**Proiect Embedded Computing**

**Correlating branch predictors**

Student:

Paros Beatrice-Elena

Table of Contents

[1. Correlating branch predictors 3](#_Toc187764583)

[1.1 Concepte cheie 3](#_Toc187764584)

[1.1.1 Predictia simpla a ramurilor 3](#_Toc187764585)

[1.1.2 Corelarea intre ramuri 3](#_Toc187764586)

[1.1.3 Baza predictiei corelate 3](#_Toc187764587)

[1.1.3.1 Tabele de istoric global 3](#_Toc187764588)

[1.1.3.2 Tabele de istoric al ramurii 3](#_Toc187764589)

[1.1.4 Performanta si eficienta 3](#_Toc187764590)

[2. Rezolvarea temei de proiect 4](#_Toc187764591)

[2.1 Creearea folderului principal si a subfolderului de lucru 4](#_Toc187764592)

[2.2 Crearea subfolderelor suplimentare 4](#_Toc187764593)

[2.3 Descrierea folderelor si continutul acestora 4](#_Toc187764594)

[2.4 Parcurgerea scriptului 9](#_Toc187764595)

[2.5 Functia hook\_branch\_predict 13](#_Toc187764596)

[2.5.1 Explicatii functie 14](#_Toc187764597)

[2.6 Rularea comenzii 15](#_Toc187764598)

[3. Concluzii si dezvoltari ulterioare 19](#_Toc187764599)

[3.1 Concluzii 19](#_Toc187764600)

[3.2 Dezvoltari ulterioare 19](#_Toc187764601)

[4. Probleme intampinate 19](#_Toc187764602)

# Correlating branch predictors

Correlating branch predictors sau Predictoarele de ramuri corelate reprezinta un tip avansat de mechanism de predictie a ramurilor, utilizat pentru a imbunatati acuratetea predictiei ramurilor in timpul executiei programelor.

## 1.1 Concepte cheie

## 1.1.1 Predictia simpla a ramurilor

In procesoarele moderne, atunci cand o ramura e conditionata de o instructiune, procesorul trebuie sa decida in ce directie sa mearga. In cazul in care procesorul nu poate ghici corect, este posibil sa se produca o intarziere.

## 1.1.2 Corelarea intre ramuri

Predictoarele de ramuri corelate utilizeaza un istoric al comportamentului unor ramuri anterioare pentru a putea prezice directia unei ramuri curente.

Predictia se face atat pe baza starii actuale a ramurii dar si pe baza a ceea s-a intamplat anterior.

## 1.1.3 Baza predictiei corelate

### 1.1.3.1 Tabele de istoric global

Acestea memoreaza istoricul global al ramurilor, adica daca au fost sau nu luate ramurile anterioare.

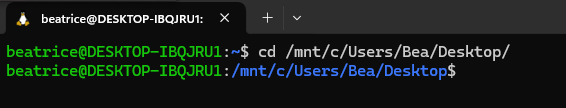
### 1.1.3.2 Tabele de istoric al ramurii

Acestea stocheaza informatiile istorice pentru ramuri specific, adica comportamentul trecut al unei anumite ramuri.

## 1.1.4 Performanta si eficienta

# 2. Rezolvarea temei de proiect

Proiectul a fost realizat folosind tool-ul WSL (windows subsystem for linux) cu ajutorul liniilor de comanda.



## 2.1 Creearea folderului principal si a subfolderului de lucru

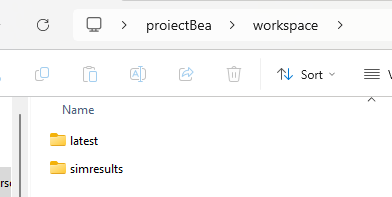
Am inceput prin crearea unui folder pe desktop denumit "proiectBea", folosind comanda mkdir proiectBea. Acesta a fost locul in care am adunat toate fisierele si documentele legate de proiect. In interiorul folderului "proiectBea", am creat un subfolder denumit "workspace".

## 2.2 Crearea subfolderelor suplimentare

In cadrul folderului "workspace", am mai creat doua foldere suplimentare pentru a organiza mai bine resursele proiectului:

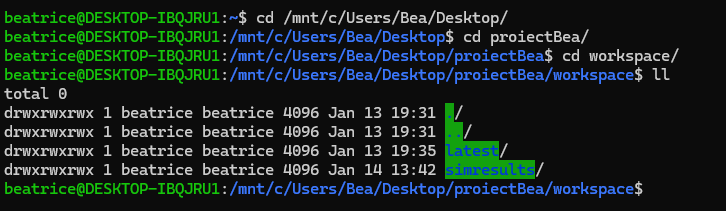
"latest" – Acest folder contine cele mai recente versiuni ale fisierelor si codurilor dezvoltate.

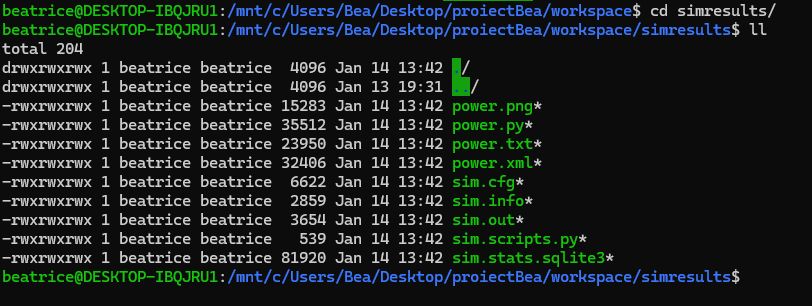
"snipersim" – in interior acestui folder avem rezultatele simularii



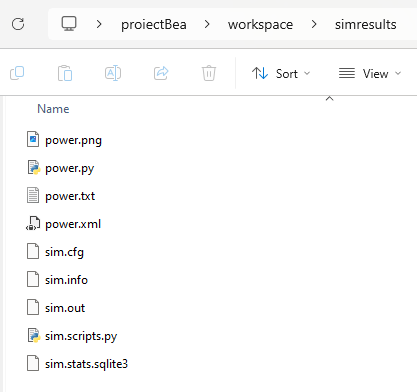
## 2.3 Descrierea folderelor si continutul acestora

Acum, daca intram in terminal si folosim comanda ll, putem vedea cele doua foldere "latest" si "snipersim" in cadrul folderului "workspace". Comanda ll afiseaza o lista detaliata a fisierelor si directoarelor din directorul curent.

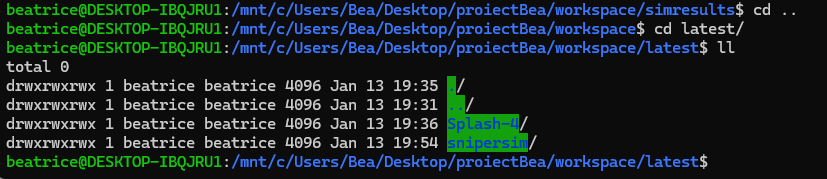
In interiorul folderului simresults se regasesc fisiere corespunzatoare rezultatelor simularii



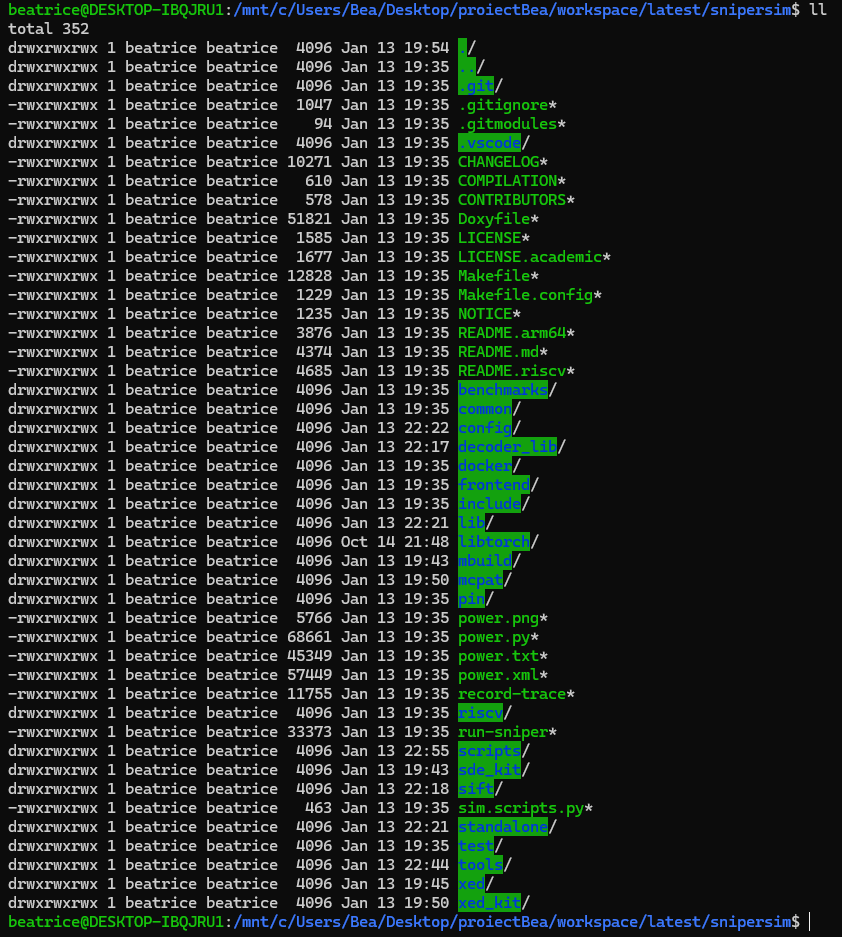
Ne intereseaza in principiu fisierul sim.out din interiorul folderului simresults, deoarece acesta contine rezultatul simularii noastre, pe care il putem vizualiza in acel folder.



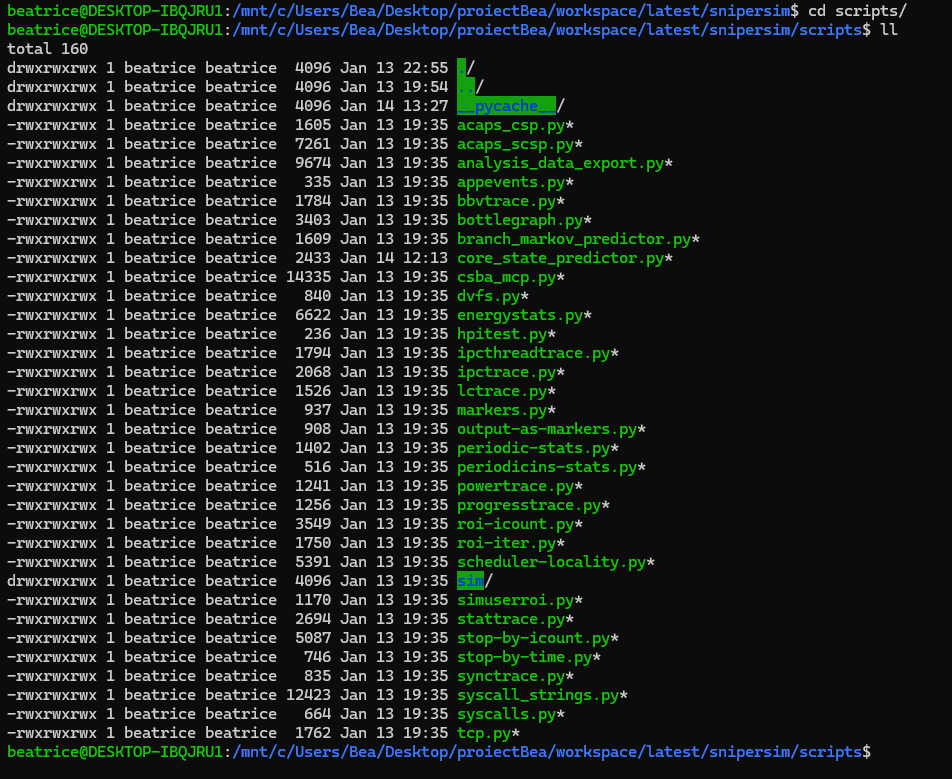
Daca ne intoarcem in folderul latest putem observa alte 2 foldere snipersim si splash-4.



In interiorul folderului snipersim se afla mai multe fisiere, printre care si scriptul care ruleaza simulatorul sniper.



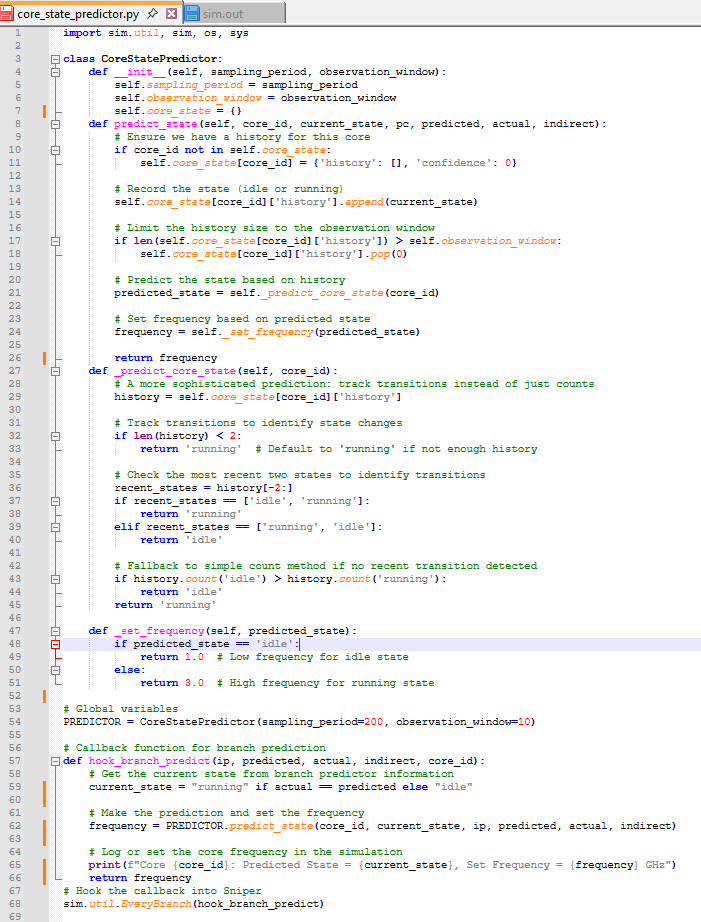
Pe langa acest script, se observa si folder-ul care contine scripturi.



In acest folder am creat un script numit core\_state\_predictor.py. Putem vizualiza scriptul folosind comanda cat core\_state\_predictor.py



O alta modalitate de a vizualiza codul este sa navigam prin directoare pana ajungem la scriptul dorit, pe care il deschidem cu Notepad.

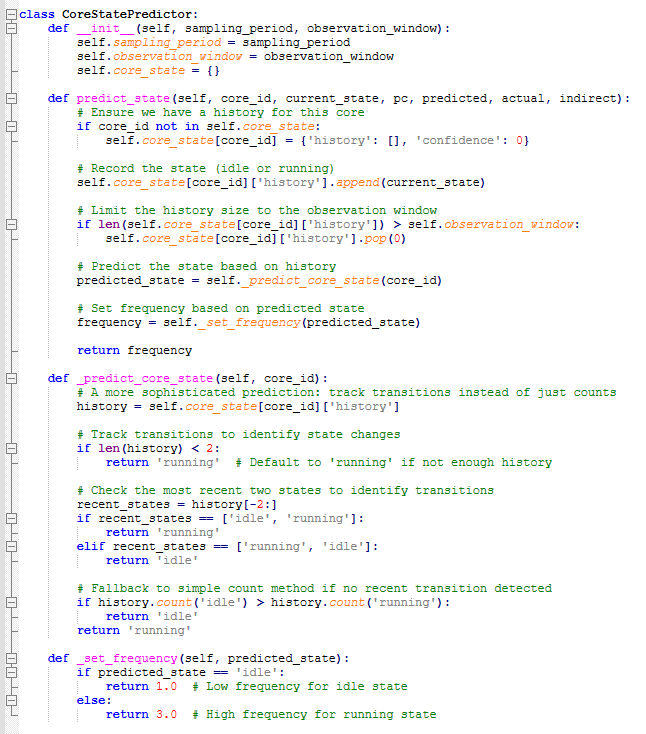


## 2.4 Parcurgerea scriptului

In prima parte a scriptului am importat modulele necesare pentru simulare (sim si sim.util) si pentru manipularea sistemului de operare si a fișierelor (os si sys).



Apoi am creat o clasa numita CoreStatePredictor in cadrul careia avem un constructor (\_\_init\_\_) si metode (predict\_state, \_predict\_core\_state, \_set\_frequency).



### 2.4.1 Constructorul clasei

Am inceput cu constructorul clasei:



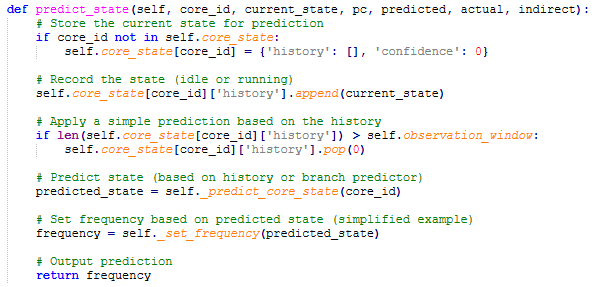
sampling\_period: perioada de esantionare (intervalul in care se face predictia).

observation\_window: dimensiunea ferestrei de observatie, numarul de stari anterioare ce sunt luate in considerare pentru predictia viitoare a starii nucleului.

core\_state: un dictionar ce stocheaza istoricul starilor fiecarui nucleu (core\_id), folosit pentru a face predictii.

### 2.4.2 Functia predict\_state

Mai departe avem functia predict\_state. Aceasta functie preia informatiile despre starea curenta a unui nucleu (core\_id) si foloseste istoricul starilor pentru a prezice starea viitoare si frecventa corecta a nucleului.



Parametrii:

core\_id: ID-ul nucleului.

current\_state: Starea curenta a nucleului (de exemplu, „idle” sau „running”).

pc: Adresa din registrul Program Counter

predicted si actual: Rezultatele predictiei ramurilor si starea reala

indirect: Informatie despre ramura, daca este indirecta

#### 2.4.2.1 Explicarea functiei

In continuare vom discuta functia:

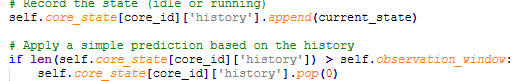


Daca nucleul specificat nu are deja un istoric, se initializeaza o intrare în dictionar pentru acel nucleu cu:

'history': o lista pentru starile anterioare.

'confidence': Un contor al increderii in predictii (nu este utilizat in intregime in codul actual).

Actualizarea istoricului



append(current\_state): adauga starea curenta la istoric

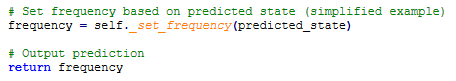
Daca lungimea istoricului este mai mare decat observation\_window, se elimiaă cea mai veche stare pentru a mentine dimensiunea fixa

Predictia starii



\_predict\_core\_state: este o metoda privata care alineaza istoricul starilor pentru a prezice urmatoarea stare

Setarea frecventei

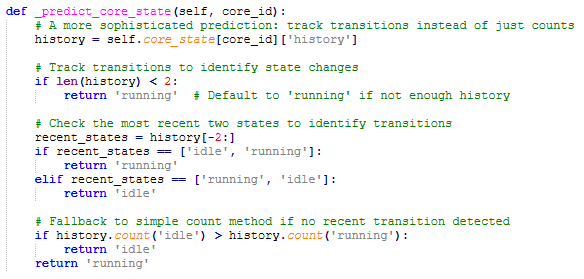


\_set\_frequency: determina frecventa procesorului bazat pe starea prezisa (frecventa scazuta pentru „idle” si mare pentru „running”).

Returneaza frecventa ca rezultat al predictiei.

### 2.4.3 Metoda \_ predict\_core\_state

In continuare am implementat o metoda private numita \_predict\_core\_state



Este o metoda privata a clasei CoreStatePredictor, iar scopul sau este de a prezice starea unui nucleu pe baza istoricului starii acestuia. Parametrul core\_id este folosit pentru a identifica nucleul specific pentru care se face predicția.Implementare metoda \_set\_frequency

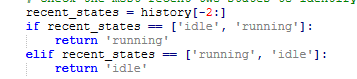
#### 2.4.3.1 Analiza metodei



Linia acceseaza istoricul starii pentru nucleul specificat de core\_id. In self.core\_state, pentru fiecare core\_id, avem o intrare care include un history, adica o lista care contine starile anterioare ale nucleului



Aici se verifica daca istoricul contine mai putin de doua stari. Daca istoricul este prea scurt pentru a face o predicție relevanta (adica nu sunt suficiente date pentru a analiza tranzitii), atunci metoda returneaza running ca stare prezisa



Din linia recent\_states = history[-2:] se extrag ultimele 2 stari din istoric care se salveaza in variabila recent\_state.

Mai departe metoda utilizeaza ultimele 2 stari pentru a detecta tranzitii:

if recent\_states == ['idle', 'running']:

return 'running'

* daca daca ultimele 2 stari sunt ['idle', 'running'], inseamna ca nucleul trece in running

elif recent\_states == ['running', 'idle']:

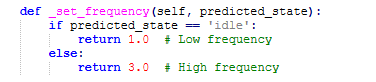
return 'idle'

* Daca daca ultimele 2 stari sunt ['running', 'idle'], inseamna ca nucleul trece in idle

In continuare, daca nu au fost detectate tranzitii clare, aceasta linie face o verificare simpla bazata pe frecventa starii idle si running in intregul istoric al nucleului:



### 2.4.4 Functia \_set\_frequency



predicted\_state: Starea prezisa.

Returneaza frecventa corespunzatoare:

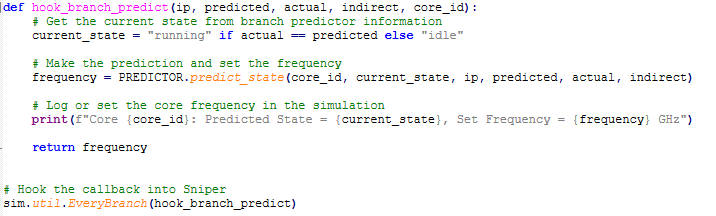
1.0 GHz pentru starea „idle”.

3.0 GHz pentru starea „running”.

### 2.4.5 Variabila globala



## 2.5 Functia hook\_branch\_predict



Aceasta este o functie de callback care este apelata in fiecare moment cand simulatorul (Sniper) proceseaza o ramura (branch) in timpul executiei unui program

### 2.5.1 Explicatii functie



Aceats alinie de cod determina starea curenta prin compararea valorii prezise (predicted) cu vacloarea reala (actual):

Daca valorile coincid (actual == predicted), nucleul este considerat activ ("running").

Daca valorile nu coincid, nucleul este considerat inactiv ("idle").



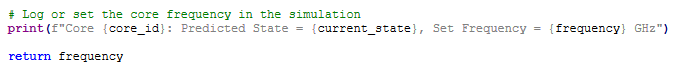
Mai departe se predictioneaza starea nucleului si ajustarea freecventei folosind metoda predict\_state a clasei CoreStatePredictor. Aceatsa este apelata cu urmatorii parametrii:

core\_id: ID-ul nucleului pentru care se face predictia.

current\_state: Starea curenta a nucleului (determinata anterior).

ip, predicted, actual, indirect: Parametrii suplimentari legati de instructiunea curenta.

Rezultatul este o valoare de frecventa (GHz) stabilita pe baza predictiei starii nucleului (inactiv/activ).



Se afiseaza resultatul si se returneaza frecventa, care va fi utilizata de simulator pentru a ajusta comportamentul nucleului.



Aici se realizeaza legarea functiei de simulator.

sim.util.EveryBranch este o functie a simulatorului Sniper care permite conectarea unui callback la evenimentele de predictie de branch.

hook\_branch\_predict este functia callback care va fi apelata de fiecare data cand simulatorul detecteaza o bifurcatie in fluxul de executie al procesorului.

Functia hook\_branch\_predict determina starea curenta a unui nucleu pe baza comportamentului predictiei de branch si utilizeaza clasa CoreStatePredictor pentru a ajusta frecventa nucleului. Acest lucru permite economisirea de energie si optimizarea performantei in functie de utilizarea procesorului.

## 2.6 Rularea comenzii

Daca rulam comanda

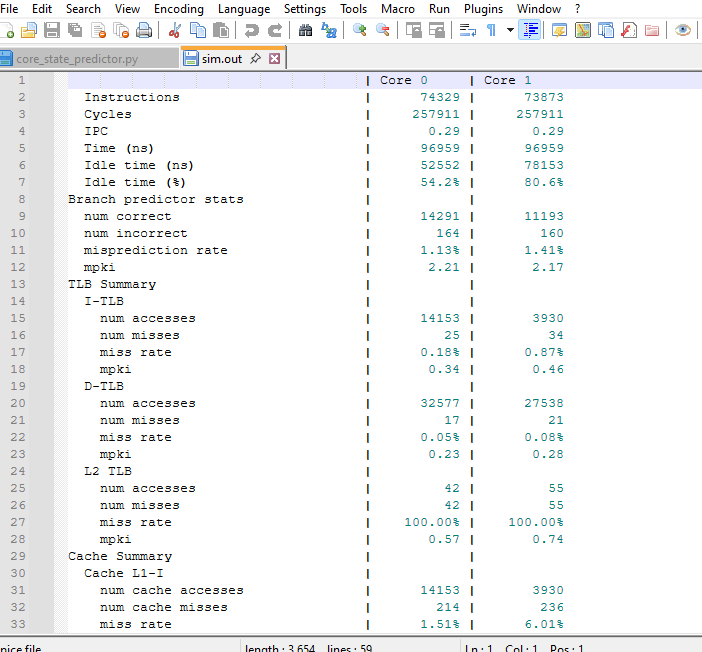
./run-sniper -v -n 2 -c gainestown --roi -s core\_state\_predictor --power -d /mnt/c/Users/Bea/Desktop/proiectBea/workspace/simresults -- /mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4/Splash-4/lu-contiguous\_blocks/LU-CONT -n32 -p2 -b8

Vom avea un rezultat care se gaseste in folderul simresults.

Explicatie linie de comanda:

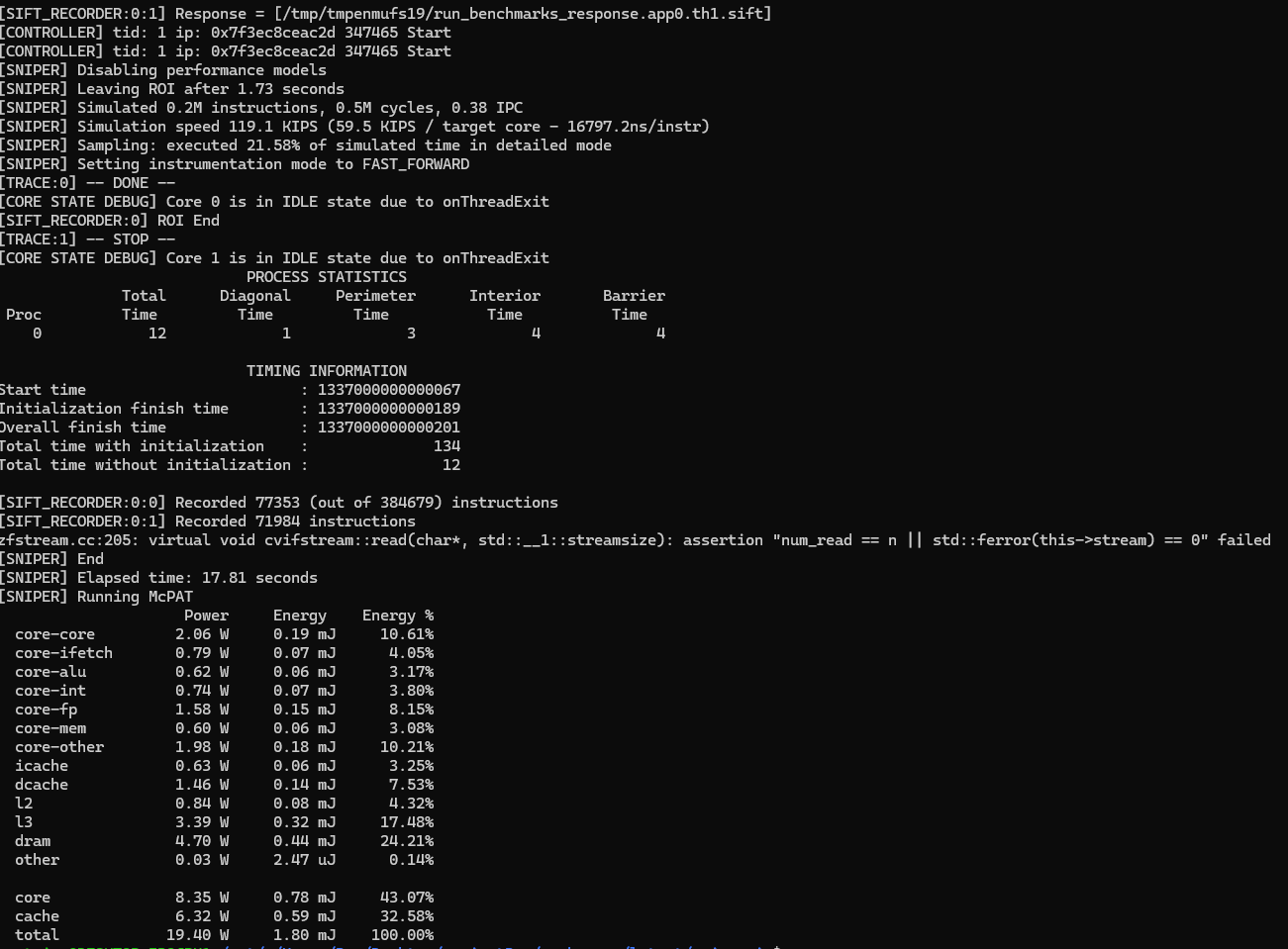
* Scriptul principal este ./run-sniper care lanseaza in executie simulatorul sniper
* -v inseamna ca se activeaza modul verbise adica primim mai multe informatii in timpul rularii
* -n 2 inseamna ca se lanseaza 2 fire de executie/coruri
* -c gainestown specifica configuratia pentru simulator
* --roi activeaza regiunea de interes, adica permite simulatorului sa se concentreze pe anumite sectiuni ale programului
* -s core\_state\_predictor specifica faptul ca se va folosi scriptul core\_state\_predictor
* --power activeaza monitorizarea si analiza consumului de putere
* -d /mnt/c/Users/Bea/Desktop/proiectBea/workspace/simresults indica directorul de output in care se stocheaza informatia
* -- este o delimitare
* /mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4/Splash-4/lu-contiguous\_blocks/LU-CONT este calea catre programul care va fi rulat
* -n32 este un parametru care indica numarul de fire de executie
* -p specifica dimensiunea blocurilor
* -b8 parametru care controloeaza dimensiunea blocurilor de date sau memorie

Mai jos se poate observa cum arata fisierul sim.out in interior

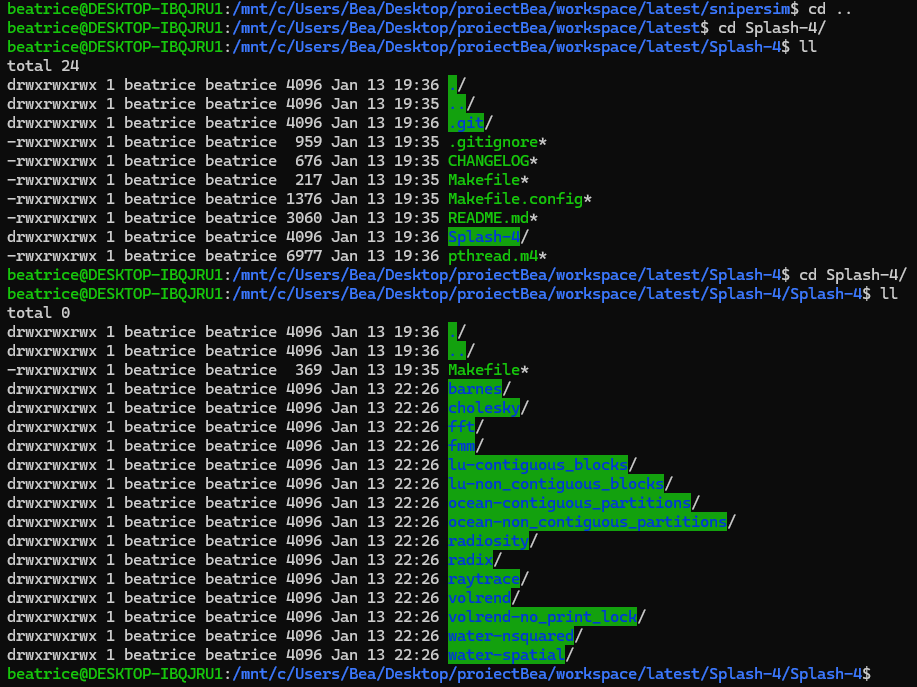


Acest fisier contine rezultatele simularii unui program pe doua nuclee de procesor (Core 0 si Core 1), oferind informatii despre instructiuni executate, cicluri, performanta predictorului de branch, rata de ratari ale cache-urilor si accesari la memorie.

Se evidentiaza o utilizare mai scazuta a Core 1, cu un timp inactiv semnificativ mai mare comparativ cu Core 0, ceea ce sugereaza o performanta mai slaba. De asemenea, sunt prezentate statistici detaliate despre accesarile TLB (memorie cache specializata utilizata pentru a stoca adresele de memorie virtuala si corespondenta acestora cu adresele de memorie fizica), cache-uri si DRAM (tip de memorie principala utilizata in majoritatea sistemelor informatice pentru stocarea temporara a datelor.), indicand posibile zone de imbunatatire in gestionarea memoriei.



In interiorul folderului Spalsh-4:



Acest folder este folosit pentru a simula sarcini reale multi-thread. Benchmark-urile (LU, FFT) reprezinta diferite tipare de calcul si comunicatie care ne ajuta sa evaluam capacitatea predictorului de stare a nucleului sa optimizeze energia si performanta in conditii realiste.

Benchmark-ul LU masoara performanta implementarii algoritmului, adica timpul necesar. FFT (Fast Fourier Transform) masoara performanta transformatei rapide, inclusive timpul necesar efectuarii calculrlor.

# 3. Concluzii si dezvoltari ulterioare

## 3.1 Concluzii

In cadrul acestui proiect, nu am reusit sa implementez intregul predictor al starii nucleului procesorului, dar am reusit sa utilizez un script care m-a ajutat sa obtin anumite rezultate in simularea comportamentului procesorului. Desi implementarea nu a fost finalizata complet, acest pas a oferit o buna baza pentru viitoare imbunatatiri si teste, iar rezultatele obtinute au demonstrat potentialul de optimizare a consumului de energie in cadrul aplicatiilor de calcul intensiv.

## 3.2 Dezvoltari ulterioare

O dezvolate ulterioara a proiectului ar putea fi implementarea unui mediu izolat de dezvoltare si testare folosind Docker, care ar permite replicarea mediului de simulare intr-un mod mai usor si mai portabil.

# 4. Probleme intampinate

In timpul implementarii si testarii simulatorului Sniper, am intampinat cateva dificultati legate de configurarea mediului de dezvoltare pe sistemul de operare Ubuntu. Unul dintre obstacolele majore a fost compatibilitatea versiunilor Python.

Initial, am incercat sa folosesc Python 3, dar am descoperit ca versiunea existenta pe sistemul meu nu era suficient de noua pentru a satisface cerintele aplicatiei. Fiindca nu am putut sa actualizez versiunea Python pe Ubuntu, am decis sa trec la o masina virtuala (WSL), unde am instalat o versiune mai recenta de Python, care era compatibila cu cerintele Sniper si ale scripturilor de simulare. Aceasta abordare a rezolvat problema si a permis continuarea dezvoltarii si testarii predictorului de stare pe platforma corespunzatoare.