Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 007:004.056
DOI: https://doi.org/10.24867/10BE21Indjic

ALAT ZA MERENJE PERFORMANSI OPENMP PROGRAMA

A TOOL FOR MEASURING THE PERFORMANCE OF OPENMP PROGRAMS

Vladimir Inđić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Alat hpcrun je prilagođen tako da koristi OMPT interfejs u cilju olakšane analize performansi izvornog koda OpenMP programa. Upotreba OMPT funkcija povratnog poziva može dovesti do drastičnog povećanja vremena izvršavanja prilikom merenja performansi OpenMP programa sa fino granulisanim paralelizmom.*

Ključne reči: *OpenMP, OMPT, merenje performansi, analiza performansi, fino granulisani paralelizam*

Abstract – *The hpcrun has been extended with OMPT interface support in order to facilitate performance analysis of OpenMP programs' source code. The usage of OMPT call back functions may lead to drastic increase in overhead while measuring the performance of OpenMP programs with fine-grained parallelism.*

Keywords: *OpenMP, OMPT, performance measurement, performance analysis, fine-grained parallelism.*

1. UVOD

Česta upotreba *OpenMP* interfejsa [1] za programiranje višenitnih aplikacija složenih naučnih simulacija, dovela je do potrebe za razvojem kvalitetnih alata za merenje performansi u cilju boljeg razumevanja relativno novog koncepta paralelnog programiranja. Cena programske rutine predstavlja univerzalnu metriku za iskazivanje performansi računarskih arhitektura, poput utrošenog procesorskog vremena, memorijskog zauzeća, itd. S obzirom na to da su cene rutina zavisne od konteksta nastanka njihovog poziva, profajliranje putanji poziva (engl. *call path profiling*) je postao opšte prihvaćen metod za merenje performansi.

Kako bi se smanjio višak vremena uzrokovan profajliranjem putanji poziva (engl. *overhead*), savetuje se upotreba profajlera (engl. *profilers*) zasnovanih na statističkom uzorkovanju [2]. U trenutku prijema uzorka, pomenuti tip profajlera odmotava stek poziva (engl. *call stack*) kako bi formirao kontekst poziva aktivne rutine koji se sastoji od frejmova prisutnih na pomenutnom steku. Formirani konteksti odgovaraju izvršnom kodu *OpenMP* programa, koji u mnogome zavisi od implementacije samog sistema za izvršavanje. Ovo dovodi do skretanje pažnje s analize performansi izvornog koda na analizu performansi implementacije sistema za izvršavanje.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Goran Sladić, vanr. prof.

Kako bi se pomenuti problem prevazišao, neophodno je formirati kontekst poziva rutine koji direktno oslikava izvorni kod. Da bi se to postiglo, potrebno je prevazići problem distribuiranog konteksta nastanka paralelnih regiona [3], a ujedno i sakriti frejmove koji odgovaraju sistemu za izvršavanje iz konteksta poziva rutina. Ovo je bila glavna motivacija za razvoj *OMPT* interfejsa u okviru *OpenMP 5.0* standarda [1]. Pomenuti interfejs je namenjen za upotrebu od strane alata za merenje performansi *OpenMP* programa. Zadatak pomenutog interfejsa je da alatima pruži informacije o trenucima nastanka paralelnih regiona, kao i da ukaže na frejmove sistema za izvršavanje.

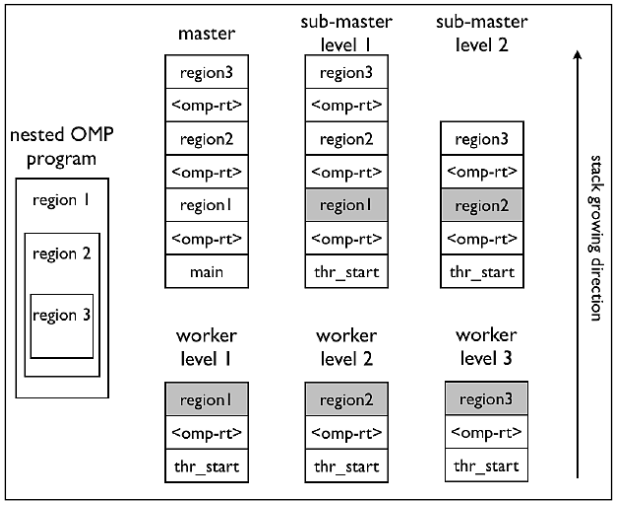
2. Upotreba *OMPT* interfejsa unutar *hpcrun* alata

S obzirom na to da se izvršavanje izvornog koda mahom dešava unutar *OpenMP* zadataka [1], *hpcrun* [4] je proširen podrškom za manipulaciju informacijama o pomenutim zadacima. Zadatak sistema za izvršavanje je da na zahtev profajlera, dostavi informacije koje odgovaraju trenutno aktivnom zadatku, tj. zadatku čiji se kod izvršava u trenutku slanja zahteva. Informacije su predstavljene u vidu 64-bitne reči dodeljene zadatku za vreme njegovog trajanja, kao i para pokazivača na frejmove sistema za izvršavanje koji predstavljaju tačku ulaza u sistem za izvršavanje, odnosno tačku izlaza iz njega [1].

2.1. Formiranje konteksta nastanka paralelnih regiona

Na primeru izvršavanja *OpenMP* programa koji sadrži tri ugnježđena paralelna regiona (slika 1) biće prikazan postupak formiranja konteksta nastanka regiona koji ne sadrži frejmove sistema za izvršavanje. Pomenuti postupak se zasniva na pristupu predloženom od strane *Liu* i ostalih [5]. Unapređenje njihovog pristupa se ogleda u tome što se konteksti nastanka regiona ne skladište u globalnoj *look-up* tabeli, već u 64-bitnoj reči dodeljenoj paralelnom regionu od strane sistema za izvršavanje [1]. Pristup kontekstu regiona više ne zahteva upotrebu složenih sinhronizacionih mehanizama zaduženih da omoguće korišćenje globalne *look-up* tabele od strane više niti konkurentno.

Kako bi se dobile informacije o trenutku nastanka regiona, *hpcrun* mora da registruje funkciju povratnog poziva (engl. *call back function*) koju će sistem za izvršavanje pozvati prilikom kreiranja paralelnog regiona. Pomenutu funkciju izvršiće master niti regiona 1, 2 i 3 prikazanih na slici 1. Njihov zadatak jeste da unutar pomenute funkcije formiraju potpun kontekst nastanka regiona koga kreiraju, pri čemu u obzir moraju uzeti i to da on može biti distribuiran na nekoliko stekova različitih niti.



Slika 1. *Stekovi poziva inicijalne master, podmaster niti i niti radilica tokom izvršavanja OpenMP programa sa tri nivoa ugnježdenog paralelizma [5].*

Naime, prilikom kreiranja regiona 1, inicijalna master nit (*master* na slici 1) odmotava stek poziva i preuzima informacije o implicitnom zadatku. Na osnovu pomenutih informacija pronalazi frejm sistema za izvršavanje koji predstavlja tačku ulaza u pomenuti sistem. Uklanja sve frejmove sistema za izvršavanje koji se nalaze iznad njega (frejmovi sa slike 1 iznad *main* frejma). Na osnovu frejmova koji se nalaze ispod tačke ulaza (*main* frejm na slici 1) formira kontekst nastanka regiona 1, koga čuva u 64-bitnoj reči dodeljenoj ovom regionu.

Tokom kreiranja regiona 2, podmaster nit nivoa 1, odmotava stek poziva i dobavlja informacije o implicitnom zadatku regiona 1. Pomenuta nit je nit radilica regiona 1. Na osnovu navedenih informacija, nit određuje tačke ulaza u sistem za izvršavanje, odnosno izlaza iz njega. Uklanja sve frejmove sistema za izvršavanje koji se nalaze iznad pomenute tačke ulaza (frejmovi sa slike 1 iznad frejma *region1*). Nakon toga, izdvaja frejmove regiona 1 koji su prisutni na njenom steku poziva, a koji se nalaze iznad pomenute tačke izlaza (frejm *region1* sa slike 1). Od izdvojenih frejmova formira delimičan kontekst nastanka regiona 2. Od sistema za izvršavanje traži 64-bitnu reč dodeljenu regionu 1, te iz nje preuzima potpun kontekst nastanka pomenutog regiona. Dodavanjem delimičnog konteksta nastanka regiona 2 na potpun kontekst nastanka regiona 1, formira se potpun kontekst nastanka regiona 2. Potonji biva smešten u 64-bitnu reč dodeljenu pomenutom regionu.

Sličan zadatak ima i podmaster nit nivoa 2 prilikom kreiranja regiona 3. Naime, ona će prvo odrediti delimičan kontekst nastanka regiona 3, koga će pridodati potpunom kontekstu nastanka regiona 2. Na taj način će formirati potpun kontekst nastanka regiona 3, koga smešta u odgovarajuću 64-bitnu reč.

Početak izvršavanja implicitnih, odnosno eksplicitnih zadataka u okviru nekog paralelnog regiona inicira se pozivom pojedinih rutina sistema za izvršavanje. Upravo zato, kao kontekst poziva implicitnog, odnosno eksplicitnog zadatka koristi se kontekst nastanka regiona u okviru koga se pomenuti zadatak izvršava. Javlja se potreba za preuzimanjem konteksta nastanka regiona iz odgovarajuće 64-bitne reči i smeštanjem istog u 64-bitnu reč dodeljenu instanci zadatka. Pomenuti proces obavlja se u funkciji povratnog poziva koja će biti pozvana od strane sistema za izvršavanje prilikom instanciranja implicitnog, odnosno eksplicitnog zadatka.

Kontekst poziva inicijalnog zadatka nema potrebe posebno izdvajati. Naime, on odgovara kodu kojim se započinje izvršavanje programa, te se samim tim nalazi na steku poziva inicijalne master niti.

2.2. Formiranje konteksta poziva rutina

Osnov za formiranje konteksta poziva rutine predstavlja kontekst nastanka regiona, tj. kontekst poziva zadatka u okviru koga se pomenuta rutina izvršava. Potrebno je razmotriti dva slučaju koji se odnose na to da li je uzorak nastao za vreme izvršavanje rutine izvornog koda, odnosno rutine sistema za izvršavanje.

Ukoliko je uzorak nastao za vreme izvršavanja rutine izvornog koda, tada je potrebno formirati kontekst poziva rutine koji se sastoji isključivo od frejmova koji odgovaraju rutinama izvornog koda. U tom slučaju, nit je dužna da odmota stek poziva, kao i da preuzme informacije o aktivnom *OpenMP* zadatku od sistema za izvršavanje. Na osnovu pomenutih informacija, određuje frejm koji predstavlja tačku izlaza iz sistema za izvršavanje, a takođe i kontekst poziva zadatka. Sve frejmove rutina izvornog koda koji se nalaze iznad pomenute tačke izlaza potrebno je iskoristiti kako bi se napravio delimičan kontekst poziva trenutno aktivne rutine izvornog koda. Ovaj kontekst biva pridodat kontekstu poziva zadatka u cilju formiranja potpunog konteksta poziva rutine.

Postavlja se pitanje šta uraditi ukoliko uzorak nastane za vreme izvršavanje rutine sistema za izvršavanje. To se može desiti ukoliko nit čeka da bude upošljena ili ukoliko izvršava određeni posao unutar samog sistema, a koji se tiče kreiranja regiona, kreiranja eksplicitnog zadatka, redukovanja rezultata, pokušaja ulaska u kritički region, itd. Slučaj čekanja niti unutar sistema biće obrađen u sledećem potpoglavlju.

Ukoliko nit obavlja određeni posao unutar sistema za izvršavanje, to znači da ona resurse troši na *overhead* izazvan implementacijom sistema za izvršavanje. Ideja je da se prikazivanjem frejmova sistema za izvršavanje koji se nalaze na vrhu steka, ukaže na neoptimalnosti u implementaciji samog sistema. Naime, nit će odmotati stek i preuzeti informacije o aktivnom *OpenMP* zadatku. Odrediće tačku ulaza u sistem za izvršavanje, kao i kontekst poziva zadatka. Izdvojiće frejmove sistema za izvršavanje koji se nalaze iznad pomenute tačke ulaza, te od njih stvoriti delimičan kontekst poziva aktivne rutine sistema. Delimičan kontekst biva pridodat na kontekst poziva *OpenMP* zadatka, u cilju formiranja potpunog konteksta poziva rutine sistema za izvršavanje.

2.3. Čekanje na poslednjoj implicitnoj barijeri

Ukoliko uzorak nastane dok nit radilica izvršava rutine sistema za izvršavanje kojima se reprezentuje čekanje na poslednjoj implicitnoj barijeri paralelnog regiona, tada se na njenom steku poziva nalaze samo frejmovi sistema za

izvršavanje. Oni će biti uklonjeni i zamenjeni jednim *idle* frejmom. Nakon toga, potrebno je odrediti kontekst čekanja niti. S obzirom na to da nit radilica ne zna da li region na čijoj barijeri čeka i dalje postoji [1], cene čekanja biće akumulirane u privatnoj promenljivoj niti (engl. *thread-private variable*). Ukoliko nit u određenom trenutku primi uzorak izvršavajući eksplicitni zadatak u okviru istog regiona, tada će se akumulirana cena čekanja dodeliti kontekstu nastanka istog. Na ovaj način se korisniku ukazuje da je čekanje izazvano nejednakom raspodelom posla, koja je izazvana izvornim kodom regiona. Ukoliko pak, nit radilica primi sledeći uzorak tokom izvršavanja rutina nekog drugog paralelnog regiona ili sama nit bude uništena od strane sistema za izvršavanje, tada će se akumulirana cena čekanja dodeliti globalnom kontekstu programa. Na ovaj način će korisniku biti ukazano da izvorni kod programa ne generiše dovoljno posla da uposli sve raspoložive resurse.

Inicijalna specifikacija *OMPT* interfejsa [1] nije omogućavala dovoljno informacija na osnovu kojih bi se moglo zaključiti da li nit čeka na poslednjoj implicitnoj barijeri. U cilju prevazilaženja ovog nedostatka, predložene su izmene koje su se ticale manipulacije tipovima *OpenMP* barijera. Od sistema za izvršavanje se zahteva da u svakom trenutku može odrediti na kojoj vrsti barijere unutar regiona nit čeka. Predložene izmene su prihvaćene u trenutnoj reviziji *OpenMP 5.1* standarda [6].

Ukoliko pak master nit čeka na poslednjoj implicitnoj barijeri, tada je postojanje regiona garantovano, te se cena čekanja može dodeliti konteksu nastanka paralelnog regiona. Slično je i sa čekanjem niti na drugim tipovima *OpenMP* barijera, te se njima neće posvetiti pažnja u ovom radu.

2.3. Evaluacija rešenja

Upotrebom *OMPT* interfejsa unutar *hpcrun* alata, cene izvršavanja rutina *OpenMP* programa dodeljene su kontekstima poziva koji odgovaraju izvornom kodu. Na ovaj načina olakšana je analiza performansi izvornog koda pomenutih programa.

Neophodno je izmeriti *overhead* unet upotrebom *OMPT* interfejsa. Ovo će biti urađeno putem dva eksperimenta, koji se zasnivaju na profajliranju nekoliko reper-programa (engl. *benchmarks*). Oba eksperimenta su obavljena na računaru koji ima *Intel Xeon Phi 7250* procesor radnog takta 1,4 GHz, sa 68 jezgara i sa po 4 hardverske niti na svakom jezgru. Od ukupnog broja niti, radi relevantnosti rezultata, korišćeno je 204 niti. Prilikom profajliranja, korišćen je *REALTIME* sistemski brojač sa periodom generisanja uzorka od 10 ms.

Prvi eksperiment se zasniva na profajliranju *benchmark*-a iz skupa *SPEC OMP 2012* [7]. Kako bi se odredio *overhead* profajliranja, svaki od *benchmark*-a je pokrenut po tri puta samostalno, nakon čega je određena srednja vrednost vremena izvršavanja (*avg_bench* u formuli (1)). Nakon toga, svaki od *benchmark*-a je izvršen još po tri puta, pri čemu je njegovo izvršavanje profajlirano upotrebom *hpcrun* alata. Određena su i srednja vremena profajliranih izvršavanja (*avg_hpcrun* u formuli (1)). Na kraju, *overhead* izazvan profajliranjem izračunat je po formuli (1). Radi preglednosti, *overhead* je izražen u procentima:

$$overhead = \left(\frac{avg\_hpcrun}{avg\_bench} - 1\right) * 100 \qquad (1)$$

Kao što se u tabeli 1 može videti, *hpcrun* unosi nizak kontrolisan *overhead* u granicama od 1% do 5%. Izuzetak predstavlja *benchmark 376.tree* koji generiše izuzetno veliki broj kratkih eksplicitnih zadataka. Razlog za to je funkcija povratnog poziva zadužena da odredi kontekst poziva eksplicitnog zadatka. Poziv pomenute funkcije je skuplji od količine posla koju svaki od zadataka obavlja, te dolazi do drastičnog porasta *overhead*-a.

Tabela 1. *Overhead iskazan u procentima unet od strane hpcrun alata prilikom profajliranja SPEC OMP 2012 skupa benchmark-a*

Benchmark	Overhead
350.md	2.62
351.bwaves	2.78
352.nab	1.68
357.bt331	1.40
358.botsalgn	1.50
359.botsspar	2.30
360.ilbdc	3.95
362.fma3d	0.97
363.swim	1.41
367.imagick	2.31
370.mgrid331	0.98
371.applu331	2.12
372.smithwa	1.50
376.kdtree	460.09

Kako bi se pomenuti problem rešio, trebalo bi izbeći registrovanje prethodno pomenute funkcije. Umesto toga, trebalo bi pokušati sa određivanjem konteksta poziva zadatka prilikom obrade uzorka nastalog za vreme izvršavanja zadatka. U tom slučaju, moglo bi se pokušati sa direktnim pristupom kontekstu nastanka regiona u okviru koga se zadatak izvršava, čime bi se izbegla potreba za njegovim kopiranjem u 64-bitnu reč dodeljenu zadatku.

Drugi eksperiment se zasniva na određivanju *overhead*-a nastalog profajliranjem *OpenMP* programa sa fino granulisanim paralelizmom. U tu svrhu, korišćen je *testpr benchmark EPCC* skupa [8]. Pomenuti *benchmark* je iskonfigurisan tako da generiše milion i po kratkih paralelnih regiona, pri čemu svaki od njih izvršava posao u trajanju od oko 1 μs. Pod pojmom kratkog paralelnog regiona, podrazumeva se region čije je trajanje kraće od perioda generisanja uzorka.

Za razliku od prethodnog eksperimenta, *tesptr* je pokrenut 10 puta samostalno, a zatim još 10 puta pri čemu je njegovo izvršavanje profajlirano *hpcrun* alatom. Značenje *avg_bench* i *avg_hpcrun* iz formule (1) promenjeno je tako da određuje srednje vreme potrebno za izvršavanje jedne instance kratkog paralelnog regiona, odnosno izvršavanje instance regiona profajlirano od strane *hpcrun* alata. Eksperiment je pokazao da *hpcrun* unosi *overhead* od 198%. Razlog za to je taj što su regioni veoma kratki, te proces formiranja konteksta nastanka regiona traje duže od njih samih. S obzirom na to da su regioni dosta kraći od perioda generisanja uzorka, velika je verovatnoća da niti neće primiti uzorak za vreme izvršavanja mnogih od njih. Samim tim, konteksti nastanka regiona za vreme

čijeg izvršavanja nisu nastali uzorci bivaju neiskorišćeni, te je njihovo formiranje suvišno.

Nizak *overhead* prilikom profajliranja je neophodan kako bi se dobila realna slika o iskorišćenosti modernih mikroprocesorkih arhitektura tokom izvršavanja fino granulisanog paralelizma [9]. Upravo zbog toga, neophodno je uvesti određene modifikacije pristupa formiranja konteksta nastanka regiona koji *hpcrun* koristi. Umesto formiranja konteksta prilikom kreiranja regiona, potrebno je odložiti proces formiranja do trenutka kada je on zaista neophodan. Odlaganja formiranja konteksta sa sobom nosi i nekoliko izazova.

Pre svega, neophodno je odrediti kada je zaista neki kontekst nastanka regiona neophodan. To znači da niti radilice moraju biti u stanju da obaveste master nit da su primile uzorak za vreme trajanja regiona, te da im je njegov kontekst neophodan. S druge strane, master nit mora biti u stanju da obavesti sve niti radilice o dostupnosti konteksta nastanka regiona u trenutku kada je formiran. Postavlja se pitanje kako formirati kontekst poziva rutine za vreme čijeg izvršavanje je nastao uzorak, ukoliko kontekst nastanka regiona još uvek nije formiran. Takođe, javlja se i problem prilikom formiranja konteksta nastanka ugnježdenih regiona u trenucima nedostupnosti konteksta spoljašnjih regiona. Razrešavanje pomenutih izazova predstavlja prostor za neophodno poboljšanje *hpcrun* alata u cilju primene nad *OpenMP* programima sa fino granulisanim paralelizmom.

3. ZAKLJUČAK

Alat *hpcrun* je modifikovan tako da koristi informacije o *OpenMP* zadacima upotrebom *OMPT* interfejsa. Na taj način, omogućeno je formiranje konteksta poziva rutina distribuiranih na nekoliko stekova različitiih niti, kao i uklanjanje nepotrebnih detalja implementacije. Formirani konteksti u potpunosti odgovaraju izvornom kodu, te olakšavaju analizu performansi istog. Informacije o sistemu za izvršavanje prikazuju se korisniku samo u situacijama kada je potrebno ukazati na *overhead* izazvan implementacijom pomenutog sistema.

Potrebno je posebno ukazati na čekanje niti na poslednjoj implicitnoj barijeri regiona, koje je uzrokovano izvornim kodom istog. Pomenuti kod ili ne generiše dovoljno posla da se iskoriste svi dostupni resursi ili posao neravnomerno raspoređuje na niti tima paralelnog regiona. Da bi se odredilo čekanje na pomenutoj barijeri, predložene su izmene trenutne specifikacije *OMPT* interfejsa koje se tiču manipulacije tipovima *OpenMP* barijera. Pomenute izmene su prihvaćene u trenutnoj reviziji *OpenMP 5.1* standarda.

Alat *hpcrun* uglavnom unosi nizak kontrolisan *overhead* prilikom profajliranja *OpenMP* programa. Izuzetak predstavljaju programi sa fino granulisanim paralelizmom. Kako bi se smanjio *overhead* nastao prilikom profajliranja pomenutih programa, neophodno je revidirati postupak formiranja konteksta nastanka regiona. Umesto formiranja pomenutog konteksta u trenutku kreiranja paralelnog regiona, predlaže se odlaganje formiranja konteksta do trenutka kada je pomenuti zaista neophodan.

Odlaganje unosi vremenski okvir u kome neka od niti može primiti uzorak izvršavajući rutine regiona čiji kontekst nije formiran. To dovodi do potrebe za novim vidom saradnje niti paralelnog regiona u cilju razmene informacija o potrebi konteksta, odnosno o njegovom formiranju. Istraživanje na temu ovakvog vida saradnje može doprineti značajnom poboljšanju kvaliteta merenja performansi *OpenMP* programa sa fino granulisanim paralelizmom.

4. LITERATURA

[1] https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf (pristupljeno u novembru 2018.)

[2] N. Froyd, J. Mellor-Crummey, R. Fowler. Low overhead call path profiling of unmodified, optimized code. *In Proceedings of the 19th annual international conference on Supercomputing*, pages 81–90, 2005.

[3] https://www.openmp.org/wp-content/uploads/ompt-tr2.pdf (pristupljeno u julu 2017.)

[4] L. Adhianto, et all. Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701, 2010.

[5] X. Liu, J. Mellor-Crummey, and M. Fagan. A new approach for performance analysis of openmp programs. *In Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 69–80, 2013.

[6] https://www.openmp.org/wp-content/uploads/openmp-TR9.pdf (pristupljeno u martu 2020.)

[7] https://www.spec.org/omp2012/ (pristupljeno u junu 2020.)

[8] https://www.epcc.ed.ac.uk/research/computing/-performance-characterisation-and-benchmarking/epcc-openmp-micro-benchmark-suite (pristupljeno u junu 2018.)

[9] A. E. Eichenberger, K. O'Brien. Experimenting with low-overhead openmp runtime on ibm blue gene/q. *IBM Journal of Research and Development*, 57(1/2):8–1, 2013.

Kratka biografija:

Vladimir Inđić rođen je u Šapcu 1995. godine. Dilpomirao je na Fakultetu tehničkih nauka 2019. godine iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo i informacione tehnologije sa prosečnom ocenom 10. Od jula 2017. godine radi na unapređenju *hpcrun* alata pod mentorstvom prof. dr *John Mellor-Crummey*, vođe *HPCToolkit* projekta (*Rice* Univerzitet, Hjuston, Teksas, SAD).