IV. IKERGAZTE
NAZIOARTEKO IKERKETA EUSKARAZ

2021eko ekainaren 9, 10 eta 11a
Gasteiz, Euskal Herria

ANTOLATZAILEA:
Udako Euskal Unibertsitatea (UEU)

INGENIARITZA ETA ARKITEKTURA

"Data-intensive" aplikazio baten fitxategi kudeaketa aldatzen memoria mugak gainditzeko

Mikel Iceta Tena, Iñaki Morlán Santacatalina eta *Jose Antonio Pascual Saiz*

161-167 or.
https://dx.doi.org/10.26876/ikergazte.iv.03.20

"Data-intensive" aplikazio baten fitxategi kudeaketa aldatzen memoria mugak gainditzeko

Iceta, M.[1], Morlan, I.[2], Pascual, J.A.[2]

[1] Centro Nacional de Biotecnología (CNB-CSIC). Darwin kalea 3, 28049 Madrid.
[2] Donostiako Informatika Fakultatea (UPV/EHU). Manuel Lardizabal Pasealekua 1, 20018 Donostia.

miceta@cnb.csic.es

Laburpena

Aplikazio zientifikoek RAM memoria izugarria behar izaten dute sarrerako datuak handiak direnean hauek prozesatu ahal izateko. Biologiaren arloan -eta genetikaren atalean zehazki- datu tamaina sorta handia dago: kilobyte gutxi batzuetatik ehunaka gigabyte-ko datu-sekuentziak aurki daitezke. Burrows-Wheeler Alligner (BWA) aplikazioak DNA irakurketak lerrokatzen ditu, gene zehatzak eta genoma baten barne hauen kokapena/k aurkitzeko. Horretarako, memorian kargatzen ditu bai bilatu nahi diren irakurketak eta bai konparatzeke dagoen genoma. Tamaina handiekin lan egiterakoan gerta daiteke ordenagailuak behar adina memoria ez izatea. Lan honetan BWA aplikazioaren memoria kudeaketa aztertu eta aldatu egingo da sarrera handiagoak prozesatu ahal izateko baliabide mugatuak dituzten konputagailuetan.

Hitz gakoak: BWA, C programazio lengoaia, Linux, memoria kudeaketa, mmap

Abstract

Scientific applications tend to be hard on system memory as to process big input data. In the field of biology -concretely on the genomics field- there is a high variety of input data sizes: readings can be found with a size range from a few kilobytes to hundreds of gigabytes. Burrows-Wheeler Aligner (BWA) is an application that aligns DNA readings in order to find specific genes and their exact location in a genome. The application loads both the readings that are to be looked for and the big genome into memory. The computer might not have the sufficient amount of memory to process big input genomes. In this work, the memory management found in BWA will be analyzed and modified in order to allow processing bigger readings in computers with limited resources.

Keywords: *BWA, C programming language, Linux, memory management, mmap*

1. Sarrera eta motibazioa

Izaki bizidunen zelulen osaera aspalditik da ikerketa gai interesgarria. Zelulen barruan dauden DNA sekuentziek gure gorputzen izaera eta bere erakuntza prozesuak bideratzen dituzte. Gene zehatzen kokapena eta bere arteko konbinaketek ezaugarri desberdinak (ezaugarri fisikoak, psikologikoak, mota bateko gaixotasunak izateko aukerak...) markatzen dituzte. Horregatik da genomika ikasketa helburu ohia biologiaren arloan. Konputagailuen agerpena eta hauen hobekuntzek ahalbideratu izan dute genetikaren ikerkuntza hedakorra azken hamarkadetan. DNA sekuentziak kodetu egin daitezke informatikoki, bai karaktere gisa (*A, T, C, G*) edota zenbaki gisa (*0, 1, 2, 3*) osterantzean era anitzetan prozesatu ahal izateko konputagailu batean.

Burrows-Wheeler Alligner (Li eta Durbin, 2009) (Li eta Durbin, 2010) genetikaren arloan erabiltzen den aplikazio ezagun bat da. Sarrera gisa hartzen ditu makina sekuentziadore batek lortutako DNA irakurketa "motz" bat edo bi eta beste "luze" bat, eta irakurketa motza luzean osterantzean bilatzeko. Honela, jakin daiteke gene zehatz bat zelula batetik erauzitako DNA-n bertan dagoen ala ez, zenbat aldiz agertzen den eta bere kokapena. Horretarako, aplikazioak datu guztiak RAM memorian kargatu egiten ditu eta gero memorian bertan egiten ditu bilaketa-eragiketa guztiak.

Sistema batean sartu daitekeen RAM memoria kantitatea mugatua da memoria iraunkorrarekin (HDD, SSD) aldez alde jartzerakoan. Genomen tamainak oso ezberdinak dira: kilobyte gutxi batzuetatik hainbat gigabyte-ko

tarteetan mugitzen dira fitxategi mota hauen tamainak. Kontuan izanda BWA aplikazioak datu guztiak memorian kargatu egiten dituela, gerta daiteke sarrerako datuen tamainen gehiketa RAM memoria baino handiagoa izatea, prozesamendua eragotziz. Memoria nagusiaren (RAM) gehiegizko erabileraren arazoa orokorra da sarrerako datu handiekin lan egiten duten aplikazioekin ("data-intensive applications" ingelesez) (Makrani *et al.*, 2018), eta era askotan saihestu daiteke.

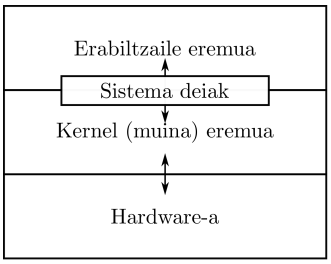
GNU/Linux sistema eragilearen barne aurkitzen da Linux muina. Sistema eragilearen muinak RAM memoria, honen erabilera eta honen gainean egindako eragiketak (besteen artean) kontrolatzen ditu. Erabilgarri dagoen memoria kantitatea bukatzen den heinean, Linux muinak RAM memoriatik diskora mugitzen ditu martxan dabilen programen memoria zati "biziak" *swap out* deitutako prozesuan. Berriro beharrezkoak suertatzen badira exekuzioan zehar, diskotik memoriara bueltan ekartzen ditu *swap in* deitutako prozesuan. Teknika hau baliagarria da memoriatik diskora mugitzen den informazio hori sarritan beharrezkoa ez denean. Horrela ez bada, denbora guztian aurrera-atzera ibiliko da muina beharrezkoak diren memoria zatiekin, sistema osoaren errendimendua andeatuz edota aplikazioaren exekuzioa geldituz[1]. Memoria gabezia arazo larria den egoeretan, aplikazio programatzaileek *malabarismoak* egin behar izaten dituzte kargatutako zatiekin edota aplikatzeko zailak diren teknikak erabiliz.

Agian *ezkutuan* badago memoriarekin elkarri eragiteko beste era bat, sarrerako datu handien prozesamendua ahalbideratu dezakeena?

2. Arloko egoera eta ikerketaren helburuak

Linux muinak aukera anitz ematen ditu fitxategiekin memorian lan egiteko. Diskoan dagoen informazioarekin lan egiteko erarik arruntena eta erabiliena *read()* eta *write()* sistema-deiak erabiltzea da. Bere aldetik, datuekin tratatzeko bi aukera nabarmen ematen ditu paradigma honek: dena RAM memorian kargatzea edo zatizka kargatzea. Fitxategi osoa memorian kolpe bakarrean kargatzeak prozesamenduaren errendimendua igo dezake, baina memoria erabilera ere igotzen du behar adina memoria ez edukitzeko arriskuarekin. Zatizkako karga burutzerakoan, bestalde, programatzaileak aldez aurreko ikerketa bat egin behar du zatien tamaina eta zein zati zein momentutan kargatu behar den erabakitzeko. Horrez gain, kontuan izan behar da *read()* eta *write()* funtzioak sistema-deiak direla. 1. irudian ikusten den bezala, sistema deiek exekuzio eremua aldatzen dute. Hau da, funtzio hauek erabil-

1. Irudia: Sistema Eragilearen exekuzio eremuen sinplifikazioa



tzen dituen aplikazioa erabiltzaile eremuan (pribilegio gutxiko exekuzio eremua) exekutatzen bada, kernel eremura (pribilegio handiko exekuzio eremua) era zuzenean mugitu behar izango da diskoko informazioa irakurri ahal izateko. Aldaketa hauek exekuzio testuinguruaren gordeketa eta denbora nabarmena behar duten bestelako eragiketak dakarte. Beraz, sistema-dei hauek sarritan erabiltzeak errendimendua andeatu egiten du.

Hain ezaguna ez den beste modu bat gordetzen du Linuxek bere barruan: mmap (Gorman, 2004). Interfaze hau ere aurki daiteke Unix-moduko beste sistema eragile batzuetan (BSD, macOS) eta bere oinarrian hiru sistema-dei daude: eratzailea (*mmap()*), aldatzailea (*mremap()*) eta ezabatzailea (*munmap()*). Mmap interfazea erabiltzerakoan diskoan dagoen fitxategi bat proiektatu daiteke prozesu baten memoria espazio birtualean, interfazeak eragiketa guztiak era gardenean burutuz. Honek esan nahi du prozesu batek fitxategi bat atzi dezakeela makinaren RAM memorian fisikoki guztiz kargatuta egongo balitz bezala, benetan egon ez badago ere. Memoria gutxiko egoeretan ez da lehen aipatutako aurrera-atzerako arazoa agertzen, mmap ez baitu diskoko *swap* memoriarekin lan egiten, *orri cache*-arekin baizik. Orri cache honek ezaugarri bereziak ditu, eta fitxategi bat mmap bidez atzitzerakoan lor

[1] Adib. *malloc()* funtzioa exekutatzerakoan. Sisteman erabilgarri dagoen memoria baino gehiago eskatu ezkero, sistema deiak huts egingo du eta NULL erakusle bat bueltatuko du, *segmentation fault* bat eraginez (aplikazioa geldiarazi egiten du muinak kasu honetan).

daiteke dinamikoki kargatu ahal izatea memoria ataskatu barik. Horrela, era erraz batean irakur daiteke fitxategi handi bateko edukia RAM memoria guztia okupatu gabe.

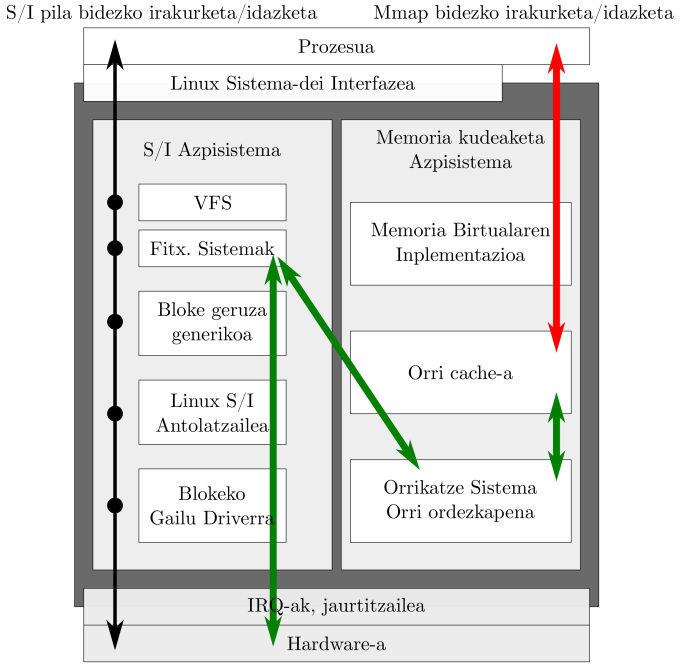
BWA aplikazioa proiektuaren SourceForge biltegian aurki daiteke[2]. Honen barne aurkitzen diren tresna anitzetatik bi dira garrantzitsuenak: *index* eta *mem*. Lehenengoak sarrerako genoma handiaren indizeak sortu egiten ditu formatu espezial batean gordetzeko osterantzean, *Burrows-Wheeler eraldaketa*(Burrows eta Wheeler, 1994) erabiliz datuak konprimitzeko. Bigarrenak indize fitxategi horiek erabiliz, kargatu egiten ditu memorian genoma originala eta irakurketa bat (edo bi), irakurketen sekuentziak genoman bilatzeko. Momentuko inplementazioak arazo bat du, fitxategiak kolpe bakarrean kargatzearen ondorioa dena: datu guztiak aldi berean kargatu ahal izateko memoria asko behar da sisteman edo ezin izango da ezer landu. Artearen egoera honen inguruan eskasa da, argitaratutako lan esanguratsuenak (Van Essen *et al.*, 2012), (Lin *et al.*, 2014) eta (Fedorova, 2019) izanda. Beraz, ikerketa honen helburuak hurrengoak dira:

1. Linux muinaren memoria kudeatze sistemaren ikerketa sakonaren burupena
2. Diskako fitxategi handiak atzitzeko era alternatiboen inguruko ikerketa
3. Mmap eta S/I pila memoria-atzipen metodoen alderapena
4. Mundu errealeko “data-intensive” aplikazio baten (BWA) datu-atzipen teknika aldaketa eta bertsio originalaren aurkako alderapena
5. Mmap interfazearen erabileraren inguruko *artearen egoera*ren hedakuntza

3. Ikerketaren muina

Memoria atzitzeko bi era hauek teknikoki aldez alde jarri baino lehen ikerketa galdera bat proposatu behar da: **mmap interfazea era klasikoen ordez erabiliz gero, errendimendua galdu egiten da ala ez?** Galderari erantzuteko Linux-en datuak atziteko era desberdinen eta hauen inplementazioak aztertu egin dira, bere konplexutasuna neurtzeko. Linux muinaren testuinguruan, diskako datuak era arrunten bidez atzitzeko sarrera/irteera pila (*ingelesez I/O stack*) osoa zeharkatu behar da. 2. Irudiaren eskema sinplifikatuan irudikatzen denaren arabera, S/I pila honek geruza anitz ditu, eta sistema deien bidez erakusten zaio erabiltzaile mailako programei. Mmap interfazeak,

2. Irudia: S/I pila erabiltzeko bidean “geldialdi” gehiago daude mmap-ekoan baino.



berriz, pila hau partzialki sahiesten du orri cache-a erabiliz. Cache honen ezaugarriak direla medio, diska eta RAM

[2] `http://bio-bwa.sourceforge.net/`

memoriaren arteko transferentziak era gardenean kudeatuak izango dira muinaren aldetik. Ondorioz, programak ez du informazioa era esplizituan kopiatu beharrik. Mmap erabiltzerakoan fitxategi bat irakurtzeko, bakarrik lehenengo eragiketak egingo du kopia esplizitoa. Orri cacheak eta bere orri-ekarpen politikak ekarriko dute informazioa modu gardenean osterantzean.

RAM erabilgarririk gabe gelditzeko arriskua dagoen momentuetan (hau da, memoria-presioa handia denean) atzeko planoan exekutatzen den orri cache-aren sistemak erabiltzen ez diren orriak ("hotzak" dauden orriak) ezabatuko ditu, diskarekin sinkronizatuz behar den kasuetan. Bakarrikan mapa sortzeko, eraldatzeko eta ezabatzeko momentuetan izuriko dira sistema-deiak, lehen aipatutako testuinguru aldaketak ia-guztiz sahietsiz. Honekin lehenengo hipotesi bat sortzen da:

(1) **Teorikoki posible da makina baten RAM memoria kopurua baino handiagoa den fitxategi bat "guztiz" erakustea aplikazio bati mmap erabiliz.**

Horrez gain beste hipotesi bat sortu egiten da errendimenduaren inguruan:

(2) **Errendimendu galerarik gabe atzitu daiteke fitxategi bat mmap bidez, S/I metodo tradizionalekin alderatzerakoan.**

Bi hipotesiak frogatzeko esperimentu bat prestatu da. Esperimentuak sintetikoki sortutako fitxategiak irakurri eta idatzi egiten ditu, 1. Taulak aipatzen dituen parametroekin. Aurretiko frogetan 4, 8 eta 16 kilobyte-ko blokeen

1. Taula: Esperimentuan erabilitako aldagaiak bere balio posibleekin.

Aldagaia	Balioa
Teknika	mmap, sist-deia
Froga	irakurketa, idazketa
Fitx. tam.	256MB, 1GB, 4GB
Cache	hotza, beroa
Atzipena	sekuentziala, ausazkoa

desberdintasunak aztertuak izan dira, baina 720 froga burutu eta gero 8KB-eko blokeak bakarrik erabiltzea erabaki da. 1. Taulan agertzen diren konbinazio bakoitzerako 5 froga egin dira, 1etik 5erako "hazi" bat emanez frogak iterazioen artean atzipen patroi berdinak erabiltzeko. Exekuzio multzo bakoitzaren denboraren batazbestekoa eta desbideratze-estandarra kalkulatu dira. Froga guztiak *Intel Xeon Gold 6130* PUZ-a[3], SATA 3 SSD diska bat eta CentOS 7.4[4] duen makina batean exekutatu dira 2. Taulak erakusten dituen banda-zabalerak lortuz.

2. Taula: Froga bakoitzean lortutako banda-zabaleraren batazbestekoa (GB/s-etan) eta desbideraketa estandarra.

Exekuzioa	Froga	256MB	desb	1GB	desb	4GB	dev
hotz sek	MMrd	0.269	0.000	0.227	0.001	0.434	0.002
	IOrd	0.220	0.005	0.085	0.005	0.121	0.002
	MMwr	2.251	0.151	2.085	0.151	0.706	0.592
	IOwr	2.023	0.196	2.009	0.196	0.812	0.435
bero sek	MMrd	0.269	0.000	0.256	0.008	0.500	0.018
	IOrd	0.227	0.001	0.229	0.000	0.229	0.001
	MMwr	2.026	0.116	2.186	0.192	0.896	0.508
	IOwr	1.835	0.209	1.917	0.172	0.958	0.501
hotz ausaz	MMrd	0.267	0.000	0.237	0.011	0.465	0.019
	IOrd	0.153	0.004	0.155	0.001	0.106	0.003
	MMwr	2.749	0.303	2.568	0.259	2.039	0.177
	IOwr	1.855	0.172	1.711	0.129	1.271	0.177
bero ausaz	MMrd	0.267	0.000	0.247	0.007	0.494	0.005
	IOrd	0.229	0.000	0.228	0.001	0.228	0.000
	MMwr	2.753	0.143	2.605	0.203	2.231	0.221
	IOwr	1.879	0.202	1.757	0.157	1.448	0.255

[3] `https://labur.eus/PUZ-Xeon6130`
[4] `https://labur.eus/centos7`

Tauletan erakusten denaren arabera, test sintetikoetan mmap erabiltzea onuragarria da read/write funtzioen erabilerarekin alderatzerakoan. **Baina, aplikazio errealetan hau ere gertatzen da? Zein hobari ekarri liteke BWA moduko aplikazio batean mmap erabiltzeak?**

Burrows-Wheeler Alligner aplikazioaren bi oinarrizko (eta gehien erabilitako) funtzioetan jarriko da arreta: bwa-index eta bwa-mem. Lehenengo funtzioak fasta izeneko formatuan onartzen ditu genoma irakurketak. Fitxategi honetatik abiatuta beste bost fitxategi sortzen ditu: *amb*, *ann*, *bwt*, *pac* eta *sa* izena duten fitxategiak. Hauek tamaina desberdinak dituzte, eta orokorrean behar den memoria zuzenean baldintzatzen dute. 3. Taulan bi eredu agertzen dira memoria erabilera irudikatzeko. Hurrengo funtzioak, bwa-mem, aipatutako fitxategi guztiak (gehi

3. Taula: BWA-index algoritmoak sortzen dituen fitxategiak, bi adibidekin. Pertzentaia sarrerako fasta fitxategiaren tamainarekin dago lotuta.

Fitxategia	SARS-CoV-2	%	ucsc.hg19	%
fasta	72.6MB	1	2.98GB	1
amb	59KB	7.9E-4	8.39KB	2.7E-6
ann	336KB	4.5E-3	3.94KB	1.2E-6
bwt	71.1MB	0.98	2.92GB	0.98
pac	17.8MB	0.24	748MB	0.24
sa	35.6MB	0.49	1.46GB	0.49
TOTALA	197.5MB	2.72	8.1GB	2.72

konparatzeke dauden irakurketak) kargatu egiten ditu RAM memorian aldi berean bilaketa burutzeko. Hau horrela izanda, taulan ikusitako "ucsc.hg19" genoma ezin izango genuke 8GB RAM dituen makina batean prozesatu.

Hau saihesteko bi aukera esanguratsu daude: datuak zati txikiagotan prozesatzea aplikazioak orain arte erabilitako read/write funtzioen bidez edota fitxategiak birtualki proiektatzea mmap interfazea erabiliz. Lehenengo aukerak programazio aldetik eskakizun handiagoa dakar, behar diren memoria zatiak une guztietan kontuan hartu egin behar baitira. Horrez gain, memoria erabilera sistemak duen memoria tamainara gerturatzerakoan, aipatutako *swap* arazoak egongo dira, memoria gabeziak programa geldiarazten ez badu. Mmap interfazea erabiltzerakoan, ordea, "bertan" izango da fitxategia programarentzat, nahiz eta ez egon guztiz kargatuta memorian, eragiketa guztiak "atzetik" gertatuz. Horrez gain, lehenengo metodoak irakurketa eta idazketa txikiago baina anitzagoak ekartzen ditu, sistema dei kopurua eta hauen denbora-galketa handituz.

Sarrerako *fasta* eta sortutako *bwt* eta *sa* fitxategiak mapeatzeak dakar aldaketarik handiena, memoria kantitate handiena suposatzen baitute. Egindako aldaketak BWA aplikazioaren kodean zuzenean egin dira, eta atzigarri daude proiektuko biltegian[5]. Orokorrean, *read()* eta *write()* funtzioen ordez bektoreen gaineko eragiketak/esleipenak sartu dira, bektore hauek mmap bidez sortutako mapak izanda. Aldaketak burutu eta gero, aplikazioaren *index* funtzioa aplikatu zaie sarrerako *fasta* fitxategi desberdinei, 4. Taulan ikusten diren prozesatze-denborak lortuz. Frogak exekutatzeko tamaina desberdinak dituzten *fasta* fitxategiak aukeratu dira, sarrerako tamaina desberdinek

4. Taula: Exekuzio-denboretan ikusitako desberdintasunak bwa-index inplementazio originala eta ikerkuntzaren inplementazioaren artean.

Izena	Fasta tam.	Index (jatorr.)	Index (mmap)	Azelerazio faktorea
Candida Albicans	13.8MB	6.68s	6.14s	**1.087x**
SARS-CoV-2	72.6MB	68.85s	65.91s	**1.044x**
Electrophorus Electricus	533.MB	561.3s	510.1s	**1.100x**
Asparagus Officinalis	1.1GB	1263s	1173s	**1.076x**
Homo Sapiens Sapiens	3.0GB	3942s	3611s	**1.091x**

aplikazioaren errendimenduan izan dezaketen efektuak neurtzeko. Frogetan 1.044 eta 1.100 tarteko azelerazio faktoreak ikusten dira. Hau da, kasu guztietan mmap erabiltzeak hobekuntza erakutsi du. Are gehiago, 8GB RAM dituen beste makina batean hasieran prozesatu ezin zen *Homo Sapiens Sapiens* fitxategiarekin lan egin daiteke kodean aldaketak egin eta gero (infinitutik 3.800 segundotara jaisten da denbora).

[5] `https://gitlab.com/dadi-tfg/bwa-mmap`

4. Ondorioak

Artikuluaren muinean proposatzen diren hipotesiak egiazkoak direla erakusten dute emaitzek. Nahiz eta fitxategiak atzitzeko era arruntena ez izan, mmap-ek eskeintzen ditu bi abantaila nagusi S/I pilaren metodoekin konparatuta:

1. RAM memorian guztiz kargatu ezin daiteken fitxategi batekin lan egitea ahalbideratzen du programatzaileari-rentzat gardena eta erraxa den era batean
2. Fitxategi tamaina batetik aurrera (256MB-tik aurrera) banda-zabalera handiagoa lortzen da mmap erabiliz, exekuzio denbora txikituz

BWA aplikazioaren kasu konkretuarekin igaro egiten dira emaitzak eremu sintetikotik errealera, hipotesiak benetazko "data-intensive" aplikazioetan aplikatu daitezkela erakutsiz.

5. Etorkizunerako planteatzen den norabidea

Behin ikusita BWA-n mmap fitxategi handiekin lan egiteko alternatiba ona dela, lan gehiago egin daiteke horren inguruan, bai BWA aplikazioa eta bai Linux muinaren testuinguruetan:

1. Aplikazioaren aldetik, algoritmoaren bigarren bertsioaren (Vasimuddin *et al.*, 2019) memoria atzipenean "mmap logika" aplikatu daiteke. Honen inguruan saiakerak egin dira proiektuan bertan[6], baina algoritmoaren kodearen aldaketak ez dira xumeak, konbertsioa egiteari arinkeria ebatsiz. Edozein modutan, planteamendu interesgarria da, bigarren bertsio honek memoria askoz gehiago behar baitu funtzionatzeko.
2. Linux muinaren aldetik, orri cache-aren funtzionamendu eta eguneraketa politikak eraldatu daitezke "data-intensive" aplikazioen datu atzipen ereduetan oinarrituz, gaur egun bakarrik ausazko eredu bat inplementatua baitu (Love, 2010). Bestalde, orri erraldoien (huge pages ingelesez) erabilera eta errendimendu-inpaktuaren inguruan ere ikertu daiteke. Honen inguruan aurrerapenak egin dira (Iceta *et al.*, 2020)[7].
3. Esperimentazioaren aldetik, frogak berregitea Linux muinaren bertsio eguneratu batekin da planteamendu argia. Egindako frogak Donostiako Informatika Fakultateak utzitako zerbitzari batean (Linux 3.10 bertsioarekin) burutu dira. Gaur egun Linux-en 5.10.23 LTS bertsioa dago atzigarri. Duela 8 urte atera zen 3.10 bertsioa eta denbora horretan memoriarekin zerikusia duen kodea aldatu da, beraz frogak ez dute guztiz islatzen lortu *daitekeen* errendimendua.

6. Erreferentziak

Burrows, Michael, eta David Wheeler. 1994. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer.

Fedorova, A., 2019. Webpage: Why is mmap faster than system calls. `https://medium.com/@sasha_f/why-mmap-is-faster-than-system-calls-24718e75ab37`. Accessed: 2021-04-21.

Gorman, Mel. 2004. *Understanding the Linux Virtual Memory Manager*. Upper Saddle River, New Jersey: Pearson Education, Inc.

Iceta, Mikel, Pascual Jose A., eta Morlan Ignacio, 2020. Development of file access policies for data-intensive applications in linux. `http://hdl.handle.net/10810/48829`.

Li, Heng, eta Richard Durbin. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics* 25.1754–1760.

——, eta ——. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26.589–595.

Lin, Zhiyuan, Minsuk Kahng, Kaeser Md. Sabrin, Duen Horng Polo Chau, Ho Lee, eta U Kang. 2014. Mmap: Fast billion-scale graph computation on a pc via memory mapping. In *2014 IEEE International Conference on Big Data (Big Data)*, 159–164.

Love, Robert. 2010. *Linux Kernel Development, 3rd ed.*. Upper Saddle River, New Jersey: Pearson Education, Inc.

[6] `https://gitlab.com/dadi-tfg/bwa-mem2-mmap`
[7] `https://gitlab.com/dadi-tfg/dadicode`

Makrani, Hosein Mohammadi, Setareh Rafatirad, Amir Houmansadr, eta Houman Homayoun. 2018. Main-memory requirements of big data applications on commodity server platform. In *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '18, p. 653–660. IEEE Press.

Van Essen, Brian, Henry Hsieh, Sasha Ames, eta Maya Gokhale. 2012. Di-mmap: A high performance memory-map runtime for data-intensive applications. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 731–735. IEEE.

Vasimuddin, M., S. Misra, H. Li, eta S. Aluru. 2019. Efficient architecture-aware acceleration of bwa-mem for multicore systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 314–324.

7. Eskerrak eta oharrak

Ikerketa lan honetan ezinbestekoak izan dira pertsona eta entitate anitz. Ez baninduke lagundu ezin izango nukeen proiektu hau bukatu. Horregatik, eskertu nahi diet:

- Jose A. Pascual eta Inaki Morlan-ei, beraiekin burututako Gradu Amaierako Lanean (Iceta *et al.*, 2020) oinarritzen baita ikerketa, eta prozesu guztian arreta handiz gidatu bainaute.
- Donostiako Informatika Fakultateari eta bertako *Intelligent Systems Group* taldeari, lana aurrera eramateko baliabidetaz hornitu bainaute.
- *University of British Columbia*-ko Elektrizitate eta Konputagailu sailako Alexandra Fedorova-ri, bere Linux-en memoriaren inguruko lana goiargia izan baita muina honen memoria-sistema ulertu ahal izateko.