АРХИТЕКТУРА РАЧУНАРА ПРОЈЕКТНИ ЗАДАТАК 1.2 – Асемблерски програм за обраду података

ТЕМА: Линеарна регресија

Аутор: Новица Тепић

Верзија: 1.0

Датум: 22.12.2022.

Садржај

[Увод 3](#_Toc125229830)

[Поступак и услови тестирања 3](#_Toc125229831)

[Поређење времена извршавања 4](#_Toc125229832)

[Поређење времена извршавања – Оптимизације gcc компајлера 8](#_Toc125229833)

[Закључак 12](#_Toc125229834)

# Увод

Конкретан проблем рачунања параметара линеарне регресије ријешен је рачунањем параметара за линеарну регресију. С обзиром да је у задатку тражено да се уради верзија са стандардним инструкцијским скупом и са одређеним инструкцијским скупом (у мом случају AVX), и једна и друга верзија су направљене и тестиране. Битно је напоменути да сам направио и верзију која ради са SSE, али с обзиром да је floating point ограничен резултати нису задовољавајући, мада ћу и то документовати у наставку документа. Поред тога, програм је реализован и у C програмском језику, гдје сам испробао различите оптимизације, које ће бити наведене касније у документу. Сва времена извршавања и њихова поређења су адекватно извршена и документована. Укратко, идеја је била да се број елеманта који ће се израчунавати у сумама као и сами елементи учитају из фајла, те да се након тога они они искористе да бисмо израчунали параметре и уписали их у излазни фајл, гдје су и улазни и излазни фајл наведени као аргументи командне линије.

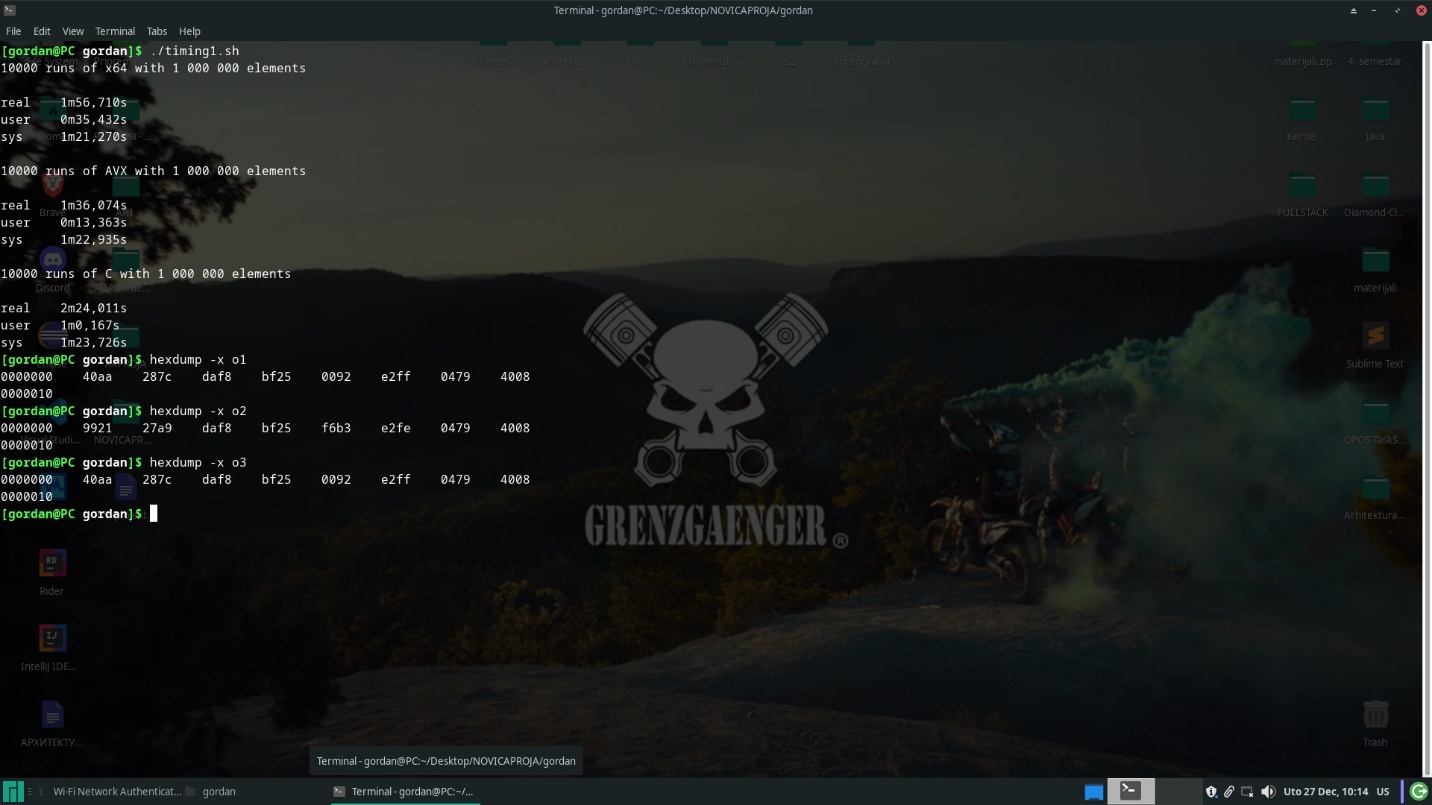
# Поступак и услови тестирања

Тестирање је извршено користећи виртуалну машину на којој је инсталиран оперативни систем Ubuntu 22.04.1 LTS, гдје сам машини додијелио четири логичка језгра, иако их имам осам. Разлог за то је што ми виртуална машина (у овом случају VirtualBox) није дозвољавала већи број језгара, те сам добио и упозорење да би перформансе биле значајно деградиране. Процесор који сам користио је AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10GHz.

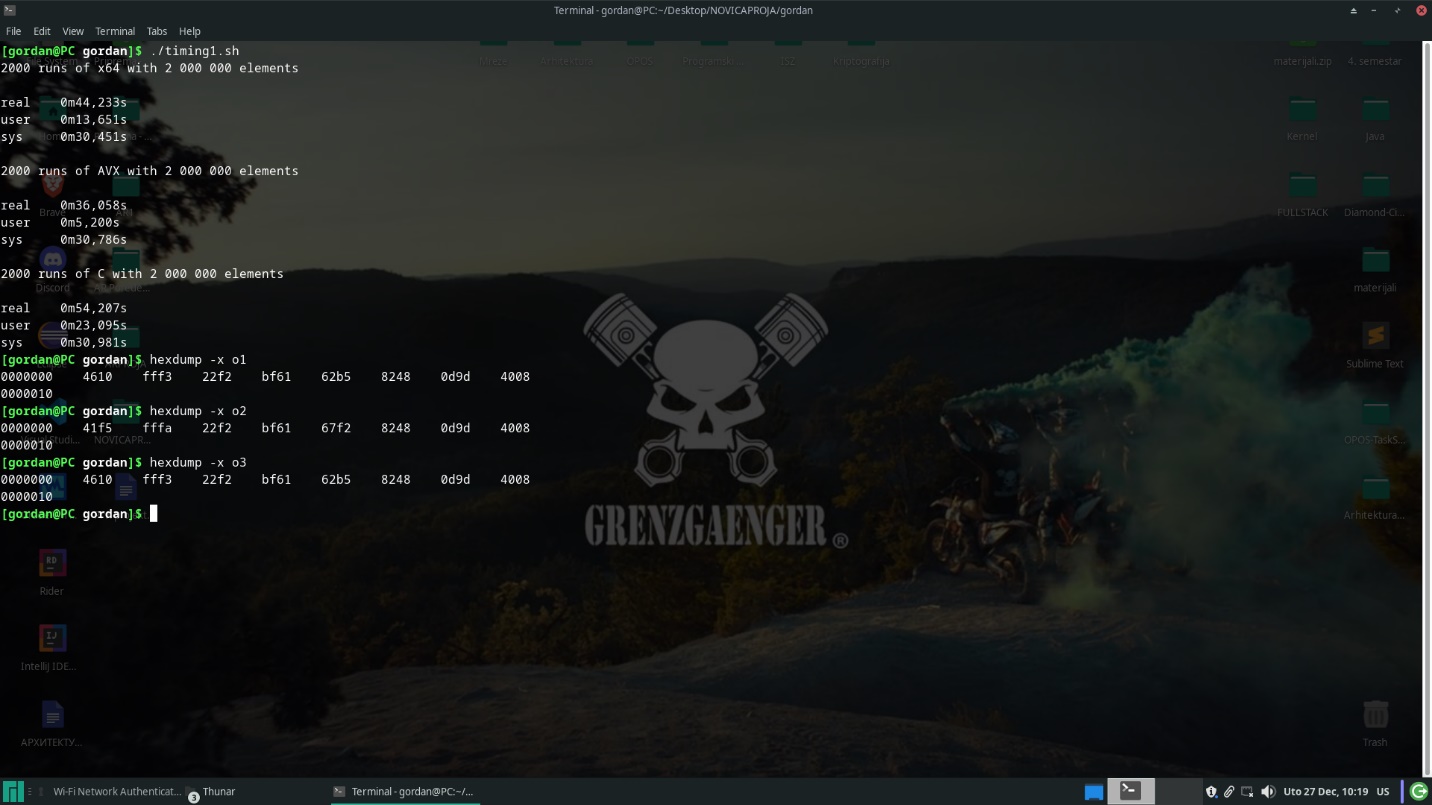
Да би се покренуло тестирање, кориштена је shell скрипта са moodle сајта предмета, коју сам модификовао на различите начине тако да су се програми извршавали различит број пута и са различитом количином улазних података.

# Поређење времена извршавања

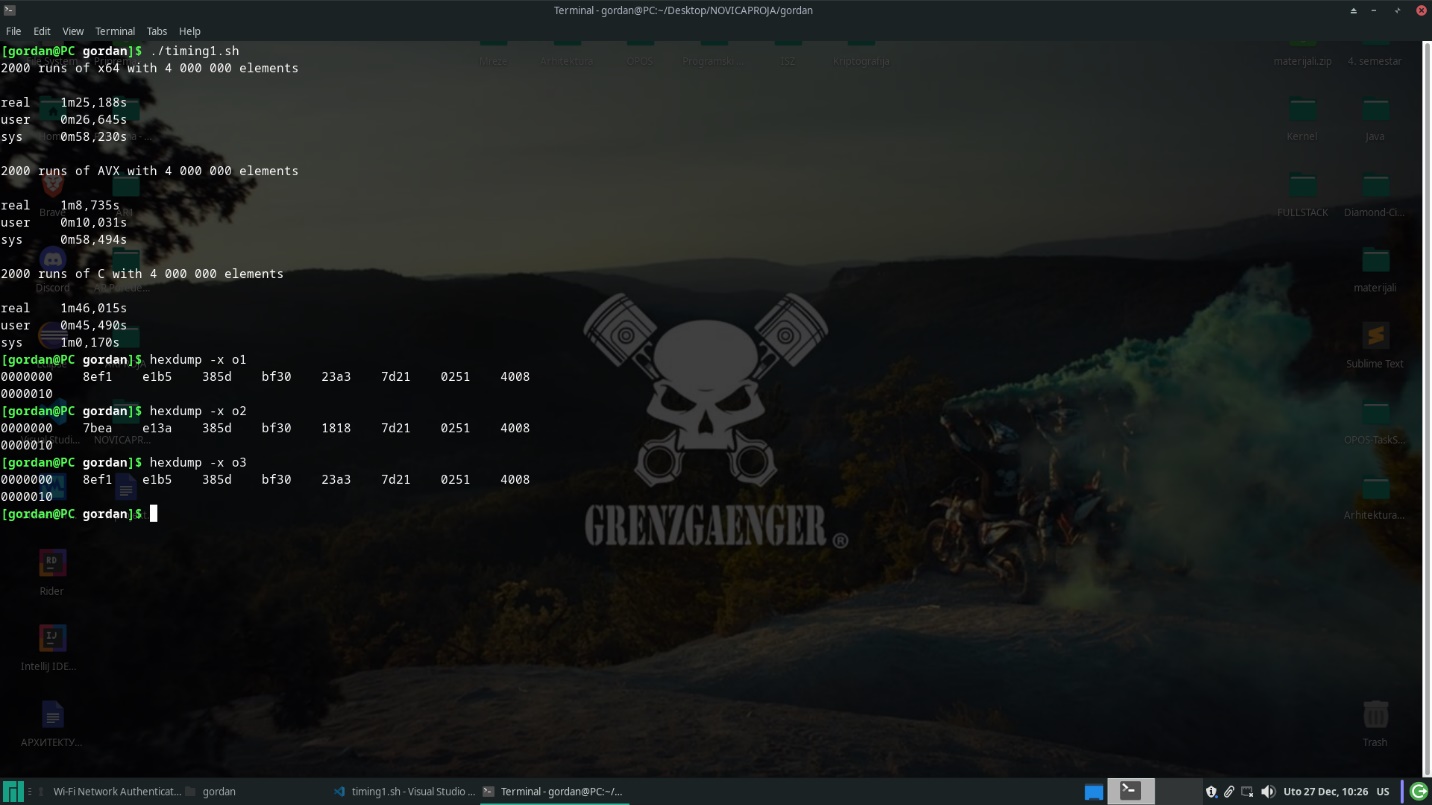
Поређење времена извршавања ћу представити помоћу screenshot-ова, на којима се јасно виде разлике у мјерењима извршавања, као и о томе да битних одступања у резултатима извршавања. Поред тога, приложио сам и документоване податке за float, о чијим проблемима ћу причати у закључку овог документа.



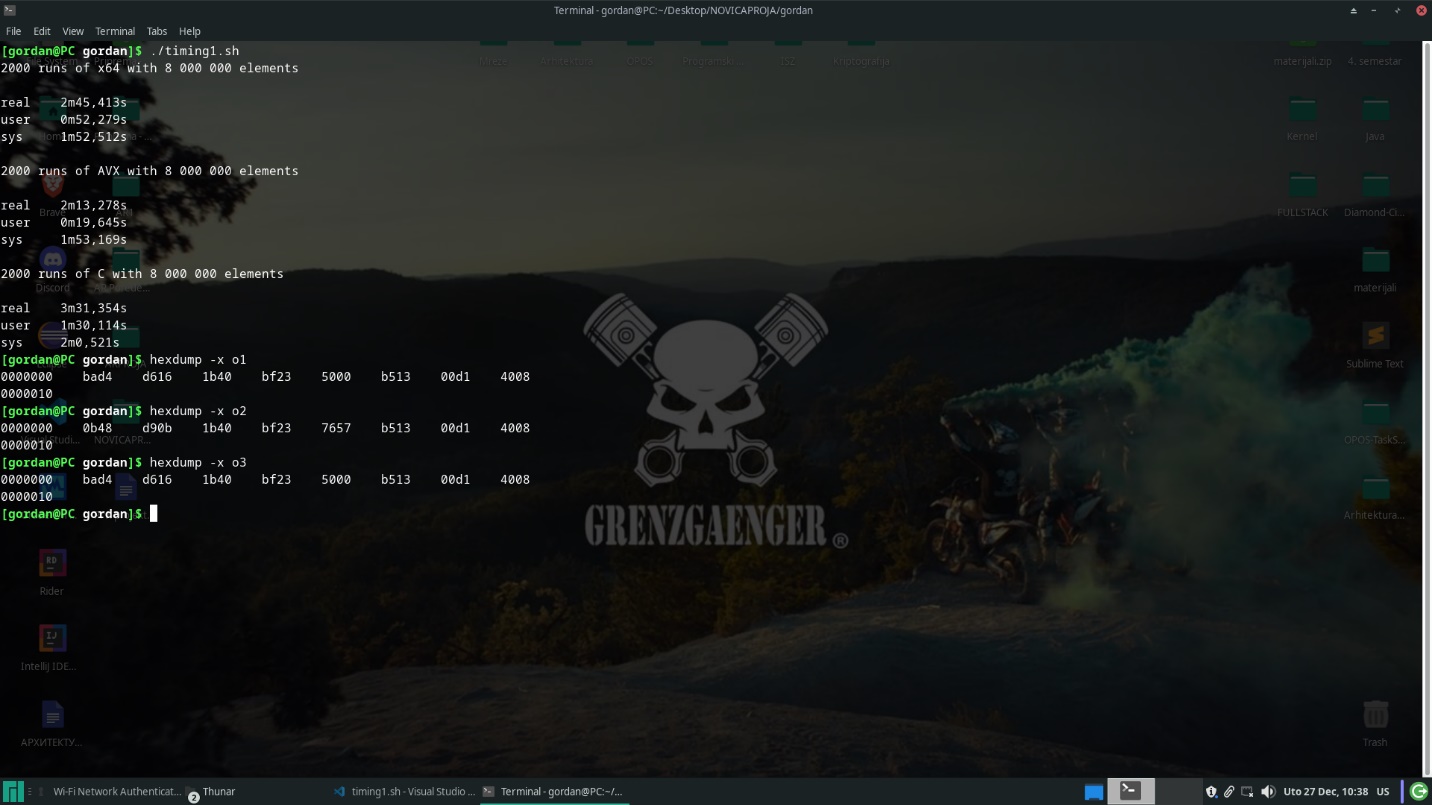
1-10 000 извршавања са 1 000 000 елемената



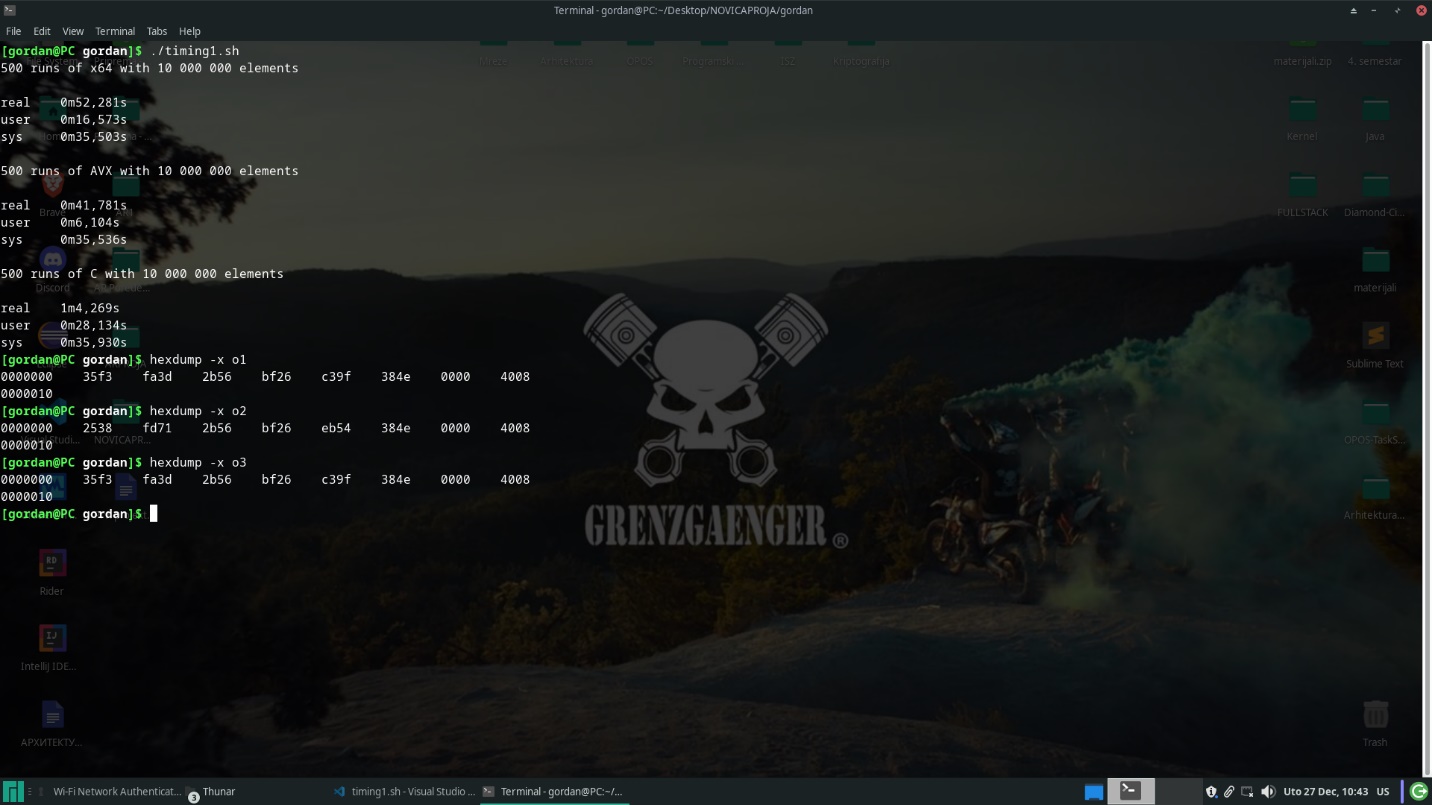
2-2 000 извршавања са 2 000 000 елемената



3-2 000 извршавања са 4 000 000 елемената



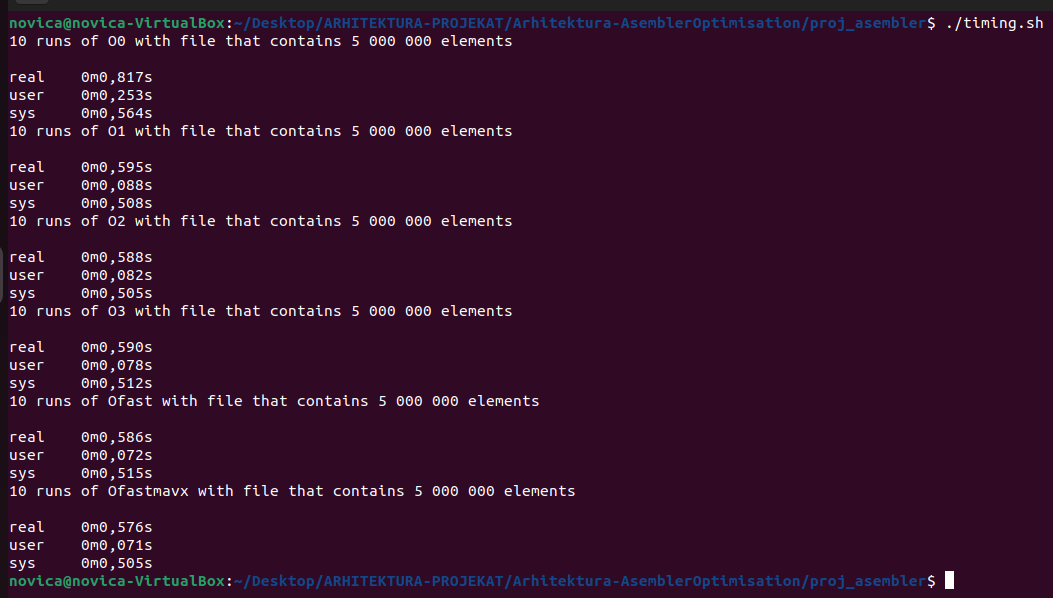
4-2 000 извршавања са 8 000 000 елемената



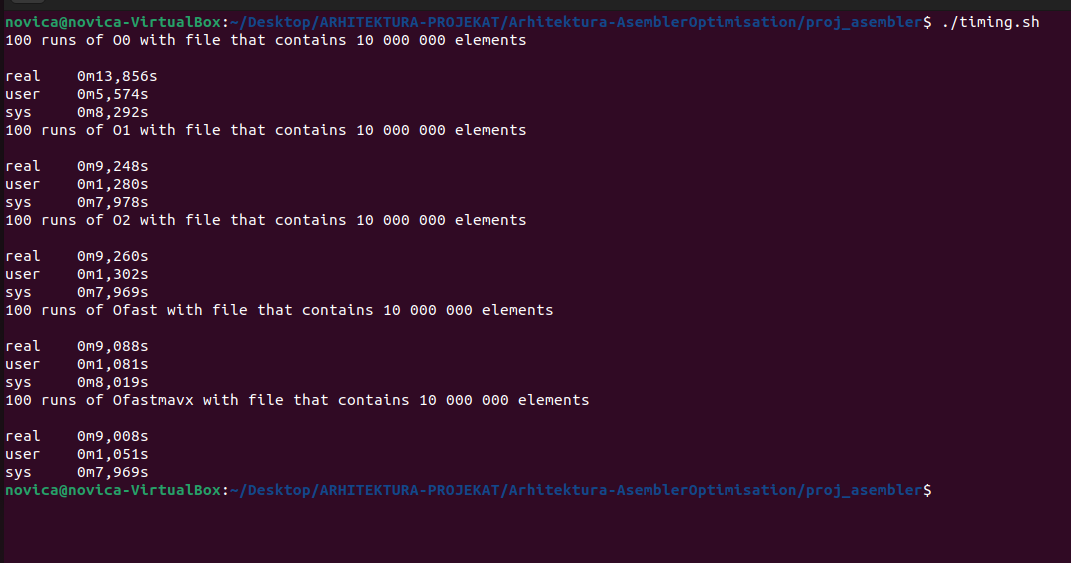
5-500 извршавања са 10 000 000 елемената

# Поређење времена извршавања – Оптимизације gcc компајлера

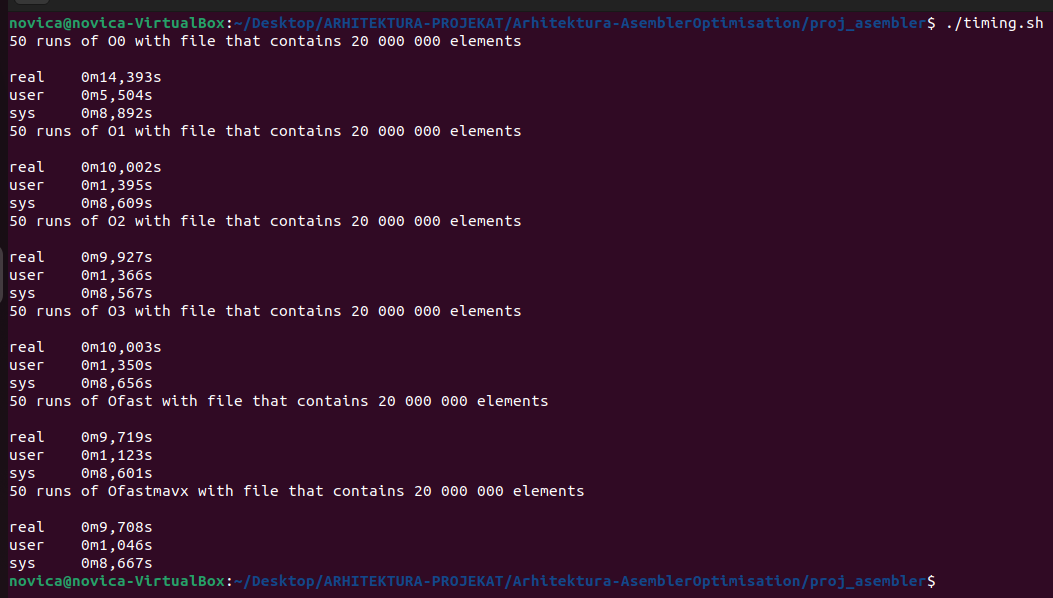
За поређење времена извршавање приложићу релевантне screenshot-ове. На њима се виде које су компајлерске оптимизације кориштене, као и број извршавања програма и времена за које су се одговарајући програми извршили. У закључку документа ћу се детаљније осврнути на резултате поређења времена извршавања.



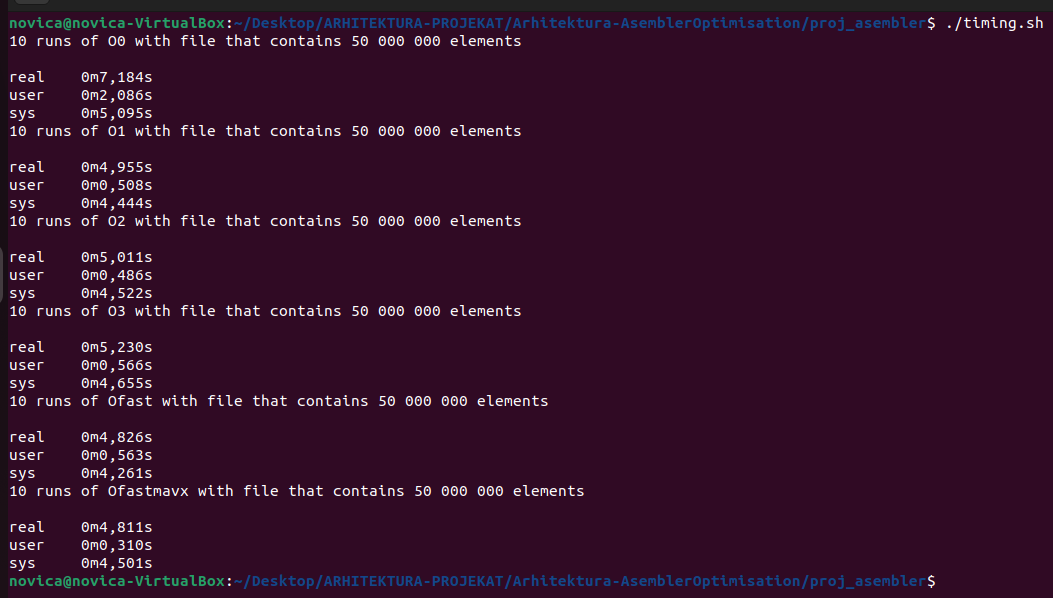
Примјер бр. 1



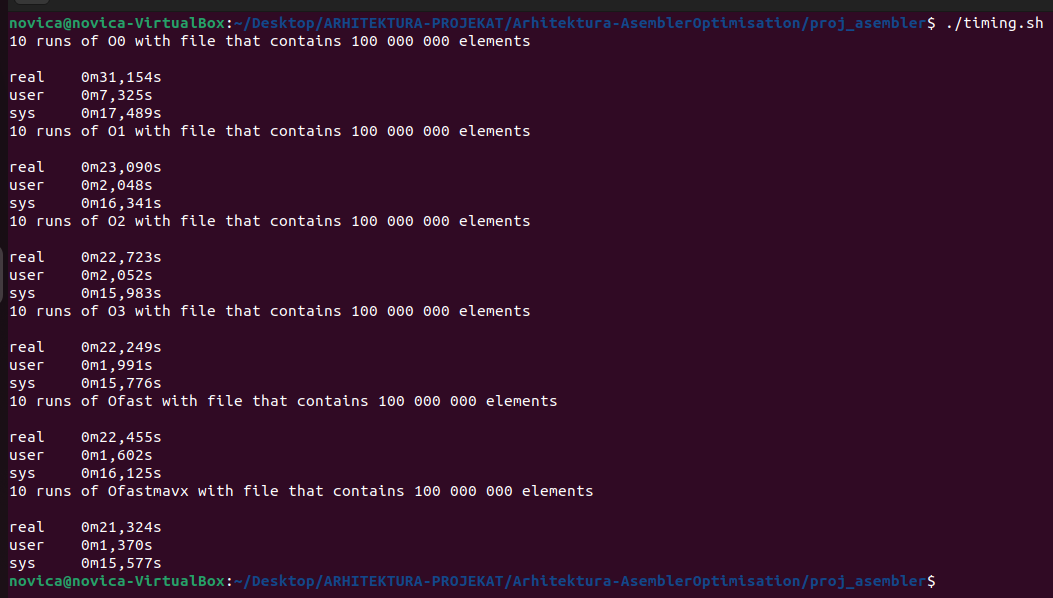
Примјер бр. 2



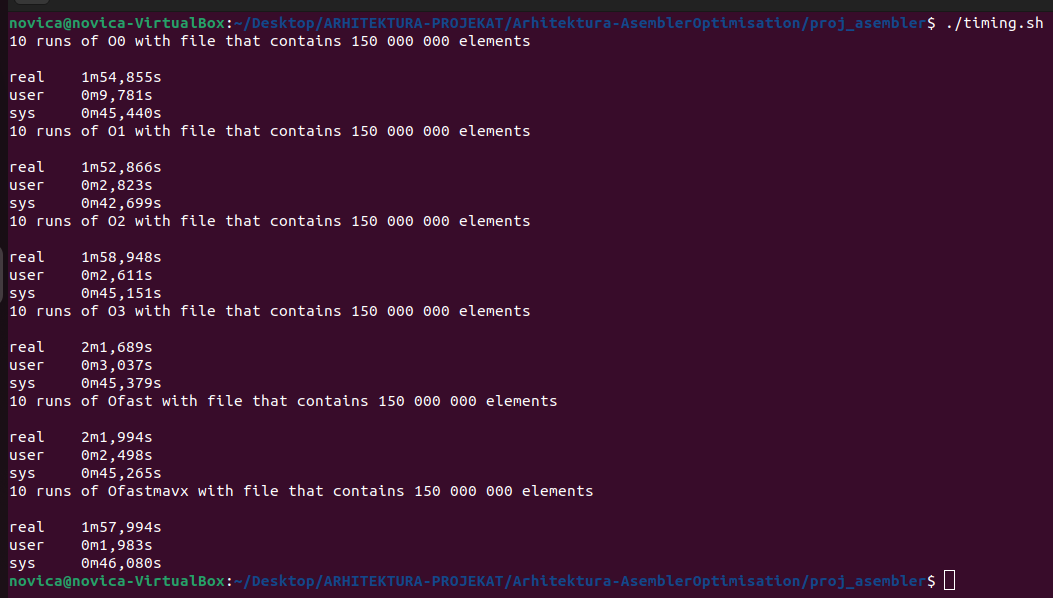
Примјер бр. 3



Примјер бр. 4



Примјер бр. 5

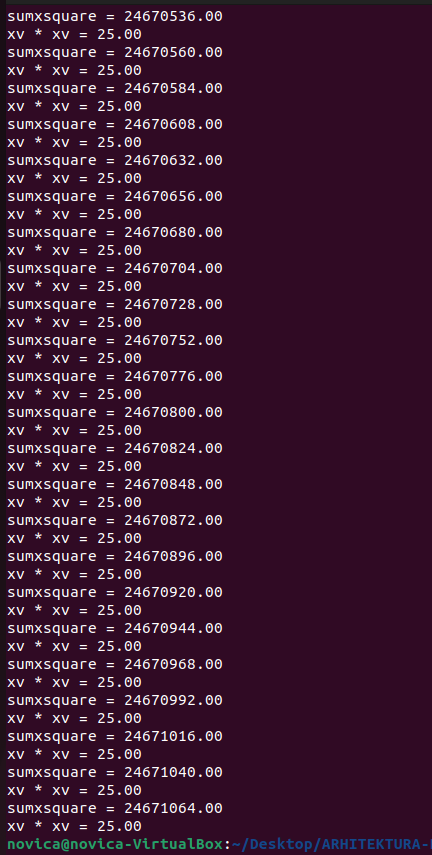


Примјер бр. 6

# Закључак

Као прву ставку ћу напоменути да сам првобитно реализовао асемблерске програме и C програм који је радио са floating point типовима података, али је то довело до одређених проблема. Наиме, floating point са рачунањем велике количине улазних података (као примјер можемо узети 1 000 000 улазних података гдје имамо случајне бројеве у распону од 1.0 до 100.0) доводи до тога да након одређеног броја рачунања floating point више не може адекватно рачунати податке, и при томе немамо одговарајуће резултате и прецизност. Упркос томе, документовао сам floating point и на другој слици се јасно види да резултати бјеже. Да бих дошао до овог закључка, ставио сам 1 000 000 улазних података гдје сам генерисао 1 000 000 података које имају вриједност 5.0, а 1 000 000 које имају вриједност 6.0 и видио да оптимизовани код ради како треба, а неоптимизовани, као и C програм, доводе до грешке.

На screenshot-у испод је приказано како C програм надодаје 24.0 умјесто 25.0, иако се јасно види да је израчунато 25.0.



Након тога сам програм написао помоћу AVX инструкцијског скупа и као што је очекивано, није било проблема са сумирањем велике количине улазних података. Проблем који сам имао при оптимизацији је да се код YMM регистара не може користити shuffle, па сам то морао симулирати на свој начин. Упркос томе, убрзања су документована и програм ради како је очекивано. C програм је најспорији, што је и очекивано, и при компајлирању нису кориштене никакве оптимизације. Што је већа количина улазних података и дуже вријеме извршавања, осјетиће се већа разлика између оптимизованог и неоптимизованог кода.

Као закључак се може истаћи да, иако рад са floating point подацима није довољно поуздан, доноси боље перформансе и резултате јер се може користит shuffle, а иначе код ymm регистара морамо радити непотребна копирања или помоћу помоћних промјенљивих или евентуално претварати packed double у packed single податке, па користити shuffle, па враћати назад у packed double податке.

Када погледамо C компајлерске оптимизације и релевантне приложене screenshot-ове, можемо видјети којим редослиједом извршавања и које компајлерске оптимизације сам користио. Најспорије вријеме извршавања је очекивано за -O0 компајлерске оптимизације, а најбрже за Ofast, mavx2. Битно је примијетити да -O1, -O2 и -O3 компајлерске оптимизације дају готово идентично вријеме извршавања. Једино неочекивано одступање је рад са 150 000 000 елемената гдје Ofast mavx2 гдје -O1 даје најбоље резулате, а -O0 је одмах након њега по питању добрих перформанси.

Потребно је истаћи да проблем може представљати и количина улазних података са којом се ради, тако да треба бити опрезан и са становишта погледа отварања, копирања и генерално рада са фајловима, мада тестови не одступају од основне идеје, а то је да оптимизације значајно доприносе побољшању перфоманси и времена извршавања, како и асемблерског програма, тако и C програма компајлираних помоћу различитих компајлерских оптимизација.