paralelno rješenje danog problema.

Neki od standardnih primjera za poučavanje paralelnog programiranja su:

- pretraživanja (Torbert, Vishkin, Tzur, & Ellison, 2010), (Chesebrough & Turner, 2010), (Ivanov, 2012),
- generiranje fraktala (Torbert, Vishkin, Tzur, & Ellison, 2010), (Chesebrough & Turner, 2010), (Ivanov, 2012),
- stanični automati (Gross, 2011), (Torbert, Vishkin, Tzur, & Ellison, 2010), (Ko, Burgstaller, & Scholz, 2013), (Ivanov, 2012),
- matrične operacije (Chesebrough & Turner, 2010), (Gross, 2011), (Hyde, 1989), (Kotz, 1995), (Ko, Burgstaller, & Scholz, 2013), (Johnson, Kotz, & Makedon, 2009),
- računanje vrijednosti broja π (Chesebrough & Turner, 2010), (Ko, Burgstaller, & Scholz, 2013), (Ivanov, 2012),
- sortiranja (Gross, 2011), (Hyde, 1989), (Kotz, 1995), (Moore, 2000), (Rifkin, 1994),
- probijanje lozinke (Ziwisky, Persohn, & Brylow, 2013), (Morris & Frinkle, 2014),
- zbrajanje elemenata liste (Jackson, 1991), (Feldhausen, Bell, & Andresen, 2014),
- prosti brojevi (Bunde D. P., A short unit to introduce multi-threaded programming, 2009), (Morris & Frinkle, 2014), (Moore, 2000),
- obrada slike (Asanovic, i dr., 2009), (Ko, Burgstaller, & Scholz, 2013),
- klijent poslužitelj aplikacije (Goldwasser & Letscher, Introducing network programming into a CS1 course, 2007).

V. ISKUSTVA POUČAVANJA PARALELNOG PROGRAMIRANJA U V. GIMNAZIJI U ZAGREBU

Mnoštvo autora, a među kojima su i: (Matos & Grasser, 2014), (Lewandowski, i dr., 2010), (Bruce, Danyluk, & Murtagh, 2010) i dr., dok (Kotz, 1995) smatra da paralelno

programiranje treba uvesti u kurikulum što ranije, prije nego se kod učenika usade navike serijskog programiranja. Slično razmišlja i (Niño, 2011). (Clements, 1999) smatra da je prelazak sa serijskog na paralelno programiranje težak te smatra da bi paralelno programiranje trebalo staviti što ranije u kurikulum. (Ben-Ari, Tool Presentation: Teaching Concurrency and Model Checking, 2009) smatra da bi se paralelno programiranje moglo početi poučavati puno ranije nego što se to radi, već u srednjoj školi. (Rifkin, 1994) smatra da bi učenike u srednjim školama trebalo istovremeno podučavati serijskom i paralelnom programiranju. Kao argument tome navodi da će podučavanje serijskog programiranja u velikoj mjeri priječiti put paralelnom razmišljanju te će se kod problema uglavnom tražiti serijska rješenja. (Broll, i dr., 2017) kaže da se uz dobro osmišljene primjere paralelno programiranje može poučavati prije fakultetskog obrazovanja.

Iskustva poučavanja paralelnog programiranja u srednjoj škola su rijetka, no opisuje ih nekoliko autora: (Torbert, Vishkin, Tzur, & Ellison, 2010), (Ben-David Kolikant, 2003). Iskustva poučavanja paralelnog programiranja učenika osnovnih i srednjih škola na informatičkim kampovima i radionicama opisuju: (Chesebrough & Turner, 2010), (Gregg, Tychonievich, Cohoon, & Hazelwood, 2012), (Feldhausen, Bell, & Andresen, 2014), (Feldhausen, Bell, & Andresen, 2014).

V.I. Nastava programiranja u prirodoslovno-matematičkim gimnazijama

V. gimnazija u Zagrebu radi prema planu prirodoslovno-matematičke gimnazije gdje učenici ovisno o smjeru imaju dva ili tri sata nastave informatike tjedno. Najveći dio nastave informatike, osobito viših razreda otpada na programiranje. Neki od nastavnih sadržaja programiranja koji su u prirodoslovno-matematičkim gimnazijama, do donošenja novog kurikuluma bili aktualni (Nastavni plan za gimnazije, 1994) su:

- naredbe za unos i ispis podataka,
- naredbe grananja,
- naredbe ponavljanja (petlje),
- potprogrami,
- složeni tipovi podataka (niz, string, skup, tekstualne datoteke),
- baze podataka,
- rekurzivni potprogrami,
- dinamičke strukture podataka,
- nestandardni tipovi podataka (stog, red i stablo),
- grafika,
- složenost algoritama.

Budući da je nastavni plan zadnjih desetak godina zastario, a zamijenjen je tek s (Predmetni kurikulumi - Informatika, 2017) nastavnici su imali svojevrsnu slobodu promjene nastavnog plana. Tako se u V. gimnaziji posljednjih 10tak godina između ostaloga podučava: objektno-usmjereno programiranje, programi s grafičkim korisničkim sustavima, kombinatorički algoritmi (kombinacije, permutacije, particije skupa itd.), algoritmi na grafovima, kriptografija te je eksperimentirano s paralelnim programiranjem, što je tema teksta u nastavku. Paralelno programiranje stavljeno je kao jedna od posljednjih tema u četvrtom razredu, kada su učenici daleko manje motivirani za rad.

V.II. Koncepti, primjeri i nastavne metode

(Niño, 2011) navodi osnovne odluke koje treba donijeti prije poučavanja paralelnog programiranja:

- odabir programskog modela,
- programski jezik,
- alati za testiranje i ispravljanje pogrešaka u programu,
- uvodni problemi i algoritmi,
- način na koji će se uvesti paralelno programiranje.

Budući da su učenici cijelo vrijeme za programiranje koristili programski jezik Python te obzirom na (John, NSF supported projects: parallel computation as an integrated component in the undergraduate curriculum in computer science, 1994) koji smatra kako je kod poučavanja paralelnog programiranja važno voditi računa da učenici nastave raditi u istom ili sličnom okruženju, odabir programskog jezika bio je lak. Dakle, paralelno programiranje je obrađivano u programskom jeziku Python. U Pythonu je moguće koristiti različite modele za pristup paralelnom programiranjem, a neki od njih su MapReduce (Adams J. C., Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns, 2015), CUDA (Tran, 2010), OpenMP i MPI (Brown, Adams, Bunde, Mache, & Shoop, A stratified view of programming language parallelism for undergraduate CS education, 2012), multithreading (Brown, Adams, Bunde, Mache, & Shoop, A stratified view of programming language parallelism for undergraduate CS education, 2012) i (Wolffe & Trefftz, 2009). U ovom slučaju je za poučavanje odabran model thredova, a neki od razloga za to su: dolazi uz standardnu instalaciju Pythona te ga nije potrebno dodatno instalirati, što u nekim situacijama može biti zahtjevno; vrlo mali broj jednostavnih naredbi; nije na visokoj niti preniskoj razini. Uz model thredova odabran je model paralelizacije podataka (svi thredovi rade isti posao samo na različitim skupovima podataka).

Kao i kod poučavanja bilo kojeg drugog nastavnog sadržaja i kod poučavanja paralelnog programiranja važni su uvodni (motivacijski) primjeri koji će učenike zainteresirati za temu. Ovdje je, za razliku od većine drugih istraživanja, odlučeno paralelno programiranje motivirati s dvije skupine primjera:

- nužnost threadova kod nekih, učenicima bliskim mrežnim aplikacijama,
- ubrzanje korištenjem thredova.

Iskustva korištenje threadova kod mrežnih aplikacijama u nastavi opisuju (Goldwasser & Letscher, Introducing network programming into a CS1 course, 2007). Radilo se je o nizu aplikacija kojima je cilj bio doći do nužnosti korištenja thredova: klijentska aplikacija za dohvaćanje točnog vremena s poslužitelja; izrada poslužitelja koji može prihvatiti klijenta, primiti od klijenta poruku te istu poruku vratiti natrag tzv. echo poslužitelj; chat poslužitelj koji treba istovremeno raditi s više klijenata; chat klijent koji istovremeno treba "čekati" poruke s poslužitelja i prikazivati ih na ekranu te "čekati" poruku koju klijent upisuje i šalje na poslužitelj.

S druge strane o važnosti ubrzanja kao motivacijskom konceptu pri poučavanju paralelnog programiranja govore mnogi autori. Tako (Joiner, Gray, Murphy, & Peck, 2006) smatraju da je kod paralelnog programiranja izuzetno važno pokazati ubrzanje i to ga pokazati u živo, na konkretnom primjeru. Isto tako je važno da primjeri koji će se obrađivati budu učenicima bliski. (Ernst D. J., 2011) smatra da su dva osnovna uvjeta da bi se učenike zainteresiralo za paralelno programiranje: ukazati na značajnu razlika u performansama, a tu razliku je potrebno ilustrirati na pravom programskom kodu. Slične stavove iznosi i (Moore, 2000) koji smatra da učenici vole vidjeti konkretne primjere i konkretno ubrzanje. (John, NSF supported projects: parallel computation as an integrated component in the undergraduate curriculum in computer science, 1994) smatra da bi kod poučavanja paralelnog programiranja učenicima trebalo ukazati na činjenice kao što su: paralelno programiranje može znatno ubrzati izvođenje programa; postoje različiti modeli paralelnog programiranja, a dva osnovna su oni s dijeljenom i distribuiranom memorijom; postoje prepreke koje značajno mogu usporiti izvođenje paralelnog programa (naglasak na sinkronizaciji i komunikaciji); potrebna je praksa na računalu.

Kada govorimo iz konteksta paralelnog programiranja u cilju ubrzanja programa (Brown, i dr., 2010) smatraju da su osnovna znanja koja bi učenici trebali usvojiti učenjem paralelnog programiranja su: raspoznati mogućnost paralelizacije u danom problemu te evaluirati primjenjivost različitih paralelnih strategija pri rješavanju problema. (Gopalakrishnan, i dr., 2009) nabrajaju teme koje bi poučavanje paralelnog programiranja trebalo obuhvaćati, a neke od njih su: diskusija o paralelnom programiranju, ilustracija ubrzanja, pregled nekih pristupa paralelnom programiranju (MPI, threadovi,…) te objašnjenje modela memorije. (Bunde D. P., A short unit to introduce multi-threaded programming, 2009) smatraju da su osnovni koncepti koje bi trebalo naučiti pri poučavanju paralelnog programiranja:

- fork/join,
- istovremeni pristup (*eng. race condition*),
- ubrzanje kod paralelnog rješenja (*eng. parallel speedup*),
- pravilna opterećenost (*eng. load balance*).

Nastavno na posljednje tri reference, a prevedeno u jezik ishoda, osnovni ishodi koje treba realizirati u okviru nastave paralelnog programiranja su:

- ustanoviti da paralelno programiranje može značajno ubrzati izvođenje programa,
- argumentirati zašto dolazi do ubrzanja programa primjenom paralelnog programiranja,
- objasniti mogućnost paralelizacije na konkretnom problemu,
- objasniti problem istovremenog pristupa, probleme do kojih zbog toga dolazi te opisati način rješavanja tog problema,
- implementirati paralelno rješenje zadanog problema korištenjem thredova.

a. Uvodni primjer – threadovi u mrežnim aplikacijama

Kao uvodni primjer za poučavanje paralelnog programiranja bio je izrada mrežnih aplikacijama s thredovima. Ovaj primjer odabran je iz razloga što je učenicima vrlo blizak. Odabrani sadržaji iz ove teme realizirani su kroz ukupno 6 nastavnih sati od čega su dva sata održana u običnoj učionici te dva puta po dva sata u informatičkoj učionici gdje su učenici radili na računalima. Kratki sadržaj nastavnih sati:

1. *sat (učionica)* – učenicima su objašnjeni osnovni pojmovi vezani uz računalnu mrežu: elementi računalne mreže, mrežni protokoli, IP adrese, DNS, slojevi mreže, portovi. Nakon toga dan je pregled Pythonovog modula *socket* koji sadrži funkcije i klase za rad s računalnom mrežom. Na kraju je implementirana klijent/poslužitelj aplikacija gdje poslužitelj čeka klijenta da se spoji, nakon spajanja klijenta na poslužitelj, klijent čeka da korisnik upiše neku poruku, upisana poruka se šalje poslužitelju, poslužitelj zaprimi poruku i sva slova poruke pretvori u velika slova te tako dobivenu poruku vraća klijentu. Primljenu poruku klijent ispisuje na ekranu. Na kraju su učenici kao domaću zadaću dobili da izmijene poslužiteljski dio aplikacije na način da poslužitelj može neprekidno prihvaćati korisnike te s njima ostvarivati opisanu komunikaciju. Učenici nisu imali većih problema s domaćom zadaćom.
2. *i 3. sat (vježbe za računalima)* – nakon kratkog ponavljanja učenici su dobili zadatak da se spoje na poslužitelj (koji je bio unaprijed pokrenut na računalu nastavnika) te nakon spajanja poslužitelju pošalju svoje ime te na kraju pročitaju poruku koju će im poslužitelj poslati i ispišu je na ekran. Nakon što je određeni broj učenika riješio zadatak oni su pomagali učenicima koji ga nisu uspjeli samostalno riješiti. Veći broj učenika u potpunosti je samostalno riješio zadatak. Nakon što su riješili ovaj zadatak, uslijedio je sličan zadatak: učenici su trebali kreirati aplikaciju s GUI gdje je u lijevom dijelu trebala biti lista s popisom, a u desnom dijelu naljepnica (*eng. label*). Aplikacija se je trebala spojiti na poslužitelj te od poslužitelja primiti popis naziva slika. Popis je bio sastavljen od naziva slika koji su bili međusobno odvajani znakom vertikalne crte (|) nazive slika bilo je potrebno upisati u postojeću listu s popisom (lijeva strana ekrana). Nakon što korisnik klikne na naziv odgovarajuće slike u listi s popisom poslužitelju treba poslati naziv slike te od poslužitelja primiti zapis slike kao niz bajtova te dobivene podatke prikazati kao sliku na naljepnici. Budući da se je radilo o zahtjevnijem problemu on je podijeljen na manje zadatke: kreiranje sučelja, dohvaćanje popisa slika s poslužitelja, slanje naziva slike poslužitelju i dohvaćanje slike, prikaz slike. Nakon detaljnije analize svakog dijela učenici su ga samostalno rješavali, a nakon nekog vremena bi neki od učenika rješenje napisao na računalu s projektorom te ga objasnio za učenike koji taj dio zadatak nisu uspješno riješili. Na ovaj način je brzina rješavanja zadatka bila prilagođena znanju i sposobnostima učenika. Učenici s boljim znanjem samostalno su rješavali zadatak bez da su sudjelovali u detaljnoj analizi s nastavnikom, oni slabijih znanja i sposobnosti su samostalno rješavali zadatak nakon detaljne analize koju su provodili s

nastavnikom, dok su učenici s najslabijim znanjima i sposobnostima rješenje zadatka prepisali s projektora. S većinom ovih manjih zadataka učenici su već bili upoznati, najčešće ne u istom ali u sličnom obliku. Izuzetak je bio preuzimanje slike s poslužitelja gdje su učenici mogli ili samostalno istraživati kako to načiniti ili pričekati detaljnu analizu i s ovim dijelom zadatka bilo je problema te ga niti jedan učenik nije samostalno riješio. Važno je napomenuti da za drugi primjer učenici nisu znali detalje implementacije poslužiteljskog dijela aplikacije.

4. *sat (učionica)* – ideja ovog sata bila je ukazati na problem kada poslužitelj treba istovremeno raditi s više klijenata, odgovarati na njihove zahtjeve te osim toga prihvaćati i nove klijente. U tu svrhu cilj je bio načiniti klijent/poslužitelj aplikaciju gdje će klijent poslužitelju slati brojeve, a poslužitelj će odgovarati porukama je li poslani broj prost ili složen. Budući da klijent tijekom jedne konekcije s poslužiteljem može slati više brojeva poslužitelj će trebati istovremeno održavati komunikaciju s više klijenata. Dakle, poslužitelj će istovremeno trebati čekati na više socketa poruke od klijenata (brojeve) te na njih odgovarati. Ovdje dolazi do problema: kako da poslužitelj istovremeno čeka poruke od svih klijenata i da čim je dobio poruku od nekog klijenta odgovara na nju. Situacija bi bila puno jednostavnija kada bi klijenti poruke slali pravilnim redoslijedom, no pravila ovdje nema. Dakle, ideja je da unutar programa imamo "radnike" od kojih svaki radi vrlo jednostavan posao: čeka poruku od klijenta te po zaprimanju odgovori na nju. Ovime dolazimo do pojma threada. Aplikacija je testirana na način da je klijentsku aplikaciju napravio još jedan učenik na pripremljenom prijenosnom računalu te je testiran rad poslužitelja s dvije instance klijentske aplikacije na prijenosnom računalu te jednom instancom na računalu na kojem je i poslužitelj. Poslužitelj je za svaki zahtjev ispisivao poruke na ekranu. Domaća zadaća bila je izmijeniti poslužiteljski dio aplikacije na način da se za prihvaćeni broj zbrajaju znamenke. Klijentski dio aplikacije trebalo je izmijeniti na način da program prekida rad kada se unese broj manji od 0. Zadatak+ je bio da se u slučaju slanja broja manjeg od 0 ugasi pripadni thread na poslužitelju.
5. *i 6. sat (vježbe za računalima)* – cilj ovog nastavnog sata bio je funkcionalna chat aplikacija i to klijentski i poslužiteljski dio aplikacije. Aplikacija je rađena u koracima.
    - poslužiteljska aplikacija koja će neprekidno prihvaćati klijente, od klijenta prihvatiti ime te svim do sada pristiglim klijentima slati poruku da se je chatu pridružila nova osoba s upisanim imenom,
    - kreiranje klijentskog dijela aplikacije za ovakvu poslužiteljsku aplikaciju pri čemu se poruke od poslužitelja ispisuju u tekstualnom prozoru Pythona,
    - doraditi klijentsku aplikaciju na način da joj se doda grafičko korisničko sučelje te da se poruke dobivene s poslužitelja ispisuju u području za tekst,
    - doraditi poslužiteljsku aplikaciju na način da za svakog klijenta može i čekati poruke od njega te za svaku zaprimljenu poruku od klijenta tu poruku proslijediti svim ostalim klijentima (uz dodatak imena klijenta koji je poslao poruku),
    - doraditi klijentski dio aplikacije na način da se grafičkom korisničkom sučelju doda i okvir za unos teksta u koji korisnik upisuje poruku te gumb Pošalji, kojim će se poruka poslati poslužitelju. Na ovaj način i klijent će trebati imati dva threada: primanje poruka s poslužitelja i prikaz u području za tekst te slanje poruka na poslužitelj.

Iako se možda ima osjećaj da ovaj dio nema veze s paralelnim programiranjem on se je pokazao kao dobar za razumijevanje threadova i potrebe za njima.

b. Threadovi kao "alat" za ubrzavanje programa

(Bunde D. P., A short unit to introduce multi-threaded programming, 2009) detaljno opisuje iskustva paralelnog programiranja na primjerima s prostim brojevima. Neki od osnovnih koraka su: sekvencijalno brojanje prostih brojeva do 2000000; paralelizacija s greškama (nema joina i locka); ubrzanje s neravnopravnom podjelom segmenata; pohranjivanje prostih brojeva, što se kasnije koristi za provjeru je li broj prost.

Sličan motivacijski primjer za ilustraciju ubrzanja te poučavanje osnovnih koncepata paralelnog programiranja načinjen je i u ovom slučaju. Prosti brojevi, iako učenicima ne jako zanimljivi, odabrani su iz razloga jer su učenici jako dobro upoznati s njima, s raznim algoritmima za njihovo generiranje, brojanje i sl. S druge strane oni mogu biti vremenski zahtjevni te će se na njima dobro moći vidjeti razlike u vremenima izvođenja programa. Nastava je i ovdje realizirana kroz ukupno 6 nastavnih sati i to kao i kod mrežnog programiranja: dva sata u učionici te dva puta po dva sata u obliku vježbi za računalima. Za razliku od mrežnog programiranja ovdje su umjesto threadova korišteni procesi (modul *multiprocessing*) i to prvenstveno iz razloga što je na taj način moguće dobiti bolja ubrzanja.

1. *sat (učionica)* – učenicima su objašnjeni još neki pojmovi vezani uz procese, razlika obzirom na threadove, funkcije za rad s procesima, pokretanje procesa, čekanje na završetak procesa i sl. Na istom satu obrađeni je nekoliko konkretnih primjera: pokrenuta su 4 procesa, svaki od njih je ispisao svoj ID i ime te je pričekao određeni (random) broj sekundi i na kraju ispisao koliko je ukupno trajao. Procesi su pokrenuti manualno jedan po jedan, a potom uporabom listi. Potom je načinjen primjer brojanja prostih brojeva do 1000000. Načinjena je funkcija koja vraća broj prostih brojeva u intervalu od *a* do *b*. Kreirana su dva oblika programa:
    - serijski program – poziva funkciju s parametrima 2 i 1000000,
    - paralelni program – kreiraju se dva procesa od kojih prvi broji proste brojeve od 2 do 500000, a drugi od 500001 do 1000000

    U oba slučaja izmjerena su vremena izvođenja programa te je ustanovljeno da se paralelni program izvodi brže ali to ubrzanje nije na razini 50% već nešto

više od 30%. Nadalje je analizirano zašto dolazi do ovakvog, a ne većeg ubrzanja te je ustanovljeno da bi jedan od razloga mogao biti u nepravednoj podjeli poslova. Naime, drugi proces ima daleko zahtjevniji posao. Brojevi za koje on provjerava jesu li prosti su daleko veći i time su vremenski zahtjevniji. Isto tako ustanovljeno je da nakon što smo kreirali paralelnu verziju programa na kraju ne znamo koliko je prostih brojeva od 2 do 1000000. Naime, kod procesa je nemoguće dobiti vrijednost koju funkcija, koja se izvodi u okviru procesa, vraća. Razgovorom smo došli do ideje da bi bilo dobro imati globalnu varijablu koja bi bila "zajednički brojač" za sve procese. Budući da se radi o procesima ta varijabla u Pythonu mora biti posebnog tipa: *Value* ili *Array*. Nakon implementacije ove varijable i pokretanja programa ustanovljeno je da paralelna verzija programa ne radi dobro: svaki puta se ispisuje drugačije rješenje i uvijek krivo. Ovim dolazimo do problema istovremenog pristupa te potrebe za "zaključavanjem varijabli" (*eng. lock*).

Domaća zadaća bila je izmjeriti izvođenje programa na računalu kod kuće za koji je trebalo ustanoviti koliko ima jezgri te testirati program na dva, četiri, osam i šesnaest procesa.

2. *i 3. sat (vježbe za računalima)* – tijekom ova dva sata učenici su rješavali niz manjih zadataka: zbrajanje elemenata liste; traženje najmanjeg elementa liste; brojanje Arsmtrongovih brojeva u zadanom intervalu; aproksimacija vrijednosti broja π Monte Carlo metodom, itd. Slično kao i prije učenici su rješavali probleme samostalno, uz analizu s nastavnikom te su učenici koji su uspješno riješili zadatke pomagali ostalim učenicima ili rješenja pisali na računalu spojenom na projektor. U svim primjerima mjerena su ubrzanja obzirom na serijske izvedbe istog programa.
4. *sat (učionica)* – sat je posvećen dodatnim ubrzanjima te je su u tom kontekstu rađene izmjene pri zaključavanju varijabli (samo jednom na kraju funkcije, umjesto svaki puta kada se pojavi prosti broj). Isto tako razmatrane su neke druge opcije: korištenje objekta Array ili paralelnog reda (Queue). Dodatno ubrzanje dobilo bi se ako bi podjela poslova bila pravednija. Jedan od načina na koji to napraviti jest kreiranje "puno" poslova te korištenje paralelnog objekta Pool. Svaki posao u ovom bi slučaju bio jedan kraći interval (primjerice 100 brojeva). Na taj način bi procesi dobivali prvo manje zahtjevne intervale, a potom sve zahtjevnije ali bi procesi bili pravednije zaposleni.
5. *i 6. sat (vježbe za računalima)* – cilj ova dva sata bio je nastavak rada na paralelizaciji nekih programa te analiza ubrzanja, a primjeri koji su obrađivani uglavnom su se odnosili na operacije s matricama.

Kao što se može primijetiti učenici su velikim dijelom bili aktivni sudionici nastave, što je osobito dolazilo do izražaja kod vježbi za računalima gdje su uglavnom samostalno rješavali zadatke. Ovo se podudara s mišljenjem (Marowka, 2008) i (Fisher, 1991), koji smatraju da je za poučavanje paralelnog programiranja izuzetno važno praktično iskustvo, odnosno da učenici sami programiraju, odnosno da se zajednički programira kako bi učenici aktivno sudjelovali u nastavi te promišljali o problemima i pokušavali dolaziti do zaključaka, a pri tome su koristili prijašnje iskustvo to novo naučeno gradivo. Na ovaj način njeguje se i konstruktivistički pristup učenju paralelnog programiranja kojega zagovaraju (Dolgopolovas, Dagienė, Minkevičius, & Sakalauskas, 2015).

VI. Provedeno istraživanje

Nakon kraju obrazovanja, posljednji sat 4. razreda učenici su ispunjavali anketu. Cilj ankete je bio ustanoviti stav učenika o poziciji paralelnog programiranja u okviru cjelokupnog obrazovanja i to s aspekta procjene: zanimljivosti, težine te korisnosti za budući profesionalni razvoj. Stavovi učenika bilježeni su Likertovom ljestvicom s 5 mogućnosti. Istraživanje je provedeno nad ukupno 122 učenika 4. razreda V. gimnazije i to u školskoj godini 2013./14. 28 učenika, u školskoj godini 2014./15. 22 učenika, u školskoj godini 2015./16. 47 učenika te u školskoj godini 2016./17. 25 učenika. Promatrana nastavna gradiva su:

- *strukturne naredbe* – naredba grananja (*if*) i petlje (*for* i *while*),
- *složeni tipovi podataka* – obuhvaća gradivo lista, stringova, skupova, rječnika te datoteka,
- *kornjačina grafika* – izrada jednostavnih crteža i animacija koristeći naredbe modula *turtle*,
- *objektno usmjereno programiranje* – osnovni pojmovi vezani uz objektno-usmjereno programiranje, niz jednostavnijih primjera te forsiranje korištenja u budućim zadacima,
- *apstraktne strukture podataka* – red, stog i binarno stablo, implementacija i korištenje pri rješavanju zadataka,
- *algoritmi nad grafovima* – definicija i načini zapisa grafa u računalu te standardni algoritmi nad grafovima: BFS, DFS, minimalno razapinjuće stablo (Prim i Kruskal), najkraća udaljenost vrha do svih ostalih vrhova (Dijkstra), maksimalna protočnost (Ford-Fulkerson),
- *kriptografija* – tradicionalni kriptografski algoritmi (Cezar, Vigenère, transpozicijsko, afino) i njihova implementacija te pregled modernih algoritama: DES i RSA,
- *programi s GUI* – izrada funkcionalnih programa sa standardnim elementima grafičkog korisničkog sučelja (modul *tkinter*),
- *paralelno programiranje* – opisano je u prethodnom poglavlju
- *baze podataka* – kreiranje strukture baze podataka te SQL naredbe za rad s podacima. Povezivanje baze podataka s Pythonom (modul *sqlite3*),
- *web programiranje* – osnove HTML-a, CSS, JavaScript (jQuery) te poslužiteljske skripte i Pythonov modul *django*.

Web programiranje imali su samo učenici generacija 2015./16. te 2016./17., dakle ukupno 72 učenika. Isto tako Algoritme nad grafovima nisu imali učenici generacije 2014./15., dakle ukupno Algoritme nad grafovima imalo je ukupno 100 učenika.

Školska godina	M	Ž	Ukupno
2013./14.	23	5	27
2014./15.	11	11	22
2015./16.	31	16	47
2016./17.	16	9	25

Tablica 1: Populacija prema školskim godinama i spolu

Tablični prikaz rezultata za kategoriju *Zanimljivost nastavnog gradiva* prikazano je u tablici 2 dok je grafički prikaz dan na slici 2.

	1	2	3	4	5
Strukturne naredbe	6	10	24	41	37
Složeni tipovi podataka	6	14	38	34	27
Kornjačina grafika	5	12	17	30	56
Objektno usmjereno programiranje	6	11	33	30	39
Apstraktne strukture podataka	8	25	30	36	19
Algoritmi nad grafovima	8	23	31	21	17
Kriptografija	3	11	26	28	51
Programi s GUI	3	9	15	27	67
Paralelno programiranje	5	21	29	30	32
Baze podataka	12	11	28	35	32
Web programiranje	7	9	9	21	26

Tablica 2: Procjena zanimljivosti (1- uopće nije zanimljivo - 5 - vrlo zanimljivo)

Primjenom Kruskal-Wallisovog testa za tri i više grupa u programu R dobiveni su rezultati: *Kruskal-Wallis chi-squared = 3.3326, df = 10, p-value = 0.9725*. Budući da je dobivena vrijednost manja od očekivane χ^2 vrijednosti na razini značajnosti 95% $P(x < 18.30704) = 0.95$ za 10 stupnjeva slobode te da je *p* vrijednost veća od 0.05 ustanovljeno je da nema statistički značajne razlike među grupama. Statistički značajna razlika u procjeni zanimljivosti nije uočena niti kada podatke promatramo obzirom na spol: *Muški - Kruskal-Wallis chi-squared = 2.5373, df = 10, p-value = 0.9903*; Ženski - *Kruskal-Wallis chi-squared = 4.6166, df = 10, p-value = 0.9153*. Razlika nije ustanovljena niti u slučaju kada gledamo odabir fakulteta. Za učenike koji su kao odabir fakulteta upisali matematika, fizika, FER ili FSB rezultati su: *Kruskal-Wallis chi-squared = 2.467, df = 10, p-value = 0.9913*, dok su rezultati za ostale fakultete: *Kruskal-Wallis chi-squared = 5.9907, df = 10, p-value = 0.816*.

	1	2	3	4	5
Strukturne naredbe	10	11	14	33	50
Složeni tipovi podataka	13	10	19	32	45
Kornjačina grafika	30	39	33	11	6
Objektno usmjereno programiranje	15	14	19	30	40
Apstraktne strukture podataka	19	20	34	23	23
Algoritmi nad grafovima	19	19	19	22	21
Kriptografija	21	20	30	23	26
Programi s GUI	14	17	20	29	39
Paralelno programiranje	17	14	18	24	44
Baze podataka	11	15	9	34	49
Web programiranje	9	5	15	15	29

Tablica 3: Procjena korisnosti (1 - uopće nije korisno- 5 - vrlo je korisno)

Kada govorimo o procjeni korisnosti nastavnog gradiva tablični prikaz rezultata dan je tablicom 3, dok je grafički prikaz dan je na slici 3. Primjenom Kruskal-Wallisovog testa u programu R dobiveni su rezultati: *Kruskal-Wallis chi-squared = 5.2287, df = 10, p-value = 0.8754*. Kao i u slučaju zanimljivosti uočavamo da ne postoji statistički značajna razlika u procjeni korisnosti nastavnog gradiva.

Gledamo li procjenu korisnosti prema spolu, rezultati su: Muški: *Kruskal-Wallis chi-squared = 4.3039, df = 10, p-value = 0.9326*; Ženski: *Kruskal-Wallis chi-squared = 7.2208, df = 10, p-value = 0.7044*. Obzirom na odabir fakulteta: tehnički (matematika, fizika, FER, FSB): *Kruskal-Wallis chi-squared = 2.5375, df = 10, p-value = 0.9903* te ostali fakulteti: *Kruskal-Wallis chi-squared = 4.8862, df = 10, p-value = 0.8986*. Dakle, niti u jednom slučaju ne postoji statistički značajna razlika.

Prikaz procjene težine nastavnog gradiva dan je tablicom 4 i slikom 4, a rezultati Kruskal-Walisovog testa na podacima su: *Kruskal-Wallis chi-squared = 3.479, df = 10, p-value = 0.9678*. Kao i u ostalim kategorijama nema statistički značajne razlike u procjeni težine nastavnih gradiva.

	1	2	3	4	5
Strukturne naredbe	65	34	12	6	2
Složeni tipovi podataka	51	29	24	10	5
Kornjačina grafika	84	23	5	5	4
Objektno usmjereno programiranje	30	33	34	12	9
Apstraktne strukture podataka	29	37	32	17	4
Algoritmi nad grafovima	16	27	34	18	5
Kriptografija	45	36	23	11	3
Programi s GUI	38	34	25	17	5
Paralelno programiranje	15	32	27	34	10
Baze podataka	40	26	25	19	9
Web programiranje	9	11	19	15	18

Tablica 4: Procjena težine nastavnog gradiva (1 - vrlo jednostavno - 5 - vrlo komplicirano)

Kruskal-Wallis test pokazuje da nema statistički značajne razlike niti među procjenama težine nastavnog gradiva: obzirom na spol: Muški - *Kruskal-Wallis chi-squared = 3.8608, df = 10, p-value = 0.9534*; Ženski: *Kruskal-Wallis chi-squared = 4.5999, df = 10, p-value = 0.9163*, a niti obzirom na vrstu fakulteta: tehnički fakulteti: *Kruskal-Wallis chi-squared = 4.3159, df = 10, p-value = 0.932*, dok su ostali fakulteti: *Kruskal-Wallis chi-squared = 5.5794, df = 10, p-value = 0.8493*.

Odgovori na pitanje koje nastavno gradivo bi izbacili, a koje je postavljeno studentima u školskim godinama 2013./14., 2014./15. te 2015./16. (ukupno 97 učenika) dani su tablicom 5 (bilo je moguće odabrati 0, 1 ili više nastavnih gradiva koje se želi izbaciti).

Nastavno gradivo	Broj učenika koji smatra da bi nastavno gradivo trebalo izbaciti
Algoritmi nad grafovima	22
Apstraktne strukture podataka	20
Kriptografija	20
Kornjačina grafika	18
Objektno usmjereno programiranje	14
Paralelno programiranje	13
Web programiranje	13
Baze podataka	11
Složeni tipovi podataka	6
Programi s GUI	5
Strukturne naredbe	3

Tablica 5: Nastavno gradivo i broj učenika koji smatra da bi to nastavno gradivo trebalo izbaciti

U školskoj godini 2016./17. učenicima je zadano da poredaju gradiva prema zanimljivosti (1 – najmanje zanimljivo,..., 11 – najzanimljivije). Mediani rezultata odgovora na ovo pitanje dani su u tablici 6:

Nastavno gradivo	Median redanja prema zanimljivosti
Kornjačina grafika	3
Algoritmi nad grafovima	4
Apstraktne strukture podataka	5
Objektno usmjereno programiranje	5
Strukturne naredbe	5
Web programiranje	6
Složeni tipovi podataka	6
Paralelno programiranje	7
Baze podataka	7
Kriptografija	8
Programi s GUI	8

Tablica 6: Mediani zanimljivosti nastavnih gradiva nakon redanja

VII. Zaključak

Paralelno programiranje danas je postalo standard. Tomu svakako pridonosi i činjenica da su sve češći uređaji koji imaju integrirano dva ili više procesora, odnosno dvije ili više procesorskih jezgri. Veliki broj sveučilišta uočio je važnost paralelnog programiranja te ga uvrstio u postojeće kolegije na nižim godinama studija ili su kreirani posebni kolegiji na nižim godinama studija za učenje paralelnog programiranja. Nekoliko je pokušaja poučavanja paralelnog programiranja na raznim kampovima za učenike te u srednjim školama. Postavlja se pitanje mogu li učenici prije fakultetskog obrazovanja uistinu razumjeti paralelno programiranje?

U opisanom istraživanju provedenom nad učenicima prirodoslovno-matematičke gimnazije, gdje učenici imaju informatiku četiri godine po dva ili tri sata tjedno te veći dio nastavnog sadržaja zauzima programiranje uočeno je da se paralelno programiranje, prema mišljenju učenika, ni po čemu ne razlikuje od ostalih nastavnih gradiva. Istraživanje je provedeno na 122 učenika 4. razreda u školskim godinama od 2013./14. do 2016./17., a usredotočeno je na 3 aspekta:

- procjena zanimljivosti nastavnog gradiva,
- procjena korisnosti nastavnog gradiva za budući profesionalni razvoj,
- procjena težine nastavnog gradiva.

Promatrano je ukupno 11 različitih nastavnih gradiva: strukturne naredbe, složeni tipovi podataka, kornjačina grafika, objektno usmjereno programiranje, apstraktne strukture podataka, algoritmi nad grafovima, kriptografija, programi s GUI, paralelno programiranje, baze podataka te web programiranje. Primjenom neparametarskog Kruskal-Wallis testa za tri ili više grupa ustanovljeno je da niti u jednoj od navedenih kategorija ne postoji statistički značajna razlika među nastavnim gradivima. Na pitanje koje nastavno gradivo bi izbacili paralelno programiranje našlo se je na 6. mjestu (od 11). Isto tako u generaciji 2016./17. postavljeno je pitanje da učenici nastavnim gradivima dodjele redne brojeve obzirom na zanimljivost, pri čemu 1 znači najmanje zanimljivo nastavno gradivo, dok 11 znači najzanimljivije nastavno gradivo. Paralelno programiranje ima median 7, a veći median imaju samo kriptografija i programi s GUI.

Na osnovu navedenih podataka slijedi da je uz opisani način poučavanja u prirodoslovno-matematičkim gimnazijama paralelno programiranje moguće poučavati.

VIII. Literatura

Adams, J. C. (2014). Injecting parallel computing into CS2. *SIGCSE '14 Proceedings of the 45th ACM technical symposium on Computer science education* (str. 277-282). Atlanta, Georgia, USA: ACM New York, NY, USA.

Adams, J. C. (2015). Patternlets A Teaching Tool for Introducing Students to Parallel Design Patterns. *IEEE International Parallel and Distributed Processing Symposium Workshops*, (str. 752-759).

Adams, J. C., Crain, P. A., Dilley, C. P., Nelesen, S. M., Unger, J. B., & Vander Stel, M. B. (2016). Seeing Is Believing: Helping Students Visualize Multithreaded Behavior. *SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (str. 473-478). Memphis, Tennessee, USA: ACM New York.

Adams, J., Brown, R., & Shoop, E. (2013). Patterns and Exemplars: Compelling Strategies for Teaching Parallel and Distributed Computing to CS Undergraduates. *IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum* (str. 1244-1251). IEEE Computer Society Washington, DC, USA.

Adl-Tabatabai, A.-R., Kozyrakis, C., & Saha, B. (December-January 2006-2007). Unlocking Concurrency. *Queue - Computer Architecture, 4*(10), 24-33.

Albrecht, J. R. (2009). Bringing big systems to small schools: distributed systems for undergraduates. *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education* (str. 101-105). Chattanooga, TN, USA: ACM New York.

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., . . . Yelick, K. (October 2009). A view of the parallel computing landscape. *Communications of the ACM - A View of Parallel Computing, 52*(10), 56-67.

Ayguadé, E., Badia, R. M., Jiménez, D., Herrero, J. R., Labarta, J., Subotic, V., & Utrera, G. (2015). Tareador: a tool to unveil parallelization strategies at undergraduate level. *WCAE '15 Proceedings of the Workshop on Computer Architecture Education* (str. Article No. 1). Portland, Oregon: ACM New York.

Bedy, M., Carr, S., Huang, X., & Shene, C.-K. (2000). A visualization system for multithreaded programming. *SIGCSE '00 Proceedings of the thirty-first SIGCSE technical symposium on Computer science education* (str. 1-5). Austin, Texas, USA: ACM New York.

Ben-Ari, M. (2001). Interactive execution of distributed algorithms. *Journal on Educational Resources in Computing (JERIC), 1*(2).

Ben-Ari, M. (2004). A suite of tools for teaching concurrency. *ITiCSE '04 Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education* (str. 251-251). Leeds, United Kingdom: ACM New York.

Ben-Ari, M. (2009). Tool Presentation: Teaching Concurrency and Model Checking. *Proceedings of the 16th International SPIN Workshop on Model Checking Software* (str. 6-11). Grenoble, France: Springer-Verlag Berlin, Heidelberg.

Ben-Ari, M., & Kolikant, Y. B.-D. (1999). Thinking parallel: the process of learning concurrency. *ITiCSE '99 Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education* (str. 13-16). Cracow, Poland: ACM New York.

Ben-Ari, M., & Silverman, S. (1999). DPLab: an environment for distributed programming. *ITiCSE '99 Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education* (str. 91-94). Cracow, Poland: ACM New York.

Ben-David Kolikant, Y. (2003). Learning concurrency: evolution of students' understanding of synchronization. *International Journal of Human-Computer Studies*, 243-268.

Berk, T. S. (1996). A simple student environment for lightweight process concurrent programming under SunOs. *SIGCSE '96 Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education* (str. 165-169). Philadelphia, Pennsylvania, USA: ACM New York.

Berry, F. C. (November 1995). An Undergraduate Parallel Processing Laboratory. *IEEE Transactions on Education, 38*(4), 306-311.

Bogaerts, S. A. (July 2017). One step at a time: Parallelism in an introductory programming course. *Journal of Parallel and Distributed Computing, 105*(C), 4-17.

Bohmann, J. A. (April 2002). An implementation of parallel processing technology for undergraduate research in computational chemistry. *Journal of Computing Sciences in Colleges, 17*(5), 32-36.

Bordignon, M., Mikkelsen, L. L., & Schultz, U. P. (2008). Implementing Flexible Parallelism for Modular Self-reconfigurable Robots. *SIMPAR '08 Proceedings of the 1st International Conference on Simulation, Modeling, and Programming for Autonomous Robots* (str. 123 - 134). Venice, Italy: Springer-Verlag Berlin, Heidelberg.

Broll, B., Ledeczi , A., Volgyesi, P., Sallai, J., Maroti, M., & Vanags, C. (2017). Introducing parallel and distributed computing to K12. *Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (str. 323-330). Lake Buena Vista, FL, USA: IEEE.

Brown, C. M., Lu, Y.-H., & Midkiff, S. (2013). Introducing Parallel Programming in Undergraduate Curriculum. *IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum* (str. 1269-1274). IEEE Computer Society Washington.

Brown, R. A., Adams, J. C., Bunde, D. P., Mache, J., & Shoop, E. (2012). A stratified view of programming language parallelism for undergraduate CS education. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 81-82). Raleigh, North Carolina, USA: ACM New York.

Brown, R. A., Adams, J. C., Bunde, D. P., Mache, J., & Shoop, E. (2013). Strategies for adding the emerging PDC curriculum recommendations into CS courses. *SIGCSE '13 Proceeding of the 44th ACM technical symposium on Computer science education* (str. 109-110). Denver, Colorado, USA: ACM New York.

Brown, R., & Shoop, E. (2011). Modules in community: injecting more parallelism into computer science curricula. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 447-452). Dallas, TX, USA: ACM New York.

Brown, R., Shoop, E., Adams, J., Clifton, C., Gardner, M., Haupt, M., & Hinsbeeck, P. (2010). Strategies for preparing computer science students for the multicore world. *ITiCSE-WGR '10 Proceedings of the 2010 ITiCSE working group reports* (str. 97-115). Ankara, Turkey: ACM New York.

Bruce, K. B., Danyluk, A., & Murtagh, T. (2010). Introducing concurrency in CS 1. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education* (str. 224-228). Milwaukee, Wisconsin, USA: ACM New York.

Bunde, D. P. (October 2009). A short unit to introduce multi-threaded programming. *Journal of Computing Sciences in Colleges, 25*(1), 9-20.

Bunde, D. P., Mache, J., & Drake, P. (October 2014). Adding parallel Haskell to the undergraduate programming language course. *Journal of Computing Sciences in Colleges*(30), 181-189.

Bunde, D., & Mache, J. (2009). Teaching concurrency beyond HPC. *Proceedings of the 2009 Workshop on Curricula in Concurrency and Parallelism.*

Bunde, D., Karavanic, K. L., Mache, J., & Mitchell, C. T. (2013). Adding GPU Computing to Computer Organization Courses. *IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum* (str. 1275-1282). IEEE Computer Society Washington, DC.

Burns, A. P., & Davies, G. L. (1988). Pascal-FC: a language for teaching concurrent programming. *ACM SIGPLAN Notices* (str. 58-66). ACM New York.

Burtscher, M., Peng, W., Qasem, A., Shi, H., Tamir, D., & Thiry, H. (2015). A Module-based Approach to Adopting the 2013 ACM Curricular Recommendations on Parallel Computing. *SIGCSE '15 Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (str. 36-41). Kansas City, Missouri, USA : ACM New York, NY, USA.

Cai, W., Milne, W. J., & Turner, S. J. (June 1993). Graphical views of the behavior of parallel programs. *Journal of Parallel and Distributed Computing - Special issue on tools and methods for visualization of parallel systems and computations, 18*(2), 223-230.

Capel, M. I., Tomeu, A. J., & Salguero, A. G. (July 2017). Teaching concurrent and parallel programming by patterns: An interactive ICT approach. *Journal of Parallel and Distributed Computing, 105*(C), 42-52.

Carr, S., & Shene, C.-K. (2000). A portable class library for teaching multithreaded programming. *ITiCSE '00 Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education* (str. 124-127). Helsinki, Finland: ACM New York.

Carr, S., Chen, P., Jozwowski, T. R., Mayo, J., & Shene, C.-K. (2002). Channels, visualization, and topology editor. *ITiCSE '02 Proceedings of the 7th annual conference on Innovation and technology in computer science education* (str. 106-110). Aarhus, Denmark: ACM New York.

Carr, S., Fang, C., Jozwowski, T., Mayo, J., & Shene, C. K. (2003). ConcurrentMentor: A visualization system for distributed programming education. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, (str. 1676-1682). Las Vegas, NV, United States.

Carr, S., Mayo, J., & Shene, C.-K. (October 2001). Teaching multithreaded programming made easy. *Journal of Computing Sciences in Colleges, 17*(1), 162-163.

Carro, M., Herranz, Á., & Mariño, J. (January 2013). A model-driven approach to teaching concurrency. *ACM Transactions on Computing Education (TOCE), 13*(1), Article No. 5.

Ceraj, I., Riley, J. T., & Shubert, C. (2009). StarHPC — Teaching parallel programming within elastic compute cloud. *Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces* (str. 353-356). Dubrovnik, Croatia: IEEE.

Cesar, E., Cortés, A., Espinosa, A., Margalef, T., Carlos, J., Sikora, M. A., & Suppi, R. (July 2017). Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies. *Journal of Parallel and Distributed Computing, 105*, 116-126.

Chesebrough, R. A., & Turner, I. (2010). Parallel computing: at the interface of high school and industry. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education* (str. 280-284). Milwaukee, Wisconsin, USA: ACM New York.

Choi, S.-E., & Lewis, E. C. (2000). A study of common pitfalls in simple multi-threaded programs. *SIGCSE '00 Proceedings of the thirty-first SIGCSE technical symposium on Computer science education* (str. 325-329). Austin, Texas, USA: ACM New York.

Clements, D. H. (1999). The Future of Educational Computing Research:The Case of Computer Programming. *Information Technology in Childhood Education Annual*, 147-179.

Computer processor history. (10. 2 2019). Dohvaćeno iz Computer Hope: https://www.computerhope.com/history/processor.htm

Cota de Freitas, H. (2013). Method for teaching parallelism on heterogeneous many-core processors using research projects. *Frontiers in Education Conference.* Oklahoma City, OK, USA: IEEE.

Cunha, J. E., & Lourenço, J. M. (1998). An integrated course on parallel and distributed processing. *SIGCSE '98 Proceedings of the twenty-*

ninth SIGCSE technical symposium on Computer science education (str. 217-221). Atlanta, Georgia, USA: ACM New York.

Dakkak, A., Pearson, C., Li, C., & Hwu, W.-m. (2017). RAI: A Scalable Project Submission System for Parallel Programming Courses. *Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (str. 315-322). Lake Buena Vista, FL, USA: IEEE.

Davies, G. L. (1990). Teaching concurrent programming with Pascal-FC. *ACM SIGCSE Bulletin* (str. 38-41). ACM New York.

Dolgopolovas, V., Dagienė, V., Minkevičius, S., & Sakalauskas, L. (January 2015). Teaching scientific computing: a model-centered approach to pipeline and parallel programming with C. *Scientific Programming*, Article No. 11.

Dukielska, M., & Sroka, J. (2010). JavaSpaces NetBeans: a linda workbench for distributed programming course. *ITiCSE '10 Proceedings of the fifteenth annual conference on Innovation and technology in computer science education* (str. 23-27). Bilkent, Ankara, Turkey: ACM New York.

Ernst, D. J. (2011). Preparing students for future architectures with an exploration of multi- and many-core performance. *ITiCSE '11 Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (str. 57-62). Darmstadt, Germany: ACM New York.

Ernst, D., Wittman, B., Harvey, B., Murphy, T., & Wrinn, M. (2009). Preparing students for ubiquitous parallelism. *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education* (str. 136-137). Chattanooga, TN, USA: ACM New York.

Ernst, J. D., & Stevenson, D. E. (2008). Concurrent CS: preparing students for a multicore world. *ITiCSE '08 Proceedings of the 13th annual conference on Innovation and technology in computer science education* (str. 230-234). Madrid, Spain: ACM New York.

Exton, C., & Kölling, M. (2000). Concurrency, objects and visualisation. *ACSE '00 Proceedings of the Australasian conference on Computing education* (str. 109-115). Melbourne, Australia: ACM New York.

Fekete, A. D. (March 2009). Teaching about threading: where and what? *ACM SIGACT News, 40*(1), str. 51-57.

Feldhausen, R., Bell, S., & Andresen, D. (2014). Minimum Time, Maximum Effect: Introducing Parallel Computing in CS0 and STEM Outreach Activities Using Scratch. *XSEDE '14 Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment* (str. Article No. 75). Atlanta, GA, USA: ACM New York.

Feldman, M. B., & Bachus, B. D. (1997). Concurrent programming CAN be introduced into the lower-level undergraduate curriculum. *ITiCSE '97 Proceedings of the 2nd conference on Integrating technology into computer science education* (str. 77-79). Uppsala, Sweden: ACM New York.

Finlayson, I., Mueller, J., Rajapakse, S., & Easterling, D. (2015). Introducing Tetra: An Educational Parallel Programming System. *IPDPSW '15 Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop* (str. 746-751). IEEE Computer Society Washington, DC, USA.

Fisher, A. L. (1991). Teaching the programming of parallel computers. *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education* (str. 102-107). San Antonio, Texas, USA: ACM New York.

Flynn, M. (1972). Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, 948 - 960.

Foley, S. S., Koepke, D., Ragatz, J., Brehm, C., & Regina, J. (July 2017). OnRamp: A web-portal for teaching parallel and distributed computing. *Journal of Parallel and Distributed Computing, 105*, 138-149.

Goldwasser, M., & Letscher, D. (2007). Introducing network programming into a CS1 course. *ITiCSE '07 Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, (str. 19-22). Dundee.

Gopalakrishnan, G., Yang, Y., Vakkalanka, S., Vo, A., Aananthakrishnan, S., Szubzda, G., . . . Atzeni, S. (2009). Some resources for teaching concurrency. *PADTAD '09 Proceedings of the 7th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging* (str. Article No. 2). Chicago, Illinois: ACM New York.

Graf, M., & Bunde, D. P. (October 2014). Using wrappers to simplify task parallel programming. *Journal of Computing Sciences in Colleges, 30*(1), 73-79.

Graham, J. R. (December 2007). Integrating parallel programming techniques into traditional computer science curricula. *ACM SIGCSE Bulletin, 39*(4), str. 75-78.

Gregg, C., Tychonievich, L., Cohoon, J., & Hazelwood, K. (2012). EcoSim: a language and experience teaching parallel programming in elementary school. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 51-56). Raleigh, North Carolina, USA: ACM New York.

Gross, T. R. (2011). Breadth in depth: a 1st year introduction to parallel programming. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 435-440). Dallas, TX, USA: ACM New York, NY, USA.

Grossman, D., & Anderson, R. E. (2012). Introducing parallelism and concurrency in the data structures course. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 505-510). Raleigh, North Carolina, USA: ACM New York.

Hennessy, J. L., & Patterson , D. A. (2011). *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann.

Herlihy, M., & Shavit, N. (2008). *The Art of Multiprocessor Programming.* Morgan Kaufmann.

Higginbotham, C. W., & Morelli, R. (1991). A system for teaching concurrent programming. *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education* (str. 309-316). San Antonio, Texas, USA: ACM New York.

Hochstein, L., Basili, V. R., Vishkin, U., & Gilbert, J. (November 2008). A pilot study to compare programming effort for two parallel programming models. *Journal of Systems and Software, 81*(11), 1920-1930.

Hundt, C., Schlarb, M., & Schmidt, B. (July 2017). SAUCE: A web application for interactive teaching and learning of parallel programming. *Journal of Parallel and Distributed Computing, 105*, 163-173.

Hunt, J. M., & Willison, T. (2011). California speedway: a concurrent programming project for beginners. *ACM-SE '11 Proceedings of the 49th Annual Southeast Regional Conference* (str. 7-12). Kennesaw, Georgia: ACM New York.

Hyde, D. G. (1989). A parallel processing course for undergraduates. *SIGCSE '89 Proceedings of the twentieth SIGCSE technical symposium on Computer science education* (str. 170-173). Louisville, Kentucky, USA: ACM New York.

Ivanov, L. (January 2012). The right balance: restructuring the parallel and scientific computing course. *Journal of Computing Sciences in Colleges, 27*(3), 115-121.

Jackson, D. (1991). A mini-course on concurrency. *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education* (str. 92-96). 1991: ACM New York.

Jacobsen, C. L., & Jadud, M. C. (2005). Towards concrete concurrency: occam-pi on the LEGO mindstorms. *SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education* (str. 431-435). St. Louis, Missouri, USA: ACM New York.

Jadud, M. C., Simpson, J., & Jacobsen, C. L. (2008). Patterns for programming in parallel, pedagogically. *SIGCSE '08 Proceedings of the 39th SIGCSE technical symposium on Computer science education* (str. 231-235). Portland, OR, USA: ACM New York.

Jaswanth, S., Herhut, S., Hudson, R. L., & Shpeisman, T. (2012). Teaching parallelism with river trail. *DCP '12 Proceedings of the 2012 workshop on Developing competency in parallelism: techniques for education and training* (str. 1-8). Tucson, Arizona, USA: ACM New York.

Jipping, M. J., Toppen, J. R., & Weeber, S. (1990). Concurrent distributed Pascal: a hands-on introduction to parallelism. *SIGCSE '90 Proceedings of the twenty-first SIGCSE technical symposium on Computer science education* (str. 94-99). Washington, D.C., USA: ACM New York.

John, D. J. (1994). NSF supported projects: parallel computation as an integrated component in the undergraduate curriculum in computer science. *SIGCSE '94 Proceedings of the twenty-fifth SIGCSE symposium on Computer science education* (str. 357-361). Phoenix, Arizona, USA: ACM New York, NY, USA.

John, D. J., & Thomas, S. J. (2014). Parallel and Distributed Computing across the Computer Science Curriculum. *IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)* (str. 1085-1090). Phoenix, AZ, USA: IEEE.

Johnson, D., Kotz, D., & Makedon, F. (2009). Teaching Parallel Computing to Freshmen. *Proceedings of the Conference on Parallel Computing for Undergraduates.*

Joiner, D. A., Gray, P., Murphy, T., & Peck, C. (2006). Teaching parallel computing to science faculty: best practices and common pitfalls. *PPoPP '06 Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming* (str. 239-246). New York, New York, USA: ACM New York, NY, USA.

K–12 Computer Science Framework. (2016).

Kitchen, A. T., Schaller, N. C., & Tymann, P. T. (September 1992). Game playing as a technique for teaching parallel computing concepts. *ACM SIGCSE Bulletin, 24*(3), str. 35-38.

Ko, Y., Burgstaller, B., & Scholz, B. (2013). Parallel from the beginning: the case for multicore programming in thecomputer science undergraduate curriculum. *SIGCSE '13 Proceeding of the 44th ACM technical symposium on Computer science education* (str. 415-420). Denver, Colorado, USA: ACM New York.

Kolikant, Y. B.-D., Ben-Ari, M., & Pollack, S. (2000). The anthropology semaphores. *ITiCSE '00 Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education* (str. 21-24). Helsinki, Finland: ACM New York.

Kotz, D. (1995). A data-parallel programming library for education (DAPPLE). *SIGCSE '95 Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education* (str. 76-81). Nashville, Tennessee, USA: ACM New York.

Kraemer, E., & Stasko, J. (June 1993). The visualization of parallel systems: an overview. *Journal of Parallel and Distributed Computing - Special issue on tools and methods for visualization of parallel systems and computations, 18*(2), 105-117.

Kumar, S. (July 2017). Research-oriented teaching of PDC topics in integration with other undergraduate courses at multiple levels: A multi-year report. *Journal of Parallel and Distributed Computing, 105*, 92-104.

Kurikulum nastavnog predmeta informatika za osnovne i srednje škole. (6. 3 2018).

Kurtz, B. L., Kim, C., & Alsabbagh, J. (1998). Parallel computing in the undergraduate curriculum. *SIGCSE '98 Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education* (str. 212-216). Atlanta, Georgia, USA: ACM New York.

Larson, E., & Palting, R. (2013). MDAT: a multithreading debugging and testing tool. *SIGCSE '13 Proceeding of the 44th ACM technical symposium on Computer science education* (str. 403-408). Denver, Colorado, USA: ACM New York.

Lewandowski, G., Bouvier, D. J., Chen, T.-Y., McCartney, R., Sanders, K., Simon, B., & VanDeGrift, T. (July 2010). Commonsense Understanding of Concurrency: Computing Students and Concert Tickets. *Communications of the ACM, 53*(7), 60-70.

Li, B., Mooring, J., Blanchard, S., Johri, A., Leko, M., & Cameron, K. W. (July 2017). SeeMore: A kinetic parallel computer sculpture for educating broad audiences on parallel computation. *Journal of Parallel and Distributed Computing, 105*(C), 183-199.

Lin, S., & Tatar, D. (2011). Encouraging parallel thinking through explicit coordination modeling. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 441-446). Dallas, TX, USA: ACM New York.

Liu, X. (December 2008). Teaching parallel computing in a small college: meeting a renewed demand. *Journal of Computing Sciences in Colleges, 24*(2), 179-188.

Lönnberg, J., Ben-Ari, M., & Malmi, L. (2011). Java replay for dependence-based debugging. *PADTAD '11 Proceedings of the Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging* (str. 15-25). Toronto, Ontario, Canada: ACM New York.

Lönnberg, J., Malmi, L., & Ben-Ari, M. (2011). Evaluating a visualisation of the execution of a concurrent program. *Koli Calling '11 Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (str. 39-48). Koli, Finland: ACM New York.

Lönnberg, J., Malmi, L., & Berglund, A. (2008). Helping students debug concurrent programs. *Koli '08 Proceedings of the 8th International Conference on Computing Education Research* (str. 76-79). Koli, Finland: ACM New York.

López, P., & Baydal, E. (July 2017). On a course on computer cluster configuration and administration. *Journal of Parallel and Distributed Computing, 105*, 127-137.

Lu, B., Conley, M., & Klein, A. (May 2014). Teaching true computer science principles to the general student. *Journal of Computing Sciences in Colleges, 29*(5), 233-239.

Lupo, C., & Wood, Z. J. (2012). Cross teaching parallelism and ray tracing: a project-based approach to teaching applied parallel computing. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 523-528). Raleigh, North Carolina, USA: ACM New York.

Mache, J., & Karavanic, K. L. (October 2012). Teaching parallelism with GPUS and a Game of life assignment. *Journal of Computing Sciences in Colleges, 28*(1), 200-202.

Malnati, G., Maria Cuva, C., & Barberis, C. (2008). JThreadSpy: A Tool for Improving the Effectiveness of Concurrent System Teaching and Learning. *CSSE '08 Proceedings of the 2008 International Conference on Computer Science and Software Engineering* (str. 549-552). IEEE Computer Society Washington, DC.

Manogaran, E. (2013). ACt-PBL: An Adaptive Approach to Teach Multi-core Computing in University Education. *IEEE Fifth International Conference on Technology for Education* (str. 19-23). Kharagpur, India: IEEE.

Marmorstein, R. (January 2015). Teaching semaphores using... semaphores. *Journal of Computing Sciences in Colleges, 30*(3), 117-125.

Marowka, A. (2008). Think Parallel: Teaching Parallel Programming Today. *IEEE Distributed Systems Online, 9*(8), 1.

Massung, S., & Heeren, C. (2013). Visualizing Parallelism in CS 2. *IPDPSW '13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum* (str. 1252-1256). IEEE Computer Society Washington, DC.

Matos, V., & Grasser, R. (October 2014). An experience on multithreading using Android's handler class. *Journal of Computing Sciences in Colleges, 30*(1), 80-86.

Matthews, S. J. (January 2016). Teaching with parallella: a first look in an undergraduate parallel computing course. *Journal of Computing Sciences in Colleges, 3*, 18-27.

Matthews, S. J. (2017). Using Phoenix++ MapReduce to introduce undergraduate students to parallel computing. *Journal of Computing Sciences in Colleges, 32*(6), 165-174.

Maxim, B. R., Bachelis, G., James, D., & Stout, Q. (1990). Introducing parallel algorithms in undergraduate computer science courses (tutorial session). *SIGCSE '90 Proceedings of the twenty-first SIGCSE technical symposium on Computer science education* (str. 255). Washington, D.C., USA: ACM New York.

McDonald, C. (1992). Teaching concurrency with Joyce and Linda. *SIGCSE '92 Proceedings of the twenty-third SIGCSE technical symposium on Computer science education* (str. 46-52). Kansas City, Missouri, USA: ACM New York, NY, USA.

McGuire, T. J. (January 2010). Introducing multi-core programming into the lower-level curriculum: an incremental approach. *Journal of Computing Sciences in Colleges, 25*(3), 118-119.

Meredith, M. J. (1992). Introducing parallel computing into the undergraduate computer science curriculum: a progress report. *SIGCSE '92 Proceedings of the twenty-third SIGCSE technical symposium on Computer science education* (str. 187-191). Kansas City, Missouri, USA: ACM New York, NY, USA.

Miller, B. P. (June 1993). What to draw? When to draw?: an essay on parallel program visualization. *Journal of Parallel and Distributed Computing - Special issue on tools and methods for visualization of parallel systems and computations, 18*(2), 265-269.

Monismith, D. R. (2015). Incorporating parallelism and high performance computing into computer science courses. *Frontiers in Education Conference (FIE).* El Paso, TX, USA: IEEE.

Moore, M. (March 2000). Introducing parallel processing concepts. *Journal of Computing Sciences in Colleges, 15*(3), 173-180.

Morris, M., & Frinkle, K. (2014). A Three-Semester, Interdisciplinary Approach to Parallel Programming in a Liberal Arts University Setting. *XSEDE '14 Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment* (str. Article No. 66). Atlanta, GA, USA: ACM New York.

Mullen, J., Byun, C., Gadepally, V., Samsi, S., Reuther, A., & Kepner, J. (July 2017). Learning by doing, High Performance Computing education in the MOOC era. *Journal of Parallel and Distributed Computing, 105*, 105-115.

Naps, T. L., & Chan, E. E. (1999). Using visualization to teach parallel algorithms. *SIGCSE '99 The proceedings of the thirtieth SIGCSE technical symposium on Computer science education* (str. 232-236). New Orleans, Louisiana, USA: ACM New York.

Nastavni plan za gimnazije. (2. 3 1994). *Glasnik Ministarstva kulture i prosvjete*, str. 169-173.

Neelima, B. (July 2017). High Performance Computing education in an Indian engineering institute. *Journal of Parallel and Distributed Computing, 105*, 73-82.

Neeman, H., Lee, L., Mullen, J., & Newman, G. (2006). Analogies for teaching parallel computing to inexperienced programmers. *ITiCSE-WGR '06 Working group reports on ITiCSE on Innovation and technology in computer science education* (str. 64-67). Bologna, Italy: ACM New York.

Nevison, C. H. (December 1995). Parallel Computing in the Undergraduate Curriculum. *Computer, 28*(12), 51-56.

Newhall, T., Danner, A., & Webb, K. C. (July 2017). Pervasive parallel and distributed computing in a liberal arts college curriculum. *Journal of Parallel and Distributed Computing, 105*, 53-62.

Niño, J. (2011). Designing an undergraduate curriculum based on parallelism and concurrency. *ACM-SE '11 Proceedings of the 49th Annual Southeast Regional Conference* (str. 1-6). Kennesaw, Georgia: ACM New York.

Offenwanger, A., & Lucet, Y. (2014). ConEE: An Exhaustive Testing Tool to Support Learning Concurrent Programming Synchronization Challenges. *WCCCE '14 Proceedings of the Western Canadian Conference on Computing Education* (str. Article No. 11). Richmond, BC, Canada: ACM New York.

Organick, E. I. (1985). Algorithms, concurrent processors, and computer science education: or, "think concurrently or capitulate?". *SIGCSE '85 Proceedings of the sixteenth SIGCSE technical symposium on Computer science education* (str. 1-5). New Orleans, Louisiana, USA: ACM New York.

Ortiz, A. (2011). Teaching concurrency-oriented programming with Erlang. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 195-200). Dallas, TX, USA: ACM New York.

Pachero, P. (2011). *An Introduction to Parallel Programming.* Morgan Kaufmann.

Palach, J. (2014). *Parallel Programming with Python.* Packt Publishing - ebooks Account.

Pan, Y. (2002). Teaching Parallel Programming Using Both High-Level and Low-Level Languages. *ICCS '02 Proceedings of the International Conference on Computational Science-Part III*, (str. 889-897).

Pan, Y. (2003). An Innovative Course in Parallel Computing. *Journal of STEM Education, 4*(3).

Paprzycki, M. (February 2006). Education: Integrating Parallel and Distributed Computing in Computer Science Curricula. *IEEE Distributed Systems Online, 7*(2), 6.

Peck, C. (2010). LittleFe: parallel and distributed computing education on the move. *Journal of Computing Sciences in Colleges, 26*(1), 16-22.

Petit, S., Sahuquillo, J., Gmez, M. E., & Selfa, V. (July 2017). A research-oriented course on Advanced Multicore Architecture. *Journal of Parallel and Distributed Computing, 105*(C), 63-72.

Plouzeau, N., & Raynal, M. (June 1992). Elements for a course on the design of distributed algorithms. *ACM SIGCSE Bulletin, 24*(2), str. 35-40.

Pollock, L., & Jochen, M. (2001). Making parallel programming accessible to inexperienced programmers through cooperative learning. *SIGCSE '01 Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (str. 224-228). Charlotte, North Carolina, USA: ACM New York.

Prasad, S. K., Chtchelkanova, A., Das, S., Dehne, F., Gouda, M., Gupta, A., . . . Wu, J. (2011). NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 617-618). Dallas, TX, USA: ACM New York.

Predmetni kurikulumi - Informatika. (14. 12 2017). Dohvaćeno iz Ministarstvo znanosti i obrazovanja: https://mzo.hr/sites/default/files/dokumenti/2018/OBRAZOVANJE/Nacionalni-kurikulumi/informatika-6-3-2018.pdf

Prins, P. R. (December 2004). Teaching parallel computing using Beowulf clusters: a laboratory approach. *Journal of Computing Sciences in Colleges, 20*(2), 55-61.

Radenski, A. (2012). Integrating data-intensive cloud computing with multicores and clusters in an HPC course. *ITiCSE '12 Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (str. 69-74). Haifa, Israel: ACM New York.

Rague, B. (2011). Measuring CS1 perceptions of parallelism. *FIE '11 Proceedings of the 2011 Frontiers in Education Conference* (str. S3E-1-1-S3E-6). IEEE Computer Society Washington, DC.

Rague, B. W. (2012). Exploring concurrency using the parallel analysis tool. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 511-516). Raleigh, North Carolina, USA: ACM New York.

Rauber, T., & Runger, G. (2013). *Parallel Programming for Multicore and Cluster Systems.* New York: Springer.

Ricken, M., & Cartwright, R. (2010). Test-first Java concurrency for the classroom. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education* (str. 219-223). Milwaukee, Wisconsin, USA: ACM New York.

Rifkin, A. (1994). Teaching parallel programming and software engineering concepts to high school students. *SIGCSE '94 Proceedings of the twenty-fifth SIGCSE symposium on Computer science education* (str. 26-30). Phoenix, Arizona, USA: ACM New York.

Rivoire, S. (2010). A breadth-first course in multicore and manycore programming. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education* (str. 214-218). Milwaukee, Wisconsin, USA: ACM New York.

Robbins, K. A., Wagner, N. R., & Wenzel, D. J. (June 1989). Virtual rings: an introduction to concurrency. *ACM SIGCSE Bulletin, 21*(2), str. 23-28.

Robbins, S. (2003). Using remote logging for teaching concurrency. *SIGCSE '03 Proceedings of the 34th SIGCSE technical symposium on Computer science education* (str. 177-181). Reno, Navada, USA: ACM New York.

Rogers, M. P. (October 2010). Honey, I shrunk the cluster!: parallel computing and Monte Carlo simulations on iPod Touches, iPhones and iPads. *Journal of Computing Sciences in Colleges, 26*(1), 9-15.

Sadowski, C. (2011). Mental models and parallel program maintenance. *ICSE '11 Proceedings of the 33rd International Conference on Software Engineering* (str. 1102-1105). Waikiki, Honolulu, HI, USA: ACM New York.

Sadowski, C., Ball, T., Bishop, J., Burckhardt, S., Gopalakrishnan, G., Mayo, J., . . . Toub, S. (2011). Practical parallel and concurrent programming. *SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education* (str. 189-194). Dallas, TX, USA: ACM New York.

Sanders, D., & Hartman, J. (1990). Getting started with parallel programming. *SIGCSE '90 Proceedings of the twenty-first SIGCSE technical symposium on Computer science education* (str. 86-88). Washington, D.C., USA: ACM New York.

Saraswat, V. A., & Bruce, K. (2010). Curricula in concurrency and parallelism. *OOPSLA '10 Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (str. 281-282). Reno/Tahoe, Nevada, USA: ACM New York.

Sarkar, V., Grossman, M., Budimlic, Z., & Imam, S. (2017). Preparing an Online Java Parallel Computing Course. *IEEE International Parallel and Distributed Processing Symposium Workshops*, (str. 360-366).

Schaller, N. C., & Kitchen, A. T. (1995). Experiences in teaching parallel computing—five years later. *ACM SIGCSE Bulletin. 27*, str. 15-20. ACM New York.

Schreiner, W. (2002). A java toolkit for teaching distributed algorithms. *ITiCSE '02 Proceedings of the 7th annual conference on Innovation and technology in computer science education* (str. 111-115). Aarhus, Denmark: ACM New York.

Shafi, A., Akhtar, A., Javed, A., & Carpenter, B. (2014). Teaching parallel programming using Java. *EduHPC '14 Proceedings of the Workshop on Education for High-Performance Computing* (str. 56-63). New Orleans, Louisiana: IEEE Press Piscataway, NJ, USA.

Shene, C.-K. (1998). Multithreaded programming in an introduction to operating systems course. *SIGCSE '98 Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education* (str. 242-246). Atlanta, Georgia, USA: ACM New York.

Shoop, E., Brown, R., Biggers, E., Kane, M., Lin, D., & Warner, M. (2012). Virtual clusters for parallel and distributed education. *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education* (str. 517-522). Raleigh, North Carolina, USA: ACM New York, NY, USA.

Stasko, J. T., & Kraemer, E. (June 1993). A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing - Special issue on tools and methods for visualization of parallel systems and computations, 18*(2), 258-264.

Strazdins, P. E. (2012). Experiences in Teaching a Specialty Multicore Computing Course. *IPDPSW '12 Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum* (str. 1283-1288). IEEE Computer Society Washington, DC.

Tikvati, A., Ben-Ari, M., & Kolikant, Y. B.-D. (2004). Virtual trees for the byzantine generals algorithm. *SIGCSE '04 Proceedings of the 35th SIGCSE technical symposium on Computer science education* (str. 392-396). Norfolk, Virginia, USA: ACM New York.

Toll, W. E. (1995). Decision points in the introduction of parallel processing into the undergraduate curriculum. *SIGCSE '95 Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education* (str. 136-140). Nashville, Tennessee, USA: ACM New York, NY, USA.

Torbert, S., Vishkin, U., Tzur, R., & Ellison, D. J. (2010). Is teaching parallel algorithmic thinking to high school students possible?: one teacher's experience. *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education* (str. 290-294). Milwaukee, Wisconsin, USA: SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education.

Touriño, J., Martín, M. J., Tarrio, J. J., & Arenaz, M. (August 2005). A Grid Portal for an Undergraduate Parallel Programming Course. *IEEE Transactions on Education, 48*(3), 391-399.

Tran, Q.-N. (2010). Teaching design & analysis of multi-core parallel algorithms using CUDA. *Journal of Computing Sciences in Colleges, 25*(4), 7-14.

Trümper, J., Bohnet, J., & Döllner, J. (2010). Understanding complex multithreaded software systems by using trace visualization. *SOFTVIS '10 Proceedings of the 5th international symposium on Software visualization* (str. 133-142). Salt Lake City, Utah, USA: ACM New York.

Valentine, D. (2014). HPC/PDC immunization in the introductory computer science sequence. *EduHPC '14 Proceedings of the Workshop on Education for High-Performance Computing* (str. 9-14). New Orleans, Louisiana: IEEE Press Piscataway, NJ.

von Praun, C. (2011). Parallel programming: design of an overview class. *X10 '11 Proceedings of the 2011 ACM SIGPLAN X10 Workshop* (str. Article No. 2). San Jose, California: ACM New York.

Wein, J., Kourtchikov, K., Cheng, Y., Gutierez, R., Khmelichek, R., Topol, M., & Sherman, C. (2009). Virtualized games for teaching about distributed systems. *SIGCSE '09 Proceedings of the 40th ACM technical symposium on Computer science education* (str. 246-250). Chattanooga, TN, USA: ACM New York.

Wilkinson, B. A., & Allen, M. (August 1999). A state-wide senior parallel programming course. *IEEE Transactions on Education, 42*(3), 167-173.

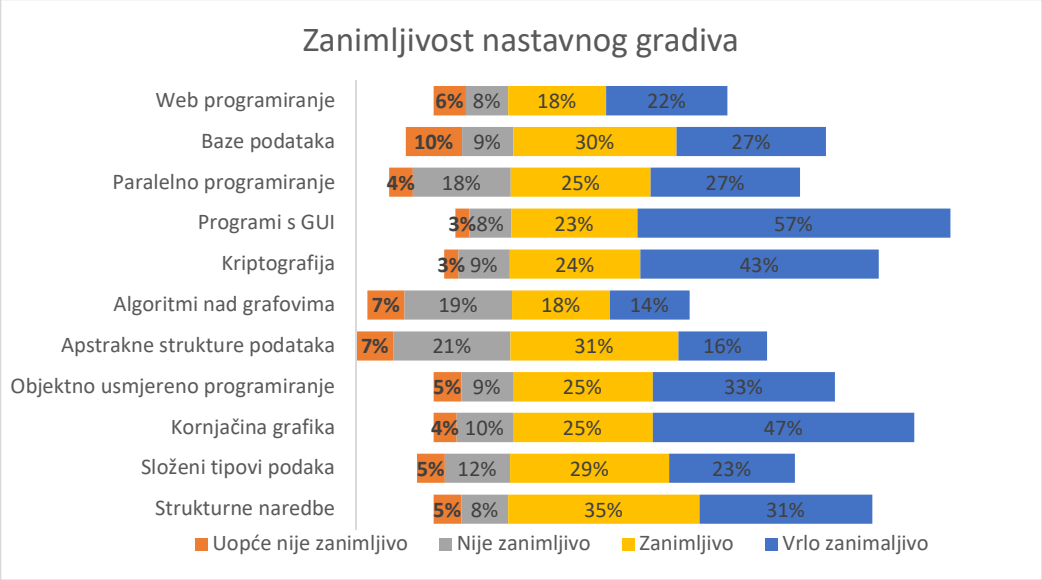
Wolffe, G., & Trefftz, C. (October 2009). Teaching parallel computing: new possibilities. *Journal of Computing Sciences in Colleges, 25*(1), 21-28.

Yeager, D. P. (1991). Teaching concurrency in the programming languages course. *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education* (str. 155-161). San Antonio, Texas, USA: ACM New York.

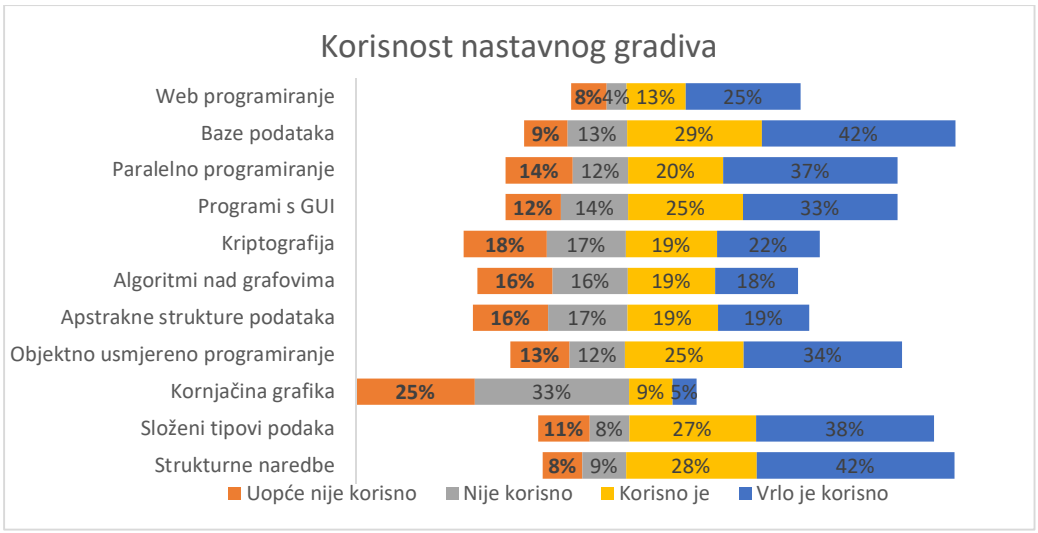
Yue, K.-b. (1994). An Undergraduate Course in Concurrent Programming Using Ad a. *ACM SIGCSE Bulletin* (str. 59-63). ACM New York.

Zimmermann, M., Perrenoud, F., & Schiper, A. (1988). Understanding concurrent programming through program animation. *SIGCSE '88 Proceedings of the nineteenth SIGCSE technical symposium on Computer science education* (str. 27-31). Atlanta, Georgia, USA: ACM New York, NY, USA.

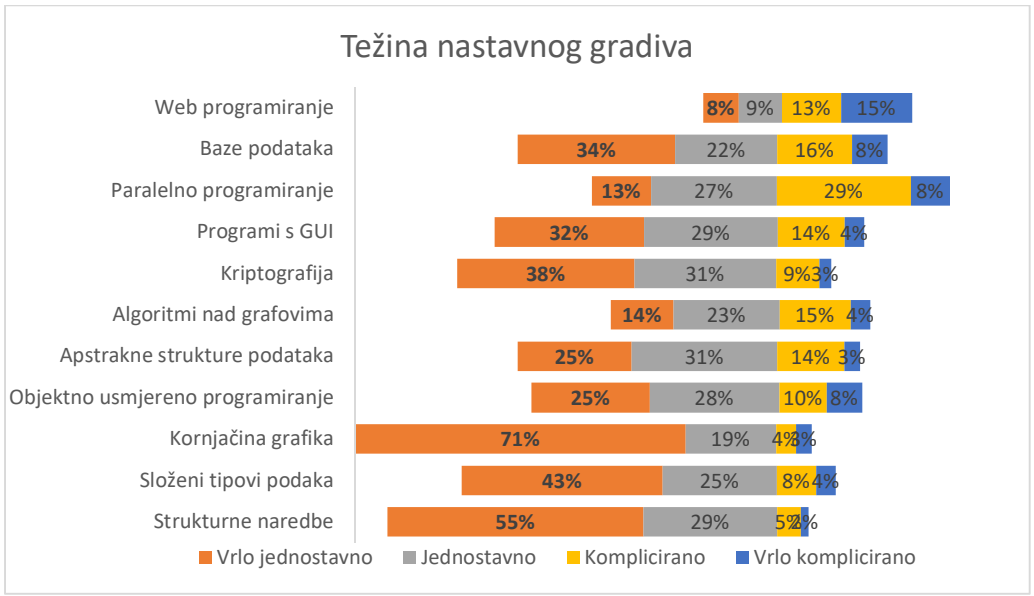
Ziwisky, M., Persohn, K., & Brylow, D. (January 2013). A down-to-earth educational operating system for up-in-the-cloud many-core architectures. *ACM Transactions on Computing Education (TOCE), 13*(1), Article No. 4.



Slika 2: Grafički prikaz procjene zanimljivosti nastavnog gradiva



Slika 3: Grafički prikaz procjene korisnosti nastavnog gradiva



Slika 4: Grafički prikaz procjene težine nastavnog gradiva