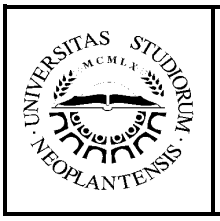


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Стефан Радовановић**
Број индекса: **РА 176/2021**

Тема рада: **Оптимизација AVC енкодера за циљну платформу Banana PI BPI-F3 коришћењем RISC-V векторских инструкција**

Ментор рада: **проф. др Мирослав Поповић**

Нови Сад, Август, 2026

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД**:		Монографска документација
Тип записа, **ТЗ**:		Текстуални штампани материјал
Врста рада, **ВР**:		Завршни (Bachelor) рад
Аутор, **АУ**:		Стефан Радовановић
Ментор, **МН**:		проф. др Мирослав Поповић
Наслов рада, **НР**:		Оптимизација AVC енкодера за циљну платформу Banana Pi BPI-F3 коришћењем RISC-V векторских инструкција
Језик публикације, **ЈП**:		Српски / латиница
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Војводина
Година, **ГО**:		2026
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/22/4/1/8/0/0
Научна област, **НО**:		Електротехника и рачунарство
Научна дисциплина, **НД**:		Рачунарска техника и рачунарске комуникације
Предметна одредница/Кључне речи, **ПО**:		АВЦ енкодер, РИСК-В архитектура, ГЦЦ, ручна векторизација
УДК		
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У овом раду представљена је оптимизација АВЦ енкодера за Банана Пи БПИ-Ф3 платформу засновану на РИСК-В архитектури са векторксом екстензијом. У оквиру рада анализиране су могућности аутоматске векторизације ГЦЦ преводиоца при нивоу оптимизације –О3, након чега су изабране функције АВЦ енкодера додатно оптимизоване применом ручне векторизације коришћењем РИСК-В векторских инструкција. Циљ рада је испитивања могућности постизања додатног побољшања перформанси ручном векторизацијом у односу на аутоматску векторизацију преводиоца.
Датум прихватања теме, **ДП**:		
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	доц. др Миодраг Ђукић
	Члан:	проф. др Иван Каштелан
	Члан, ментор:	проф. др Мирослав Поповић

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Stefan Radovanović	
Mentor, **MN**:		Miroslav Popović, PhD	
Title, **TI**:		Optimization of the AVC encoder for target platform Banana Pi BPI-F3 using RISC-V vector instructions	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2026	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/22/4/1/8/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering and Computer Communications	
Subject/Key words, **S/KW**:		AVC encoder, RISC-V architecture, GCC, manual vectorization	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This paper presents the optimization of an AVC encoder for the Banana Pi BPI-F3 platform based on the RISC-V architecture with a vector extension. The paper analyzes the capabilities of automatic vectorization performed by the GCC compiler at the -O3 optimization level, after which selected functions of the AVC encoder are further optimized using manual vectorization with RISC-V vector instructions. The aim of this paper is to investigate the possibility of achieving additional performance improvements through manual vectorization compared to the compiler's automatic vectorization.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:			
Defended Board, **DB**:	President:	Miodrag Đukić, PhD	
	Member:	Ivan Kaštelan, PhD	Menthor's sign
	Member, Mentor:	Miroslav Popović, PhD	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ИЗРАДУ ЗАВРШНОГ (BACHELOR) РАДА	Датум:

Студијски програм:	Рачунарство и аутоматика		
Студент:	Стефан Радовановић	Број индекса:	РА 176/2021
Област:	Електротехника и рачунарство		
Ментор:	проф. др Мирослав Поповић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА
ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

НАСЛОВ ЗАВРШНОГ (BACHELOR) РАДА:

Оптимизација AVC енкодера за циљну платформу Banana PI BPI-F3 коришћењем RISC-V векторских инструкција

ТЕКСТ ЗАДАТКА:

У оквиру овог дипломског рада потребно је урадити следеће:

1. Упознати се са архитектуром платформе Banana Pi BPI-F3 и њеном подршком за векторске инструкција (RISC-V Vector Extension) као и са структуром и радом AVC енкодера.
2. Евалуирати перформансе AVC енкодера при коришћењу ауто-векторизације (O3) и идентификовати ограничења таквог приступа.
3. Извршити анализу постојеће имплементације AVC енкодера и идентификовати критичне делове кода погодне за векторизацију.
4. Израдити пројекат унапређених оптимизација заснованих на ручној векторизацији коришћењем векторских инструкција.
5. Имплементирати предложене оптимизације у изабране делове AVC енкодера користећи одговарајуће векторске инструкције.
6. Верификовати исправност оптимизоване имплементације поређењем излазног видео садржаја са референтном верзијом.
7. Евалуирати перформансу имплементације са ауто-векторизацијом и ручно оптимизоване верзије (број инструкција, меморијско заузеће, итд.).
8. Уочити предности и мане унапређених оптимизација.
9. Написати текст дипломског рада.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

_____

Кандидат

_____

Захвалност

Захваљујем се свом ментору, проф. др Мирославу Поповићу, као и техничком ментору Душану Стојковић на пруженој помоћи током израде завршног рада.

Такође велику захвалност дугујем својој породици и пријатељима, на безусловној подршци током целукупног процеца школовања.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

GCC - GNU Compiler Collection – ГНУ скуп преводилаца

RISC - Reduced Instruction Set Computer – рачунар са смањеним скупом инструкција

AVC - Advanced Video Coding – напредно видео кодовање

RVV - RISC-V Vector – РИСК-В векторско проширење

H.264 - Standard for video compression – стандард за видео компресију

SIMD - Single Instruction Multiple Data – једна инструкција, више података

LINUX - open-source operating system – оперативни систем отвореног кода

PERF - native performance analyzing tool for Linux – Нативни алат за анализу перформанси за Линукс

CMP - compare tool for Linux – Алат за поређење за Линукс

TOOLCHAIN - set of software development tools – Скуп алата за развој програма

VL - vector length – дужина вектора

SAD - Sum of absolute differences – Збир апсолутних разлика

1. Увод

У савременим мултимедијалним системима, ефикасна обрада видео садржаја представља један од кључних изазова, посебно у контексту ограничених хардверских ресурса и потребе за високом енергетском ефикасношћу. Стандард **AVC** (**H.264**), који се сматра де факто стандардном у индустрији видео компресије, омогућава значајно смањење количине података уз очување квалитета слике, али истовремено поставља високе захтеве у погледу хардверских ресурса. Због тога оптимизација видео енкодера има важну улогу у унапређењу перформанси различитих система, од уграђених уређаја до серверских платформи.

Архитектура **RISC-V**, као отворени и модуларни скуп инструкција, пружа флексибилну основу за истраживање оптимизација на различитим нивоима софтвера. Његова векторска проширења омогућавају паралелну обраду података и представљају значајан потенцијал за убрзање алгоритама карактеристичних за видео енкодере. У том контексту савремени преводиоци, попут **GCC-a**, нуде могућност аутоматске векторизације, чиме се део оптимизација препушта самом преводиоцу. Међутим, степен ефикасности оваквог приступа у великој мери зависи од структуре кода и способности преводиоца да препозна обрасце погодне за паралелизацију.

Поред аутоматске векторизације, ручно имплементиране оптимизације коришћењем вектроских инстуркција омогућавају прецизнију контролу над начином извршавања кода и искоришћењем хардверских ресурса. Иако овакав приступ захтева дубље разумевање архитектуре и додатни развојни напор, он може значајно унапредити перформансе у критичним деловима енкодера.

У овом раду фокус је на поређењу перформанси оптимизација добијених аутоматском векторизацијом у **GCC** преовдиоцу и оних остварених ручном имплементацијом коришћењем векторских инструкција на **RISC-V** платформи. Коришћењем међуархитектурног преводиоца омогућена је израда и тестирање софтвера

за циљну архитектуру, уз анализу перформансиу реалним условима извршавања. Посебан акценат стављен је на идентификацију критичних делова кода који највише профитирају од векторизације, као и на квантитативно поређење постигнутих убрзања.

Циљ рада је да се, на основу добијених резултата, укаже на ограничења постојећих механизама аутоматске векторизације и предложе потенцијална унапређења у виду нових или побољшаних шаблона за препознавање образаца у коду. Истовремено, рад има за циљ да укаже на значај потенцијал **RISC-V** архитектуре у домену мултимедијалне обраде, посебно у контексту ефикасне имплементације савремених алгоритама видео компресије.

2. Теоријске основе

Ово поглавље покрива основне теоријске и техничке основе потребне за започињање имплементације рада.

2.1 GCC преводилац

GNU Compiler Collection (GCC) представља скуп преводиоца отвореног кода намењених за превођење програмских језика као што су **C, C++, Objective C** и други. **GCC** подржава велики број процесорских архитектура попут **x86, ARM** и **RISC-V,** што га чини корисним у развоју системског и уграђеног софтвера. [5]

Основна улога преводиоца јесте превођење изворног кода у машински код циљне платформе. Током процеса превођења **GCC** врши различите оптимизације са циљем повећања перформанси програма, смањење величине извршног кода и ефикасније коришћење хардверских ресурса. **GCC** подржава различите нивое оптимизације. У оквиру оптимизације – O3, која представља највиши ниво оптимизације, аутоматски извршава векторизацију одређених делова кода коришћењем доступних векторских инструкција циљне архитектуре.

Аутоматска векторизација представља процес у ком преводилац препознаје делове кода погодне за паралелну обраду података и аутоматски генерише векторске инструкције. Оваква оптимизација се најчешће примењује над петљама које извршавају исту операцију над већим бројем података. [2] Иако се овим приступом могу значајно унапредити перформансе програма, њене могућности су ограничене сложеношћу кода. Преводилац најчешће није у могућности да безбедно векторизује петље које садрже сложене зависности, неправилне приступе меморији или гранања. Због наведених ограничења аутоматске векторизације, у захтевним апликацијама често се јавља потреба за ручним оптимизацијама коришћењем векторских инструкција, како

би се остварило боље искоришћење хардверских могућности и постигле боље перформансе програма.

2.2 RISC-V архитектура

RISC-V представља отворену и модуларну архитектуру инструкцијског скупа заснованог на **RISC** принципима. [4] За разлику од затворених структура **RISC-V** омогућава произвођачима хардвера и истраживачима слободно коришћење и проширивање архитектуре без лиценцних ограничења.

Једна од главних карактеристика ове архитектуре је поменута модуларност. Основни инструкцијски скуп може се проширивати додатним проширењима у зависности од потреба. Захваљујући овом приступу, **RISC-V** омогућава ефикасну имплементацију како једноставних уграђених система, тако и процесора високих перформанси намењених захтевним рачунарским програмима.

2.2.1 RISC-V векторска проширења (RVV)

RISC-V векторско проширење уводи подршку за векторску обраду података коришћењем **SIMD** принципа рада. Основна идеја оваквог приступа обраде јесте извршавање једне инстуркције над више података истовремено, чиме се постиже значајно убрзање алгоритама који обрађују велике количине података. **RVV** користи скуп векторских регистара над којима се извршавају векторске инстуркције као и променљиве дужине вектора, што омогућава већу флексибилност и бољу преносивост програма између различитих имплементација процесора. [1]

Векторске инструкције су погодне за обраду мултимедијалних података, дигиталну обраду сигнала и других апликација које садрже велики број понављајућих операција над низовима података. Због тога **RVV** представља значајну могућност за оптимизацију перформанси видео енкодера као што је **AVC** енкодер.

2.3 AVC (H.264) видео енкодер

AVC, познат и као **H.264,** представља стандард за компресију видео садржаја развијен са циљем постизања виског квалитета слике уз знатно ниже брзине преноса података од предходних стандарда. Стандард је развије од стране **ITU-T** и **ISO/IEC**, представља

најкоришћенији стандард видео компресије у видео стримингу, телевизијским системима и мултимедијалним апликацијама. [3]

Основни циљ **AVC** стандарда јесте смањење колићине података потребних за складиштење и пренос видео садржаја уз минималан губитак квалитета слике. Ово се постиже коришћењем различитих техника компресије, као што су просторна и временска предикција, трансформације, квантизација и ентропијско кодовање.

2.3.1 AVC енкодер

AVC систем се састоји из енкодера и декодера. Енкодер представља сложенији део система чији је задатак анализа и компресија улазног видео садржаја, док декодер врши реконструкцију компресованог видеа за потребе репродукције. Велики број операција које се извршавају током енкодовања подразумева обраду великих количина података кроз понављајуће математичке операције над низовима пиксела. Због тога **AVC** енкодер представља погодан кандидат за оптимизацију коришћењем векторских инструкција и **SIMD** приступа обради података.

Делови енкодера који су рачунарски захтевни попут предикција, трансформација и филтрирање слике, често представљају критичне тачке за перформансе система. Оптимизација ових делова може значајно смањити време извршавања енкодера и побољшати укупне перформансе обраде видео садржаја.

3. Концепт решења

У овом поглављу биће представљен концепт решења и приступ оптимизацији **AVC** енкодера за **RISC-V** архитектуру, са посебним освртом на примену векторских инструкција. Основна идеја решења заснива се на анализи постојеће имплементације **AVC** пројекта са проширењем за **RISC-V** архитектуру, идентификације функција погодних за оптимизацију, њихова векторизација и проширење постојећег **AVC** пројекта са оптимизацијом енкодера. Такође ће бити описан поступак функционална верификација и евалуација добијених оптимизација.

3.1 Организација пројекта

Пројекат је организован као секвенцијални процес који се састоји од више фаза, почевши од простојеће имплементације **AVC** енкодера, преко његовог превођења и анализе извршавања на **RISC-V** архитектури, до имплементације векторских оптимизација, функционалне верификације и коначне евалуације перформанси. Организација пројекта приказан је на слици 3.1.

Слика 3.1 – Организација пројекта

Процес започиње постојећим **AVC** енкодером који представља полазну основу за развој оптимизоване имплементације. Пројекат се најпре преводи за циљну **RISC-V** архитектуру, након чега се AVC енкодер покреће са одабраним улазним видео записом. Добијени резултат

представља референтну основу за даљу анализу и проверу исправности оптимизованих имплементација.

3.1.1 Анализа извршавања и избор функција за оптимизацију

Након превођења и покретања **AVC** енкодера анализира се његово извршавање коришћењем **perf** алата у **Linux** окружењу. За прикупљање укупних показатеља извршавања користи се **perf stat**, док се расподела извршавања по појединачним функцијама анализира помоћу **perf record** и **perf report**.

На основу добијених резултата идентификују се функције које представљају значајан део укупног извршавања AVC енкодера. Поред удела функција у укупном извршавању, анализира се и структура њеног кода како би се утврдило да ли постоји могућност примене векторске обраде. Посебна пажња усмерена је на функције које садрже већи број понављајућих операција над подацима и које су погодне за примену **SIMD** принципа.

На овај начин се обавља избор функција које представљају приоритетне кандидате за даљу оптимизацију. Циљ није само оптимизација функција која има највећи удео у извршавању, већ избор функција код којих се може очекавти значајно побољшање применом **RISC-V** векторских инструкција.

3.1.2 RISC-V векторска оптимизација

Након идентификације функција погодних за оптимизацију приступа се имплементацији њихових векторских верзија. За реализацију оптимизација користе се **RISC-V** векторске инструкције, чиме се омогућава паралелна обрада више елемената података једном инструкцијом.

Приликом имплементације не врше се измене осногног алгоритма **AVC** енкодера. Оптимизација се реализује на нивоу изабраних функција, при чему се настоји очувати постојећа функционалност и алгоритамски ток енкодовања. На тај начин омогућено је директно поређење резултата оригиналне и оптимизоване имплементације.

3.1.3 Проширење AVC пројекта за RISC-V

Оптимизоване функције се реализују у новим датотекама које су намењене **RISC-V** архитектури. Ове датотеке се укључују у процес превођења приликом изградње **AVC** енкодера за циљну платформу. На овај начин постојећи **AVC** пројекат се проширује подршком за **RISC-V** векторске оптимизације, без потребе за изменом основне имплементације енкодера.

Оваква организација омогућава да оригинална имплементација остане сачувана као референтна верзија, док се **RISC-V** специфичне оптимизације издвајају у додатне имплементационе датотеке. Тиме се олакшава тестирање и поређење различитих верзија енкодера, као и независна анализа утицаја сваке реализоване оптимизације на перформасе система.

3.1.4 Функционална верификација

Након проширења **AVC** пројекта и имплементације оптимизованих функција врши се функционална верификација. Основни услов пре спровођења евалуације перформанси јесте да оптимизована имплементација даје функционално исправан резултат.

За потребе провере користи се исти улазни видео запис који се најпре обрађује оригиналном имплементацијом **AVC** енкодера, а затим и проширеном, оптимизованом имплементацијом. Добијени енкодовани видео запис се декодују, након чега се резултујући видео записи пореде. За поређење се користи **cmp** команда у **Linux** окружењу.

Уколико се резултати оригиналне и оптимизоване имплементације подударају, оптимизација се сматра функционално исправном и може се приступити мерењу њених перформанси. У случају неслагања, имплементација се додатно анализира и коригује пре наставка процеса евалуације.

На овај начин се обезбеђује да евентуално побољшање перформанси није постигнуто на рачун функционалне исправности **AVC** енкодера.

3.1.5 Евалуација и поређење перформанси

Након успешне функционалне верификације евалуира се перформанса реализованих оптимизација. У оквиру евалуације пореде се резултати добијени применом **GCC** аутоматске векторизације при нивоу оптимизације **–O3** са резултатима добијеним ручном векторизацијом коришћењем **RISC-V** векторских инструкција.

На овај начин могуће је утврдити у којој мери ручно реализоване векторске оптимизације унапређују перформансе у односу на код добијен аутоматском векторизацијом **GCC** преводиоца. На основу добијених резултата биће извршено поређење различитих приступа оптимизацији и утврђивање предности и ограничења аутоматске и ручне векторизације за конкретну примену на **RISC-V** архитектури.

4. Програмско решење

У овом поглављу описан је поступак имплементације **RISC-V** векторских оптимизација **AVC** енкодера. Најпре је описано окружење за превођење и извршавање **AVC** енкодера за циљну **RISC-V** архитектуру, након чега је приказан поступак профилисања и избор функција за оптимизацију. Затим су описани принципи примене RISC-V векторских инструкција на одабраним функцијама. У оквиру рада, векторске оптимизације имплементиране су коришћењем векторских уграђених функција. Уграђена функција представљају **C** функције које програмеру омогућавају директно коришћење функционалности векторског проширења **RISC-V** архитектуре, при чему преводилац њихове позиве преводи у одговарајуће машинске векторске инструкције.

Као репрезентативни примери импелементације приказане су оптимизације функција:

- ih264e_sixtapfilter_horz
- ih264e_sixtap_filter_2dvh_vert
- ime_calculate_sad4_prog

На крају поглавља описан је поступак функционалне верификације оптимизованих имплементација.

4.1 Припрема окружења и превођење AVC енкодера

За реализацију и тестирање оптимизација било је неопходно припремити окружење за унакрсно превођење **AVC** пројекта за **RISC-V** архитектуру. **AVC** пројекат је преведен помоћу **RISC-V GCC** алата, чиме је омогућено генерисање извршног кода намењеног циљној архитектури. У коришћеном окружењу коришћен је **riscv64-linux-gnu-gcc** преводилац, на **Ubuntu** оперативном систему.

Пре превођења пројекта, подешени су потребни параметри окружења, укључујући путању до **RISC-V toolchain-a** и одговарајуће параметре за укрштено превођење. Након подешавања окружења извршено је превођење **AVC** пројекта за 64-битну **RISC-V** архитектуру.

У овој фази добијена је почетна верзија **AVC** енкодера која служи као референтна основа за даљу анализу. Над овом верзијом извршено је покретање енкодера са одабраним улазним видео записом и прикупљање података потребних за профилисање извршавања.

4.2 Профилисање и избор функција за оптимизацију

Пре приступа ручној векторизацији било је неопходно утврдити који делови **AVC** енкодера имају највећи утицај на укупне перформансе. У ту сврху коришћен је **perf** алат доступан у **Linux** окружењу.

За прикупљање укупних показатеља извршавања коришћена је команда **perf stat**, док је за детаљну анализу расподеле извршавања по функцијама коришћен **perf report**. На основу резултата **perf report** могуће је утврдити које функције имају највећи удео у укупном броју процесорских циклуса током извршавања **AVC** енкодера.

Подаци добијени профилисањем коришћени су као један од критеријума за избор функција које ће бити предмет даље оптимизације. Међутим, сама количина времена проведеног у функцији није била довољна за њен избор. Поред резултата профилисања анализирана је и структура изворног кода, са циљем да се утврди да ли функција садржи операције погодне за паралелну обрдау применом векторских инструкција.

На основу ове анализе као значајни кандидати за оптимизацију издвојене су функције:

- ih264e_sixtapfilter_horz,
- ih264e_sixtap_filter_2dvh_vert,
- ime_calculate_sad4_prog.

У материјалима са анализе ове функције су издвојене као кандидати за оптимизацију, при чему је за сваку посебно анализирана могућност аутоматске векторизације и начин реализације ручне векторизације.

4.3 Имплементација RISC-V векторских оптимизација

Након избора функција притупило се њиховој ручној векторизацији коришћењем **RISC-V** векторксих инструкција. Основна идеја била је да се скаларне операције које се понављају над

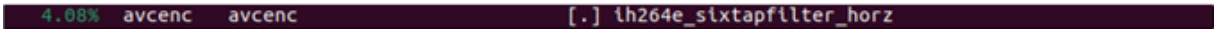
већим бројем елемената података замене векторксим операцијама, тако да се више елемената обради једном векторском инстуркцијом.

При имплементацији је коришћен приступ у ком се подаци учитавају у векторске регистре, по потреби проширују на већи број битова, над њима се извршавају аритметичке операције, а резултат се на крају враћа у одговарајући формат и уписује у меморију. У зависности од природе функције коришћене су различите дужине елемената и различити фактори груписања векторских регистара.

Посебна пажња посвећена је избору одговарајућег типа векторских регистара и вредности **vl**, јер се у различитим деловима алгоритма обрађују подаци различите ширине. На овај начин омогућено је искоришћење векторског проширења без промене основне логике алгоритма.

4.3.1 Оптимизација функције ih264e_sixtapfilter_horz

Функција **ih264e_sixtapfilter_horz** реализује хоризонтални део шестостепеног филтера који се користи у процесу интерполације **AVC** енкодера. Анализом извршавања утврђено је да функција представља погодног кандидата за оптимизацију док је додатном анализом утврђено да **GCC** преводилац није у могућности да аутоматски векторизује целокупну функцију на жељени начин. Основни разлог представља начин приступа подацима, јер се за израчунавање сваког излазног елемента приступа већем броју суседних елемената улазног низа, који се затим користе у више аритметичких операција са одговарајућим коефицијентима. Оваква организација обраде захтева да се подаци учитавају и комбинују у складу са поступком шестостепеног филтрирања, што ограничава могућности преводиоца да целокупну петљу аутоматски преобрази у одговарајући облик векторских операција.



Слика 4.1 – Резултат анализе извршавања

Основна скаларна имплементација обрађује појединачне елементе у петљи. За сваки елемент извршава се комбинација множења коефицијентима шестостепеног филтера, сабирања добијених вредности, померања резултата и ограничавања резултата на опсег 8-битних вредности.

У оптимизованој имплементацији већи број елемената обрађује се паралелно коришћењем **RISC-V** векторских инструкција. Подаци се најпре учитавају у векторске регистре, након чега се над њима примењују операције множења и акумулације. На овај начин се више независних операција из скаларне петље извршава у оквиру исте векторске обраде.

```
void ih264e_sixtapfilter_horz(UWORD8* pu1_src,
    UWORD8* pu1_dst,
    WORD32 src_strd,
    WORD32 dst_strd)
{
    UWORD32  u4_i, u4_j;
    UWORD32  u4_w, u4_h;
    /* width and height of interpolation */
    u4_w = HP_PL_WD;
    u4_h = MB_SIZE;
    pu1_src -= 2;
    for (u4_i = 0; u4_i < u4_h; u4_i++)
    {
        for (u4_j = 0; u4_j < u4_w; u4_j++, pu1_dst++, pu1_src++)
        {
            WORD16 i16_temp;
            i16_temp = ih264_g_six_tap[0] * (*pu1_src + pu1_src[5])
                + ih264_g_six_tap[1] * (pu1_src[1] + pu1_src[4])
                + ih264_g_six_tap[2] * (pu1_src[2] + pu1_src[3]);
            i16_temp = (i16_temp + 16) >> 5;
            *pu1_dst = CLIP_U8(i16_temp);
        }
        pu1_src += src_strd - u4_w;
        pu1_dst += dst_strd - u4_w;
    }
}
```

Слика 4.2 – Оригинална функција

У скаларној имплементацији за сваки елемент излазног бафера израчунава се вредност шестостепеног филтера применом шест улазних вредности и три коефицијента. Након акумулације резултат се помера за одговарајући број битова и ограничава опсег 8-битне вредности.

У векторизованој верзији користи се **RISC-V** уграђене функције за учитавање података, проширено множењем и акумулацију, као и операције померањаи ограничавања резултата. Након извршене аритметике резултат се конвертује у 8-битни формат и уписује у излазни бафер. У наставку је приказан репрезентативни део поступка учитавања и акумулације. Исти поступак се понавља за преостале улазне векторе који учествују у шестотапном филтеру. Приликом обраде већине елемената користи се векторски приступ, док се преостали елемент који не припадају целој векторској гурпи обрађује скаларно.

```
void ih264e_sixtapfilter_horz_rvv(UWORD8 * pu1_src,
    UWORD8 * pu1_dst,
    WORD32 src_strd,
    WORD32 dst_strd)
{
    ...

    vuint8mf2_t v0, v1, v2, v3, v4, v5;
    vuint8mf2_t vtemp;
    vint16m1_t vr;
    size_t vl;

    for (u4_i = 0; u4_i < u4_h; u4_i++)
    {
        WORD16 i16_temp = 0;

        vl = __riscv_vsetvl_e8mf2(16);
        v0 = __riscv_vle8_v_u8mf2(&pu1_src[0], vl);
        vl = __riscv_vsetvl_e16m1(16);
        vr = __riscv_vmv_v_x_i16m1(0, vl);
        vr = __riscv_vwmaccsu_vx_i16m1(vr, ih264_g_six_tap[0], v0, vl);

        ...

        vr = __riscv_vadd_vx_i16m1(vr, 16, vl);
        vr = __riscv_vsra_vx_i16m1(vr, 5, vl);

        vr = __riscv_vmin_vx_i16m1(vr, 255, vl);
        vr = __riscv_vmax_vx_i16m1(vr, 0, vl);

        vl = __riscv_vsetvl_e8mf2(16);
        vtemp = __riscv_vreinterpret_v_i8mf2_u8mf2(
            __riscv_vncvt_x_x_w_i8mf2(vr, vl));
        __riscv_vse8_v_u8mf2(&pu1_dst[0], vtemp, vl);
    }
        pu1_src += 16;
        pu1_dst += 16;

        i16_temp = ih264_g_six_tap[0] * (*pu1_src + pu1_src[5])
            + ih264_g_six_tap[1] * (pu1_src[1] + pu1_src[4])
            + ih264_g_six_tap[2] * (pu1_src[2] + pu1_src[3]);

        i16_temp = (i16_temp + 16) >> 5;

        *pu1_dst = CLIP_U8(i16_temp);

        pu1_src += 1;
        pu1_dst += 1;

        pu1_src += src_strd - u4_w;
        pu1_dst += dst_strd - u4_w;
}
```

Слика 4.3 – Оптимизована функција ih264e_sixtapfilter_horz

Оваква организација омогућава да се задржи функционална еквивалентност са оригиналном имплементацијом, уз истовремено смањење броја инструкција потребних за обраду већег броја елемената. Према извршеним мерењима, оптимизована верзија ове функције остварила је смањење броја **CPU** циклуса од 63,71% у односу на оригиналну имплементацију.

```
  1.64%  avcenc   avcenc                    [.] ih264e_sixtapfilter_horz_rvv
```

Слика 4.4 – Резултат анализе након оптимизације

4.3.2 Оптимизација функције ih264e_sixtap_filter_2dvh_vert

Функција **ih264e_sixtap_filter_2dvh_vert** представља сложенији пример оптимизације јер обухвата две фазе обраде. Прва фаза реализује вертикално филтрирање, након чега се резултат користи као улаз за хоризонтално филтрирање. У оригиналној имплементацији ове операције извршавају се скаларно над појединачним елементима.

Додатни проблем приликом аутоматске векторизације преставља начин приступа подацима у вертикалној фази филтрирања. За израчунавање сваког елемента резултата користи се шест вредности које се налазе у различитим редовима улазног бафера, при чему су

одговарајући елементи међусобно удаљени за вредност корака реда. На тај начин се подацима приступа са одређеним кораком у меморији, уместо секвенцијалног приступа суседним елементима. Овакав начин приступа подацима представља сложенији образац приступа меморији и отежава **GCC** преводиоцу да ову операцију аутоматски преобрази у ефикасан облик векторских операција. Резултат прве, вертикалне фазе филтрирања смештају се у међубафер, организован по редовима, при чему су елементи међурезултата представљени 32-битним вредностима.

У другој фази филтрирања елементима међубафера приступа се хоризонтално, односно за израчунавање једног излазног елемента користи се више суседних елемената истог реда, са померајима од -2 до +3 у односу на тренутну позицију. Оваква организација подразумева да се резултати прве фазе смештају у засебан међубафер, а затим посебно учитавају и обрађују у другој фази, што додатно усложњава организацију векторске обраде.

Ручна оптимизација је реализована тако што се више елемената различитих редова учитава у векторске регистре. Пошто се у међуфази користе подаци веће ширине, 8-битни улазни подаци се проширују на већи број битова пре извршавања аритметичких операција. Након тога се применом операција множења и акумулације реализује шестостепени филтер.

```
void ih264e_sixtap_filter_2dvh_vert_rvv(...)
{
    ...

    vuint8m1_t v0, v1, v2, v3, v4, v5;
    vuint16m2_t v0_ext, v1_ext, v2_ext, v3_ext, v4_ext, v5_ext;
    vint32m4_t vr;
    size_t vl;

    for (row = 0; row < ht; row++)
    {
        pu1_src -= 2;
        pi4_pred_temp -= 2;

        vl = __riscv_vsetvl_e8m1(22);
        v0 = __riscv_vle8_v_u8m1(&pu1_src[-2 * src_strd], vl);
        vl = __riscv_vsetvl_e16m2(22);
        v0_ext = __riscv_vwcvtu_x_x_v_u16m2(v0, vl);
        vl = __riscv_vsetvl_e32m4(22);
        vr = __riscv_vmv_v_x_i32m4(0, vl);
        vr = __riscv_vwmaccsu_vx_i32m4(vr, ih264_g_six_tap[0], v0_ext, vl);

        vl = __riscv_vsetvl_e8m1(22);
        v1 = __riscv_vle8_v_u8m1(&pu1_src[-1 * src_strd], vl);
        vl = __riscv_vsetvl_e16m2(22);
        v1_ext = __riscv_vwcvtu_x_x_v_u16m2(v1, vl);
        vl = __riscv_vsetvl_e32m4(22);
        vr = __riscv_vwmaccsu_vx_i32m4(vr, ih264_g_six_tap[1], v1_ext, vl);

        vl = __riscv_vsetvl_e8m1(22);
        v2 = __riscv_vle8_v_u8m1(&pu1_src[0 * src_strd], vl);
        vl = __riscv_vsetvl_e16m2(22);
        v2_ext = __riscv_vwcvtu_x_x_v_u16m2(v2, vl);
        vl = __riscv_vsetvl_e32m4(22);
        vr = __riscv_vwmaccsu_vx_i32m4(vr, ih264_g_six_tap[2], v2_ext, vl);

        ...

        __riscv_vse32_v_i32m4(&pi4_pred_temp[0], vr, vl);

        pu1_src += src_strd + 2;
        pi4_pred_temp += i4_pred_strd + 2;
    }

    vint32m2_t v0_h, v1_h, v2_h, v3_h, v4_h, v5_h;
    vint32m2_t vr1_h, vr2_h;

    for (row = 0; row < ht; row++)
    {
        vl = __riscv_vsetvl_e32m2(16);
        // vr1_h = pu1_dst1 ; vr2_h = pu2_dst2
        vr2_h = __riscv_vmv_v_x_i32m2(0, vl);
        vr1_h = __riscv_vmv_v_x_i32m2(0, vl);

        v0_h = __riscv_vle32_v_i32m2(&pi4_pred[-2], vl);
        vr2_h = __riscv_vadd_vv_i32m2(vr2_h, v0_h, vl);

        v1_h = __riscv_vle32_v_i32m2(&pi4_pred[-1], vl);
        vr2 h =   riscv vmacc vx i32m2(vr2 h, ih264 g six tap[1], v1 h, vl);

        v2_h = __riscv_vle32_v_i32m2(&pi4_pred[0], vl);
        vr2_h = __riscv_vmacc_vx_i32m2(vr2_h, ih264_g_six_tap[2], v2_h, vl);

        v3_h = __riscv_vle32_v_i32m2(&pi4_pred[1], vl);
        vr2_h = __riscv_vmacc_vx_i32m2(vr2_h, ih264_g_six_tap[2], v3_h, vl);

        v4_h = __riscv_vle32_v_i32m2(&pi4_pred[2], vl);
        vr2_h = __riscv_vmacc_vx_i32m2(vr2_h, ih264_g_six_tap[1], v4_h, vl);

        v5_h = __riscv_vle32_v_i32m2(&pi4_pred[3], vl);
        vr2_h = __riscv_vadd_vv_i32m2(vr2_h, v5_h, vl);

        vr2_h = __riscv_vadd_vx_i32m2(vr2_h, 512, vl);
        vr2_h = __riscv_vsra_vx_i32m2(vr2_h, 10, vl);

        vr1_h = __riscv_vadd_vx_i32m2(v2_h, 16, vl);
        vr1_h = __riscv_vsra_vx_i32m2(vr1_h, 5, vl);
        // CLIP_U8
        vr1_h = __riscv_vmin_vx_i32m2(vr1_h, 255, vl);
        vr1_h = __riscv_vmax_vx_i32m2(vr1_h, 0, vl);
        vr2_h = __riscv_vmin_vx_i32m2(vr2_h, 255, vl);
        vr2_h = __riscv_vmax_vx_i32m2(vr2_h, 0, vl);

        vl = __riscv_vsetvl_e8mf2(16);
        //     pu1_dst2[col] = CLIP_U8(tmp);
        __riscv_vse8_v_u8mf2(&pu1_dst2[0],
            __riscv_vreinterpret_v_i8mf2_u8mf2(
                __riscv_vncvt_x_x_w_i8mf2(
                    __riscv_vncvt_x_x_w_i16m1(vr2_h, vl), vl)), vl);
        //     pu1_dst1[col] = CLIP_U8((pi4_pred[col] + 16) >> 5);
        __riscv_vse8_v_u8mf2(&pu1_dst1[0],
            __riscv_vreinterpret_v_i8mf2_u8mf2(
                __riscv_vncvt_x_x_w_i8mf2(
                    __riscv_vncvt_x_x_w_i16m1(vr1_h, vl), vl)), vl);

        pi4_pred += 16;
        pu1_dst2 += 16;
        pu1_dst1 += 16;

        tmp = (pi4_pred[-2] + pi4_pred[3]) +
            ih264_g_six_tap[1] * (pi4_pred[-1] + pi4_pred[2]) +
            ih264_g_six_tap[2] * (pi4_pred[0] + pi4_pred[1]);

        tmp = (tmp + 512) >> 10;

        pu1_dst2[0] = CLIP_U8(tmp);
        pu1_dst1[0] = CLIP_U8((pi4_pred[0] + 16) >> 5);

        pi4_pred += 1;
        pu1_dst2 += 1;
        pu1_dst1 += 1;
        ...
    }
}
```

Слика 4.5 – Оптимизована функција ih264e_sixtap_filter_2dvh_vert

Након завршетка прве фазе резултати се користе у другој, хоризонталној фази филтрирања. И у овом делу се примењују векторске операције над више елемената истовремено, чиме се избегава појединачна скаларна обрада сваког елемента.

Посебан део имплементације односи се на операције померања и ограничавања резултата. Након извршене аритметике, резултат се доводи у одговарајући опсег и конвертује у формат погодан за упис у излазни бафер.

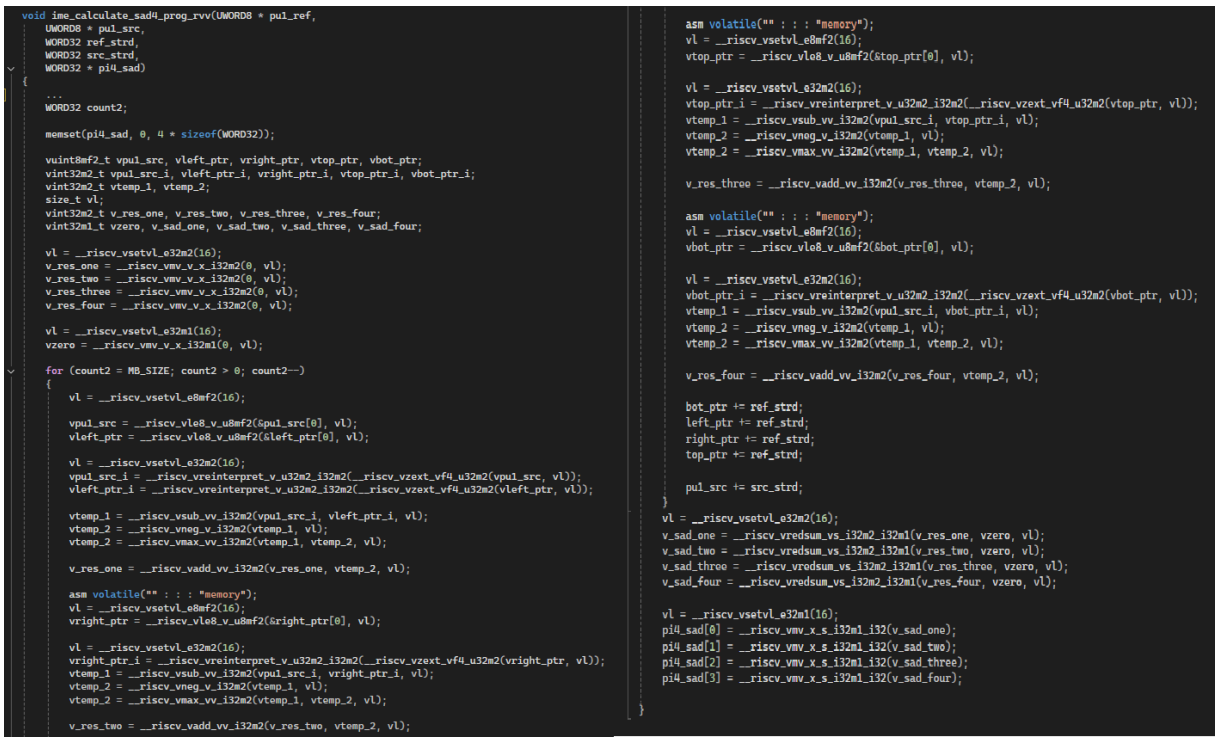
Ова функција представља пример ситуације у којој ручна векторизација захтева детаљније познавање начина на који се подаци обрађују и смештају у меморији, јер је поред саме аритметике потребно прилагодити и приступ подацима захтевима векторске архитектуре.

4.3.3 Оптимизација функције ime_calculate _sad4_prog

Функција **ime_calculate_sad4_prog** користи се за израчунавање суме апсолутних разлика **(SAD – Sum of Absolute Differences)** између изворног блока и четири референтне позиције распоређене у облику дијаманта. За сваку од четири позиције се одговарајућа **SAD** вредност, која се користи у даљем процесу процене тренда.

Ова функција представља погодан кандидат за векторизацију јер се исти тип операције понавља над великим бројем елемената. За сваки елемент потребно је израчунати разлику између вредности изворног и референтног блока, одредити апсолутну вредност разлике и акумулирати резултат.

У векторизованој имплементацији вредности изворног и референтних блокова учитавају се у векторске регистре. Пошто се улазни подаци представљају 8-битним вредностима, врши се њихово проширивање на већи број битова пре извршавања аритметичких операција.



```
void ime_calculate_sad4_prog_rvv(UWORD8 * pu1_ref,
    UWORD8 * pu1_src,
    WORD32 ref_strd,
    WORD32 src_strd,
    WORD32 * pi4_sad)
{
    ...
    WORD32 count2;

    memset(pi4_sad, 0, 4 * sizeof(WORD32));

    vuint8mf2_t vpu1_src, vleft_ptr, vright_ptr, vtop_ptr, vbot_ptr;
    vint32m2_t vpu1_src_i, vleft_ptr_i, vright_ptr_i, vtop_ptr_i, vbot_ptr_i;
    vint32m2_t vtemp_1, vtemp_2;
    size_t vl;
    vint32m2_t v_res_one, v_res_two, v_res_three, v_res_four;
    vint32m1_t vzero, v_sad_one, v_sad_two, v_sad_three, v_sad_four;

    vl = __riscv_vsetvl_e32m2(16);
    v_res_one = __riscv_vmv_v_x_i32m2(0, vl);
    v_res_two = __riscv_vmv_v_x_i32m2(0, vl);
    v_res_three = __riscv_vmv_v_x_i32m2(0, vl);
    v_res_four = __riscv_vmv_v_x_i32m2(0, vl);

    vl = __riscv_vsetvl_e32m1(16);
    vzero = __riscv_vmv_v_x_i32m1(0, vl);

    for (count2 = MB_SIZE; count2 > 0; count2--)
    {
        vl = __riscv_vsetvl_e8mf2(16);

        vpu1_src = __riscv_vle8_v_u8mf2(&pu1_src[0], vl);
        vleft_ptr = __riscv_vle8_v_u8mf2(&left_ptr[0], vl);

        vl = __riscv_vsetvl_e32m2(16);
        vpu1_src_i = __riscv_vreinterpret_v_u32m2_i32m2(__riscv_vzext_vf4_u32m2(vpu1_src, vl));
        vleft_ptr_i = __riscv_vreinterpret_v_u32m2_i32m2(__riscv_vzext_vf4_u32m2(vleft_ptr, vl));

        vtemp_1 = __riscv_vsub_vv_i32m2(vpu1_src_i, vleft_ptr_i, vl);
        vtemp_2 = __riscv_vneg_v_i32m2(vtemp_1, vl);
        vtemp_2 = __riscv_vmax_vv_i32m2(vtemp_1, vtemp_2, vl);

        v_res_one = __riscv_vadd_vv_i32m2(v_res_one, vtemp_2, vl);

        asm volatile("" : : : "memory");
        vl = __riscv_vsetvl_e8mf2(16);
        vright_ptr = __riscv_vle8_v_u8mf2(&right_ptr[0], vl);

        vl = __riscv_vsetvl_e32m2(16);
        vright_ptr_i = __riscv_vreinterpret_v_u32m2_i32m2(__riscv_vzext_vf4_u32m2(vright_ptr, vl));
        vtemp_1 = __riscv_vsub_vv_i32m2(vpu1_src_i, vright_ptr_i, vl);
        vtemp_2 = __riscv_vneg_v_i32m2(vtemp_1, vl);
        vtemp_2 = __riscv_vmax_vv_i32m2(vtemp_1, vtemp_2, vl);

        v_res_two = __riscv_vadd_vv_i32m2(v_res_two, vtemp_2, vl);

        asm volatile("" : : : "memory");
        vl = __riscv_vsetvl_e8mf2(16);
        vtop_ptr = __riscv_vle8_v_u8mf2(&top_ptr[0], vl);

        vl = __riscv_vsetvl_e32m2(16);
        vtop_ptr_i = __riscv_vreinterpret_v_u32m2_i32m2(__riscv_vzext_vf4_u32m2(vtop_ptr, vl));
        vtemp_1 = __riscv_vsub_vv_i32m2(vpu1_src_i, vtop_ptr_i, vl);
        vtemp_2 = __riscv_vneg_v_i32m2(vtemp_1, vl);
        vtemp_2 = __riscv_vmax_vv_i32m2(vtemp_1, vtemp_2, vl);

        v_res_three = __riscv_vadd_vv_i32m2(v_res_three, vtemp_2, vl);

        asm volatile("" : : : "memory");
        vl = __riscv_vsetvl_e8mf2(16);
        vbot_ptr = __riscv_vle8_v_u8mf2(&bot_ptr[0], vl);

        vl = __riscv_vsetvl_e32m2(16);
        vbot_ptr_i = __riscv_vreinterpret_v_u32m2_i32m2(__riscv_vzext_vf4_u32m2(vbot_ptr, vl));
        vtemp_1 = __riscv_vsub_vv_i32m2(vpu1_src_i, vbot_ptr_i, vl);
        vtemp_2 = __riscv_vneg_v_i32m2(vtemp_1, vl);
        vtemp_2 = __riscv_vmax_vv_i32m2(vtemp_1, vtemp_2, vl);

        v_res_four = __riscv_vadd_vv_i32m2(v_res_four, vtemp_2, vl);

        bot_ptr += ref_strd;
        left_ptr += ref_strd;
        right_ptr += ref_strd;
        top_ptr += ref_strd;

        pu1_src += src_strd;
    }
    vl = __riscv_vsetvl_e32m2(16);
    v_sad_one = __riscv_vredsum_vs_i32m2_i32m1(v_res_one, vzero, vl);
    v_sad_two = __riscv_vredsum_vs_i32m2_i32m1(v_res_two, vzero, vl);
    v_sad_three = __riscv_vredsum_vs_i32m2_i32m1(v_res_three, vzero, vl);
    v_sad_four = __riscv_vredsum_vs_i32m2_i32m1(v_res_four, vzero, vl);

    vl = __riscv_vsetvl_e32m1(16);
    pi4_sad[0] = __riscv_vmv_x_s_i32m1_i32(v_sad_one);
    pi4_sad[1] = __riscv_vmv_x_s_i32m1_i32(v_sad_two);
    pi4_sad[2] = __riscv_vmv_x_s_i32m1_i32(v_sad_three);
    pi4_sad[3] = __riscv_vmv_x_s_i32m1_i32(v_sad_four);
}
```

Слика 4.6 – Оптимизована функција ime_calculate_sad4_prog

За израчунавање апсолутне вредности разлике коришћене су векторске операције одузимања, негирања и избора максималне вредности. Добијене вредности се затим акумулирају у одговарајуће векторске регистре. Након обраде свих елемената користи се редукциона операција како би се елементи векторског резултата сабрали и добила коначна **SAD** вредност за сваку од четири референтне позиције.

Овим приступом једна функција истовремено представља четри независна **SAD** израчунавања, што је погодно за примену векторске архитектуре. У односу на скаларну имплементацију, више елемената се обрађује паралелно, док се коначно сабирање врши применом векторске редукције.

4.4 Интеграција оптимизованих функција у AVC пројекат

Након имплементација RISC-V векторизованих функција било је потребно интегрисати их у постојећи AVC пројекат. При томе није мењан основни алгоритам AVC енкодера, већ су оптимизоване имплементације реализоване у додатним датотекама намењеним RISC-V архитектури.

Приликом превођења AVC пројекта за циљну архитектуру одговарајуће оптимизоване функције се укључују у процес изградње. На тај начин се постојећа имплементација проширује RISC-V специфичним оптимизацијама, док неизмењене функције из референтне имплементације остају и даље доступне.

Оваква организација омогућава да се оригинална и оптимизована верзија енкодера извршавају над истим улазним подацима и да се њихови резултати независно провере и упореде.

5. Резултати

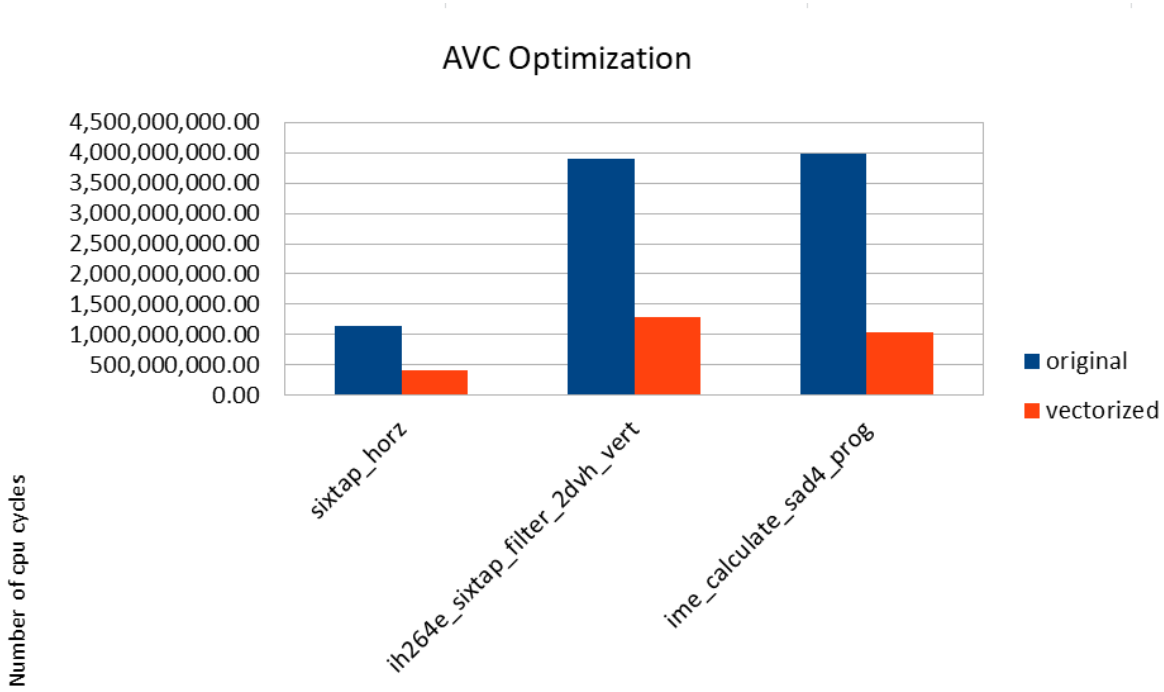
У овом поглављу приказани су резултати добијени након имплементације **RISC-V** векторских оптимизација изабраних функција **AVC** енкодера. Циљ мерења је био да се утврди утицај ручно реализованих векторских оптимизација на број процесорских циклуса потребних за извршавање посматраних функција.

Мерења су извршена за три предходно изабране функције на основу профилисања **AVC** енкодера. Резултати мерења броја процескорских циклуса приказани су у табели 5.1. Вредности приказане у табели представљају средњу вредност броја процесорских циклуса потребних за извршавање посматраних функција, добијену на основу више понављања мерења.

Табела 5.1 – Поређење броја процесорских циклуса оригиналних и векторизоцаних функција

Функција	Оригинална имплементација	Векторизована имплементација	Смањење
sih264e_sixtapfilter_horz	1 142 751 240	414 675 825	64%
ih264e_sixtap_filter_2dvh_vert	3 891 623 500	1 298 545 149	67%
ime_calculate_sad4_prog	3 977 805 818	1 038 533 234	74%

Из резултата се може уочити значајно смањење броја процесорских циклуса код свих посматраних функција. Најмање смањење остварено је код функције **ih264e_sixtapfilter_horz**, где је број циклуса смањен за 64% што одговара приближном убрзању од 2,76 пута. Код функције **ih264e_sixtap_filter_2dvh_vert** остварено је смањење од 67% односно убрзање од 3 пута. Док је највеће побољшање постигнуто код функције **ime_calculate_sad4_prog** код које је број циклуса смањен за 74% односно постигнуто убрзање од 3,83 пута.



Слика 5.1 – Смањење броја процесорских циклуса применом векторизације

5.1 Ограничења тестирања

Добијени резултати представљају показатељ утицаја примењених векторских оптимизација на перформансе посматраних функција, али их је потребно посматрати у контексту услова под којима су мерења извршена. Мерења су спроведена коришћењем једног процесорског језгра, тако да приказано смањење броја процесорских циклуса представља резултат оптимизације у условима једнојезграног извршавања.

Резултати се због тога не могу директно генерализовати на укупно време извршавања **AVC** енкодера при коришћењу већег броја процесорских језгара, где би на перформансе утицали и паралелно извршавање, расподела оптерећења и други фактори система. Такође, за потпунију процену ефикасности предложених оптимизација било би потребно спровести већи број тестова над различитим улазним видео материјалима, као и испитати различита подешавања енкодера и услова извршавања.

Ипак, резултати добијени у оквиру спроведених мерења показују јасан тренд смањења броја процесорских циклуса код свих посматраних функција и представљају основу за даље тестирање примене **RISC-V** векторских инструкција у овом пољу.

6. Закључак

У оквиру овог рада разматране су могућности примене **RISC-V** архитектуре са векторским проширењем у области обраде видео података, са посебним освртом на оптимизацију **AVC** енкодера за **Banana PI BPI-F3** платформу. Као полазна основа коришћена је верзија **AVC** енкодера преведена уз примену оптимзације **–O3**, при чему је испитивано у којој мери додатна ручна контрола над векторским операцијама може да побољша перформансе изабраних функција.

Аутоматска векторизација **GCC** преводиоца преставља значајно побољшање у односу на конвенционални, скаларни начин извршавања, јер омогућава преводиоцу да искористи могућности векторске архитеткуре тамо где је то могуће.

Међутим, резултати добијени у оквиру овог рада показују да у одређеним критичним деловима **AVC** енкодера постоји додатни простор за оптимизацију применом ручно реализованих векторских операција.

Добијени резултати указују да ручна векторизација може да обезбеди значајно додатно побољшање у односу на резултате добијене аутоматском векторизацијом. Предност ручног приступа огледа се пре свега у већој контроли над начином обраде података, избором векторских операција и организацијом података у векторским регистрима. Поред тога, ручна векторизација представља могуће решење за делове програма за које **GCC** преводилац не може да примени аутоматску векторизацију, односно за које није могуће генерисати одговарајући векторски код. Такав приступ захтева детаљније познавање циљне архитектуре и доводи до сложенијег кода, што представља један од основних недостатака преводиоца.

На основу спорведеног рада може се закључити да аутоматска векторизација представља добру полазну основу за оптимизацију, али да ручна векторизација критичних делова програма може да обезбеди додатно побољшање перформанси када могућности аутоматксе векторизације нису у потпуности искоришћене.

Са друге стране, недостаци ручне оптимизације огледају се у значајном ангажовању високо стручних инжењера, дужем времену реализације и већој могућности појаве програмских грешака. Додатни недостатак представља и зависност имплементације од конкретне архитектуре и њених векторских могућности, што може отежати пренос оваквих оптимизација на друге архитектуре.

Један од могућих праваца будућег рада јесте развој аутоматизованог приступа заснованог на техникама вештачке интелигенције, који би пружио подршку инжењерима при анализи програмског кода, идентификацији функција погодних за векторизацију и имплементацији одговарајућих ручних оптимизација. На овај начин би се могло смањити време потребно за анализу и развој оптимизованих имплементација, уз задржавање могућности ручне контроле над коначним решењем.

На тај начин, даљи развојби могао да обједини предности аутоматизованих техника оптимизације и стручне интервенције инжењера, са циљем ефикаснијег искоришћења векторских могућности **RISC-V** архитектуре.

7. Литература

[1] " RISC-V International, *"V" Standard Extension for Vector Operations*", [Online]. Available: https://docs.riscv.org/reference/vector-c-intrinsics/v1.0/rvv-intrinsic-spec.html

[2] "GNU Project, *Auto-vectorization in GCC*", [Online]. Available: https://gcc.gnu.org/projects/tree-ssa/vectorization.html

[3] "ITU-T, *Recommendation H.264 – Advanced video coding for generic audiovisual services*", [Online]. Available: https://www.itu.int/rec/t-rec-h.264

[4] "GNU Project, *RISC-V Options*", [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html

[5] "GNU Project, GCC", [Online]. Available: https://gcc.gnu.org