Proiect Embedded Computing
Correlating branch predictors

Student:
Paros Beatrice-Elena

Table of Contents

1. Correlating branch predictors 3
1.1 Concepte cheie 3
1.1.1 Predictia simpla a ramurilor 3
1.1.2 Corelarea intre ramuri 3
1.1.3 Baza predictiei corelate 3
1.1.3.1 Tabele de istoric global 3
1.1.3.2 Tabele de istoric al ramurii 3
1.1.4 Performanta si eficienta 3
2. Rezolvarea temei de proiect 4
2.1 Creearea folderului principal si a subfolderului de lucru 4
2.2 Crearea subfolderelor suplimentare 4
2.3 Descrierea folderelor si continutul acestora 4
2.4 Parcurgerea scriptului 10
2.5 Functia hook_branch_predict 16
2.5.1 Explicatii functie 16
2.6 Rularea comenzii 17
3. Concluzii si dezvoltari ulterioare 21
3.1 Concluzii 21
3.2 Dezvoltari ulterioare 21
4. Probleme intampinate 21

1. 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.

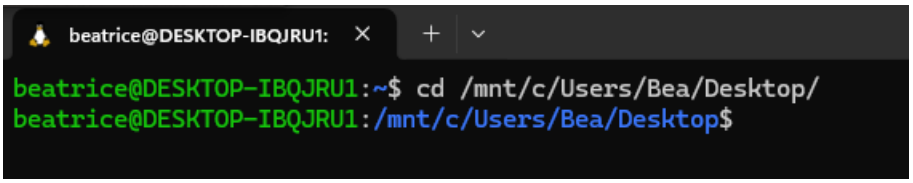
1.1.3.2 Tabele de istoric al ramurii

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 Crearea folderului principal si a subfolderului de lucru

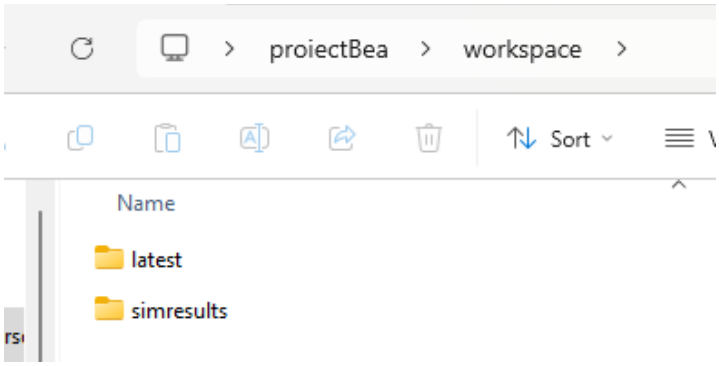
Am inceput prin crearea unui folder pe desktop denumit "proiectBea", folosind comanda mkdir proiectBea. Acesta a fost locul in care am adunat toate fisierele si documentele legate de proiect. In interiorul folderului "proiectBea", am creat un subfolder denumit "workspace".

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

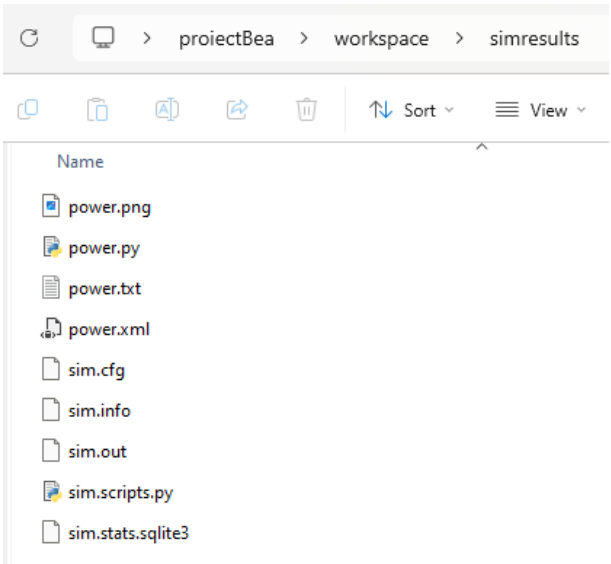
Acum, daca intram in terminal si folosim comanda ll, putem vedea cele doua foldere "latest" si "snipersim" in cadrul folderului "workspace". Comanda ll afiseaza o lista detaliata a fisierelor si directoarelor din directorul curent.

```
beatrice@DESKTOP-IBQJRU1:~$ cd /mnt/c/Users/Bea/Desktop/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop$ cd proiectBea/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea$ cd workspace/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace$ ll
total 0
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:31 ./
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:31 ../
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:35 latest/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 14 13:42 simresults/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace$
```

In interiorul folderului simresults se regasesc fisiere corespunzatoare rezultatelor simularii

```
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace$ cd simresults/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/simresults$ ll
total 204
drwxrwxrwx 1 beatrice beatrice  4096 Jan 14 13:42 ./
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:31 ../
-rwxrwxrwx 1 beatrice beatrice 15283 Jan 14 13:42 power.png*
-rwxrwxrwx 1 beatrice beatrice 35512 Jan 14 13:42 power.py*
-rwxrwxrwx 1 beatrice beatrice 23950 Jan 14 13:42 power.txt*
-rwxrwxrwx 1 beatrice beatrice 32406 Jan 14 13:42 power.xml*
-rwxrwxrwx 1 beatrice beatrice  6622 Jan 14 13:42 sim.cfg*
-rwxrwxrwx 1 beatrice beatrice  2859 Jan 14 13:42 sim.info*
-rwxrwxrwx 1 beatrice beatrice  3654 Jan 14 13:42 sim.out*
-rwxrwxrwx 1 beatrice beatrice   539 Jan 14 13:42 sim.scripts.py*
-rwxrwxrwx 1 beatrice beatrice 81920 Jan 14 13:42 sim.stats.sqlite3*
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/simresults$
```

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.



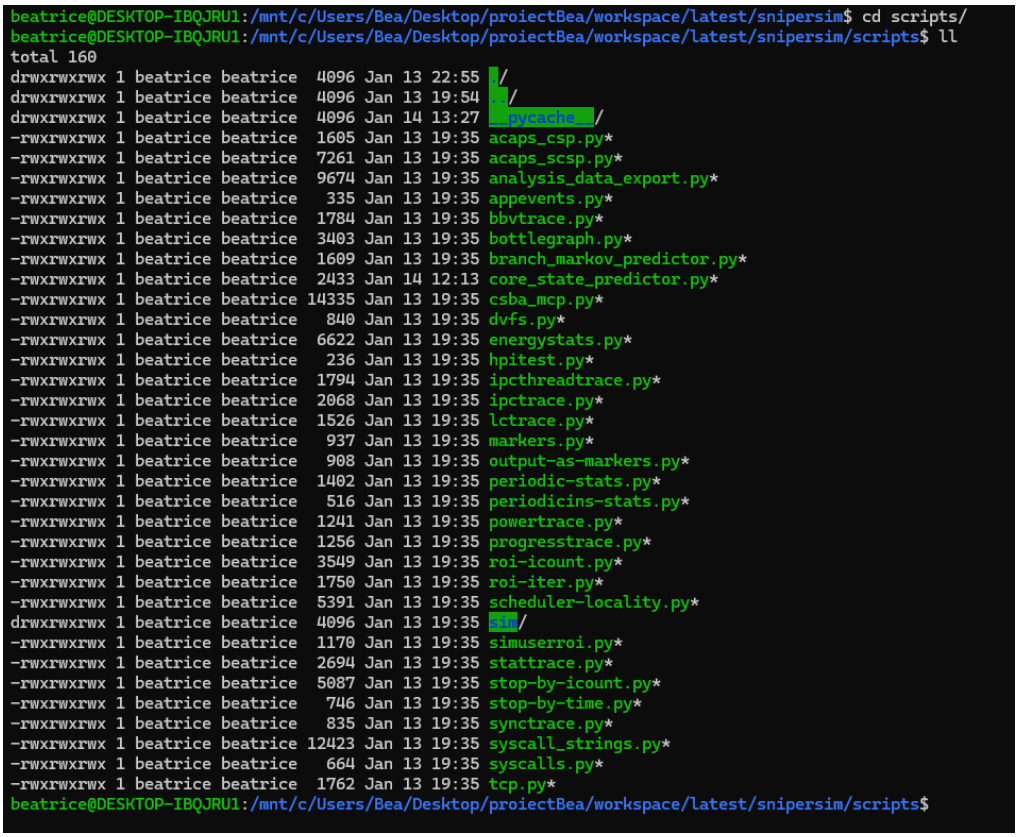
Daca ne intoarcem in folderul latest putem observa alte 2 foldere snipersim si splash-4.

```
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/simresults$ cd ..
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace$ cd latest/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest$ ll
total 0
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:35 ./
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:31 ../
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 Splash-4/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:54 snipersim/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest$
```

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

```
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/snipersim$ ll
total 352
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:54 ./
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 ../
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 .git/
-rwxrwxrwx 1 beatrice beatrice  1047 Jan 13 19:35 .gitignore*
-rwxrwxrwx 1 beatrice beatrice    94 Jan 13 19:35 .gitmodules*
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 .vscode/
-rwxrwxrwx 1 beatrice beatrice 10271 Jan 13 19:35 CHANGELOG*
-rwxrwxrwx 1 beatrice beatrice   610 Jan 13 19:35 COMPILATION*
-rwxrwxrwx 1 beatrice beatrice   578 Jan 13 19:35 CONTRIBUTORS*
-rwxrwxrwx 1 beatrice beatrice 51821 Jan 13 19:35 Doxyfile*
-rwxrwxrwx 1 beatrice beatrice  1585 Jan 13 19:35 LICENSE*
-rwxrwxrwx 1 beatrice beatrice  1677 Jan 13 19:35 LICENSE.academic*
-rwxrwxrwx 1 beatrice beatrice 12828 Jan 13 19:35 Makefile*
-rwxrwxrwx 1 beatrice beatrice  1229 Jan 13 19:35 Makefile.config*
-rwxrwxrwx 1 beatrice beatrice  1235 Jan 13 19:35 NOTICE*
-rwxrwxrwx 1 beatrice beatrice  3876 Jan 13 19:35 README.arm64*
-rwxrwxrwx 1 beatrice beatrice  4374 Jan 13 19:35 README.md*
-rwxrwxrwx 1 beatrice beatrice  4685 Jan 13 19:35 README.riscv*
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 benchmarks/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 common/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:22 config/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:17 decoder_lib/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 docker/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 frontend/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 include/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:21 lib/
drwxrwxrwx 1 beatrice beatrice  4096 Oct 14 21:48 libtorch/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:43 mbuild/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:50 mcpat/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 pin/
-rwxrwxrwx 1 beatrice beatrice  5766 Jan 13 19:35 power.png*
-rwxrwxrwx 1 beatrice beatrice 68661 Jan 13 19:35 power.py*
-rwxrwxrwx 1 beatrice beatrice 45349 Jan 13 19:35 power.txt*
-rwxrwxrwx 1 beatrice beatrice 57449 Jan 13 19:35 power.xml*
-rwxrwxrwx 1 beatrice beatrice 11755 Jan 13 19:35 record-trace*
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 riscv/
-rwxrwxrwx 1 beatrice beatrice 33373 Jan 13 19:35 run-sniper*
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:55 scripts/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:43 sde_kit/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:18 sift/
-rwxrwxrwx 1 beatrice beatrice   463 Jan 13 19:35 sim.scripts.py*
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:21 standalone/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:35 test/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 22:44 tools/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:45 xed/
drwxrwxrwx 1 beatrice beatrice  4096 Jan 13 19:50 xed_kit/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/snipersim$
```

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

```
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/snipersim/scripts$ cat core_state_predictor.py
import sim.util, sim, os, sys

class CoreStatePredictor:
    def __init__(self, sampling_period, observation_window):
        self.sampling_period = sampling_period
        self.observation_window = observation_window
        self.core_state = {}

    def predict_state(self, core_id, current_state, pc, predicted, actual, indirect):
        # Ensure we have a history for this core
        if core_id not in self.core_state:
            self.core_state[core_id] = {'history': [], 'confidence': 0}

        # Record the state (idle or running)
        self.core_state[core_id]['history'].append(current_state)

        # Limit the history size to the observation window
        if len(self.core_state[core_id]['history']) > self.observation_window:
            self.core_state[core_id]['history'].pop(0)

        # Predict the state based on history
        predicted_state = self._predict_core_state(core_id)

        # Set frequency based on predicted state
        frequency = self._set_frequency(predicted_state)

        return frequency

    def _predict_core_state(self, core_id):
        # A more sophisticated prediction: track transitions instead of just counts
        history = self.core_state[core_id]['history']

        # Track transitions to identify state changes
        if len(history) < 2:
            return 'running'  # Default to 'running' if not enough history

        # Check the most recent two states to identify transitions
        recent_states = history[-2:]
        if recent_states == ['idle', 'running']:
            return 'running'
        elif recent_states == ['running', 'idle']:
            return 'idle'

        # Fallback to simple count method if no recent transition detected
        if history.count('idle') > history.count('running'):
            return 'idle'
        return 'running'

    def _set_frequency(self, predicted_state):
        if predicted_state == 'idle':
            return 1.0  # Low frequency for idle state
        else:
            return 3.0  # High frequency for running state


# Global variables
PREDICTOR = CoreStatePredictor(sampling_period=200, observation_window=10)

# Callback function for branch prediction
def hook_branch_predict(ip, predicted, actual, indirect, core_id):
    # Get the current state from branch predictor information
    current_state = "running" if actual == predicted else "idle"

    # Make the prediction and set the frequency
    frequency = PREDICTOR.predict_state(core_id, current_state, ip, predicted, actual, indirect)

    # Log or set the core frequency in the simulation
    print(f"Core {core_id}: Predicted State = {current_state}, Set Frequency = {frequency} GHz")

    return frequency


# Hook the callback into Sniper
sim.util.EveryBranch(hook_branch_predict)
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/snipersim/scripts$
```

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

core_state_predictor.py | sim.out

```python
import sim.util, sim, os, sys

class CoreStatePredictor:
    def __init__(self, sampling_period, observation_window):
        self.sampling_period = sampling_period
        self.observation_window = observation_window
        self.core_state = {}
    def predict_state(self, core_id, current_state, pc, predicted, actual, indirect):
        # Ensure we have a history for this core
        if core_id not in self.core_state:
            self.core_state[core_id] = {'history': [], 'confidence': 0}

        # Record the state (idle or running)
        self.core_state[core_id]['history'].append(current_state)

        # Limit the history size to the observation window
        if len(self.core_state[core_id]['history']) > self.observation_window:
            self.core_state[core_id]['history'].pop(0)

        # Predict the state based on history
        predicted_state = self._predict_core_state(core_id)

        # Set frequency based on predicted state
        frequency = self._set_frequency(predicted_state)

        return frequency
    def _predict_core_state(self, core_id):
        # A more sophisticated prediction: track transitions instead of just counts
        history = self.core_state[core_id]['history']

        # Track transitions to identify state changes
        if len(history) < 2:
            return 'running'  # Default to 'running' if not enough history

        # Check the most recent two states to identify transitions
        recent_states = history[-2:]
        if recent_states == ['idle', 'running']:
            return 'running'
        elif recent_states == ['running', 'idle']:
            return 'idle'

        # Fallback to simple count method if no recent transition detected
        if history.count('idle') > history.count('running'):
            return 'idle'
        return 'running'

    def _set_frequency(self, predicted_state):
        if predicted_state == 'idle':
            return 1.0  # Low frequency for idle state
        else:
            return 3.0  # High frequency for running state

# Global variables
PREDICTOR = CoreStatePredictor(sampling_period=200, observation_window=10)

# Callback function for branch prediction
def hook_branch_predict(ip, predicted, actual, indirect, core_id):
    # Get the current state from branch predictor information
    current_state = "running" if actual == predicted else "idle"

    # Make the prediction and set the frequency
    frequency = PREDICTOR.predict_state(core_id, current_state, ip, predicted, actual, indirect)

    # Log or set the core frequency in the simulation
    print(f"Core {core_id}: Predicted State = {current_state}, Set Frequency = {frequency} GHz")
    return frequency
# Hook the callback into Sniper
sim.util.EveryBranch(hook_branch_predict)
```

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).

```
import sim.util, sim, os, sys
```

Apoi am creat o clasa numita CoreStatePredictor in cadrul careia avem un constructor (__init__) si metode (predict_state, _predict_core_state, _set_frequency).

```python
class CoreStatePredictor:
    def __init__(self, sampling_period, observation_window):
        self.sampling_period = sampling_period
        self.observation_window = observation_window
        self.core_state = {}

    def predict_state(self, core_id, current_state, pc, predicted, actual, indirect):
        # Ensure we have a history for this core
        if core_id not in self.core_state:
            self.core_state[core_id] = {'history': [], 'confidence': 0}

        # Record the state (idle or running)
        self.core_state[core_id]['history'].append(current_state)

        # Limit the history size to the observation window
        if len(self.core_state[core_id]['history']) > self.observation_window:
            self.core_state[core_id]['history'].pop(0)

        # Predict the state based on history
        predicted_state = self._predict_core_state(core_id)

        # Set frequency based on predicted state
        frequency = self._set_frequency(predicted_state)

        return frequency

    def _predict_core_state(self, core_id):
        # A more sophisticated prediction: track transitions instead of just counts
        history = self.core_state[core_id]['history']

        # Track transitions to identify state changes
        if len(history) < 2:
            return 'running'  # Default to 'running' if not enough history

        # Check the most recent two states to identify transitions
        recent_states = history[-2:]
        if recent_states == ['idle', 'running']:
            return 'running'
        elif recent_states == ['running', 'idle']:
            return 'idle'

        # Fallback to simple count method if no recent transition detected
        if history.count('idle') > history.count('running'):
            return 'idle'
        return 'running'

    def _set_frequency(self, predicted_state):
        if predicted_state == 'idle':
            return 1.0  # Low frequency for idle state
        else:
            return 3.0  # High frequency for running state
```

2.4.1 Constructorul clasei

Am inceput cu constructorul clasei:

```python
def __init__(self, sampling_period, observation_window):
    self.sampling_period = sampling_period
    self.observation_window = observation_window
    self.core_state = {}
```

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.

```
def predict_state(self, core_id, current_state, pc, predicted, actual, indirect):
    # Store the current state for prediction
    if core_id not in self.core_state:
        self.core_state[core_id] = {'history': [], 'confidence': 0}

    # Record the state (idle or running)
    self.core_state[core_id]['history'].append(current_state)

    # Apply a simple prediction based on the history
    if len(self.core_state[core_id]['history']) > self.observation_window:
        self.core_state[core_id]['history'].pop(0)

    # Predict state (based on history or branch predictor)
    predicted_state = self._predict_core_state(core_id)

    # Set frequency based on predicted state (simplified example)
    frequency = self._set_frequency(predicted_state)

    # Output prediction
    return frequency
```

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:

```
if core_id not in self.core_state:
    self.core_state[core_id] = {'history': [], 'confidence': 0}
```

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

```
# Record the state (idle or running)
self.core_state[core_id]['history'].append(current_state)

# Apply a simple prediction based on the history
if len(self.core_state[core_id]['history']) > self.observation_window:
    self.core_state[core_id]['history'].pop(0)
```

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 state (based on history or branch predictor)
predicted_state = self._predict_core_state(core_id)
```

_predict_core_state: este o metoda privata care alineaza istoricul starilor pentru a prezice urmatoarea stare

Setarea frecventei

```
# Set frequency based on predicted state (simplified example)
frequency = self._set_frequency(predicted_state)

# Output prediction
return frequency
```

_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

```
def _predict_core_state(self, core_id):
    # A more sophisticated prediction: track transitions instead of just counts
    history = self.core_state[core_id]['history']

    # Track transitions to identify state changes
    if len(history) < 2:
        return 'running'  # Default to 'running' if not enough history

    # Check the most recent two states to identify transitions
    recent_states = history[-2:]
    if recent_states == ['idle', 'running']:
        return 'running'
    elif recent_states == ['running', 'idle']:
        return 'idle'

    # Fallback to simple count method if no recent transition detected
    if history.count('idle') > history.count('running'):
        return 'idle'
    return 'running'
```

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

```
history = self.core_state[core_id]['history']
```

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

```
if len(history) < 2:
    return 'running'
```

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

```
recent_states = history[-2:]
if recent_states == ['idle', 'running']:
    return 'running'
elif recent_states == ['running', 'idle']:
    return 'idle'
```

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:

```
if history.count('idle') > history.count('running'):
    return 'idle'
return 'running'
```

2.4.4 Functia _set_frequency

```
def _set_frequency(self, predicted_state):
    if predicted_state == 'idle':
        return 1.0  # Low frequency
    else:
        return 3.0  # High 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

```
# Global variables
PREDICTOR = CoreStatePredictor(sampling_period=200, observation_window=10)
```

2.5 Functia hook_branch_predict

```
def hook_branch_predict(ip, predicted, actual, indirect, core_id):
    # Get the current state from branch predictor information
    current_state = "running" if actual == predicted else "idle"

    # Make the prediction and set the frequency
    frequency = PREDICTOR.predict_state(core_id, current_state, ip, predicted, actual, indirect)

    # Log or set the core frequency in the simulation
    print(f"Core {core_id}: Predicted State = {current_state}, Set Frequency = {frequency} GHz")

    return frequency


# Hook the callback into Sniper
sim.util.EveryBranch(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

```
# Get the current state from branch predictor information
current_state = "running" if actual == predicted else "idle"
```

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").

```
# Make the prediction
frequency = predictor.predict_state(core_id, current_state, ip, predicted, actual, indirect)
```

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).

```
# Log or set the core frequency in the simulation
print(f"Core {core_id}: Predicted State = {current_state}, Set Frequency = {frequency} GHz")

return frequency
```

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

```
# Hook the callback into Sniper
sim.util.EveryBranch(hook_branch_predict)
```

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

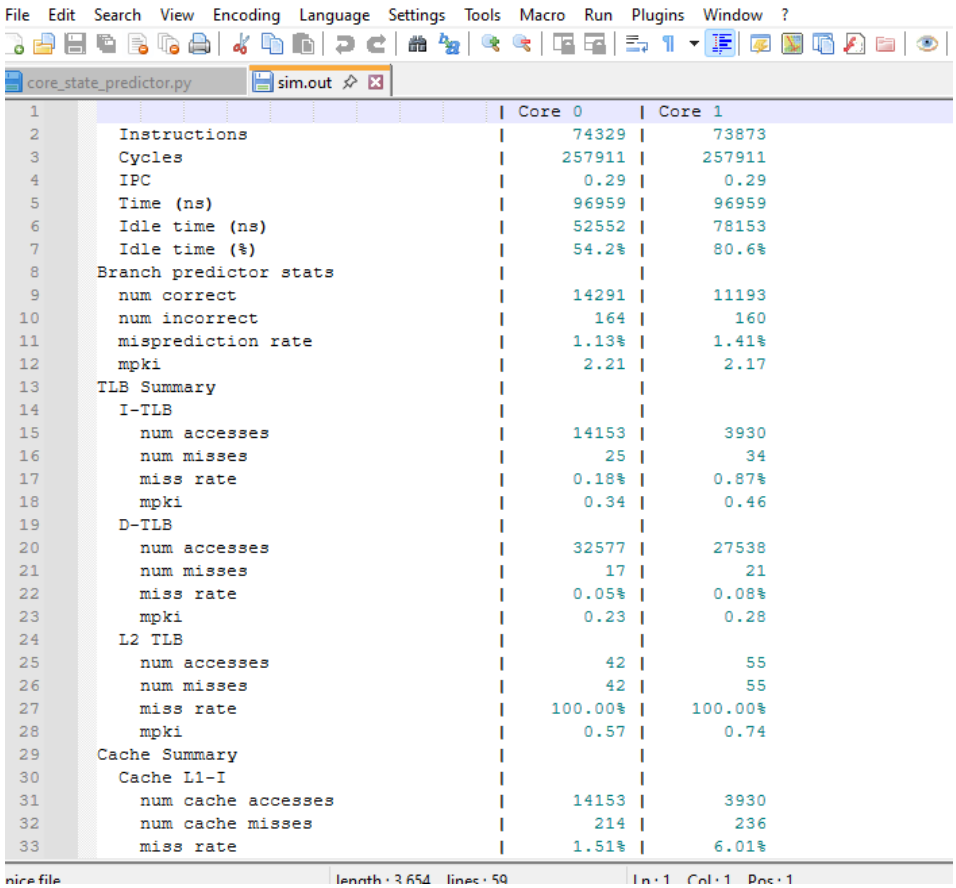
➔ /mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4/Splash-4/lu-contiguous_blocks/LU-CONT este calea catre programul care va fi rulat

➔ -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

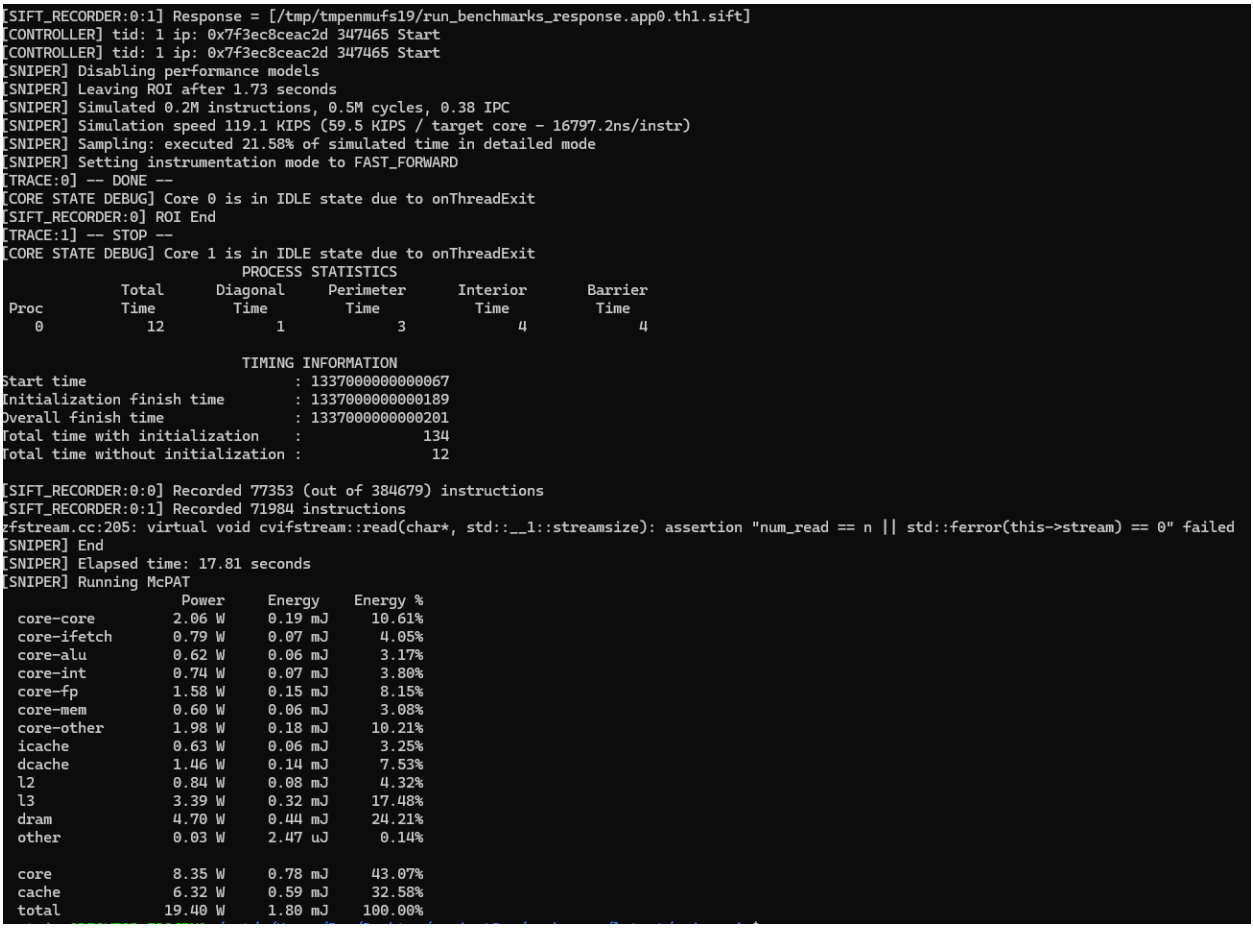
Mai jos se poate observa cum arata fisierul sim.out in interior



	Core 0	Core 1
Instructions	74329	73873
Cycles	257911	257911
IPC	0.29	0.29
Time (ns)	96959	96959
Idle time (ns)	52552	78153
Idle time (%)	54.2%	80.6%
Branch predictor stats		
num correct	14291	11193
num incorrect	164	160
misprediction rate	1.13%	1.41%
mpki	2.21	2.17
TLB Summary		
I-TLB		
num accesses	14153	3930
num misses	25	34
miss rate	0.18%	0.87%
mpki	0.34	0.46
D-TLB		
num accesses	32577	27538
num misses	17	21
miss rate	0.05%	0.08%
mpki	0.23	0.28
L2 TLB		
num accesses	42	55
num misses	42	55
miss rate	100.00%	100.00%
mpki	0.57	0.74
Cache Summary		
Cache L1-I		
num cache accesses	14153	3930
num cache misses	214	236
miss rate	1.51%	6.01%

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.



```
[SIFT_RECORDER:0:1] Response = [/tmp/tmpenmufs19/run_benchmarks_response.app0.th1.sift]
[CONTROLLER] tid: 1 ip: 0x7f3ec8ceac2d 347465 Start
[CONTROLLER] tid: 1 ip: 0x7f3ec8ceac2d 347465 Start
[SNIPER] Disabling performance models
[SNIPER] Leaving ROI after 1.73 seconds
[SNIPER] Simulated 0.2M instructions, 0.5M cycles, 0.38 IPC
[SNIPER] Simulation speed 119.1 KIPS (59.5 KIPS / target core - 16797.2ns/instr)
[SNIPER] Sampling: executed 21.58% of simulated time in detailed mode
[SNIPER] Setting instrumentation mode to FAST_FORWARD
[TRACE:0] -- DONE --
[CORE STATE DEBUG] Core 0 is in IDLE state due to onThreadExit
[SIFT_RECORDER:0] ROI End
[TRACE:1] -- STOP --
[CORE STATE DEBUG] Core 1 is in IDLE state due to onThreadExit
                        PROCESS STATISTICS
             Total      Diagonal     Perimeter      Interior       Barrier
 Proc        Time         Time         Time           Time          Time
    0           12            1            3              4             4

                        TIMING INFORMATION
Start time                    : 1337000000000067
Initialization finish time    : 1337000000000189
Overall finish time           : 1337000000000201
Total time with initialization    :             134
Total time without initialization :              12

[SIFT_RECORDER:0:0] Recorded 77353 (out of 384679) instructions
[SIFT_RECORDER:0:1] Recorded 71984 instructions
zfstream.cc:205: virtual void cvifstream::read(char*, std::__1::streamsize): assertion "num_read == n || std::ferror(this->stream) == 0" failed
[SNIPER] End
[SNIPER] Elapsed time: 17.81 seconds
[SNIPER] Running McPAT
                    Power     Energy    Energy %
  core-core         2.06 W     0.19 mJ    10.61%
  core-ifetch       0.79 W     0.07 mJ     4.05%
  core-alu          0.62 W     0.06 mJ     3.17%
  core-int          0.74 W     0.07 mJ     3.80%
  core-fp           1.58 W     0.15 mJ     8.15%
  core-mem          0.60 W     0.06 mJ     3.08%
  core-other        1.98 W     0.18 mJ    10.21%
  icache            0.63 W     0.06 mJ     3.25%
  dcache            1.46 W     0.14 mJ     7.53%
  l2                0.84 W     0.08 mJ     4.32%
  l3                3.39 W     0.32 mJ    17.48%
  dram              4.70 W     0.44 mJ    24.21%
  other             0.03 W     2.47 uJ     0.14%

  core              8.35 W     0.78 mJ    43.07%
  cache             6.32 W     0.59 mJ    32.58%
  total            19.40 W     1.80 mJ   100.00%
```

In interiorul folderului Spalsh-4:

```
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/snipersim$ cd ..
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest$ cd Splash-4/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4$ ll
total 24
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 ./
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:35 ../
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 .git/
-rwxrwxrwx 1 beatrice beatrice  959 Jan 13 19:35 .gitignore*
-rwxrwxrwx 1 beatrice beatrice  676 Jan 13 19:35 CHANGELOG*
-rwxrwxrwx 1 beatrice beatrice  217 Jan 13 19:35 Makefile*
-rwxrwxrwx 1 beatrice beatrice 1376 Jan 13 19:35 Makefile.config*
-rwxrwxrwx 1 beatrice beatrice 3060 Jan 13 19:35 README.md*
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 Splash-4/
-rwxrwxrwx 1 beatrice beatrice 6977 Jan 13 19:36 pthread.m4*
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4$ cd Splash-4/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4/Splash-4$ ll
total 0
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 ./
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 19:36 ../
-rwxrwxrwx 1 beatrice beatrice  369 Jan 13 19:35 Makefile*
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 barnes/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 cholesky/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 fft/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 fmm/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 lu-contiguous_blocks/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 lu-non_contiguous_blocks/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 ocean-contiguous_partitions/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 ocean-non_contiguous_partitions/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 radiosity/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 radix/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 raytrace/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 volrend/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 volrend-no_print_lock/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 water-nsquared/
drwxrwxrwx 1 beatrice beatrice 4096 Jan 13 22:26 water-spatial/
beatrice@DESKTOP-IBQJRU1:/mnt/c/Users/Bea/Desktop/proiectBea/workspace/latest/Splash-4/Splash-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.